NI-DAQmx User Manual

Contents

NI-DAQmx User Manual

The NI-DAQmx User Manual provides detailed descriptions of the product functionality and the step by step processes for use.

Looking for Something Else?

For information not found in the User Manual for your product, such as specifications and API reference, browse **Related Information**.

Related information:

- Download NI-DAQmx
- NI-DAQmx Release Notes
- License Setup and Activation
- Dimensional Drawings
- Product Certifications
- Letter of Volatility
- Discussion Forums
- NI Learning Center
- NI-DAQmx LabVIEW API Reference
- NI-DAQmx Properties Reference

New Features and Changes

Learn about updates—including new features and behavior changes—introduced in each version of the NI-DAQmx.

NI-DAQmx 2024 Q3.1 Changes

Learn about new features, behavior changes, and other updates in NI-DAQmx 2024 Q3.1.

New Features

Added support for the following NI mioDAQ devices.

- USB-6421
- USB-6423
- USB-6451
- USB-6453

NI-DAQmx 2024 Q3 Changes

Learn about new features, behavior changes, and other updates in NI-DAQmx 2024 Q3.

Behavior Changes

- Support for VB6 has been removed.

NI-DAQmx 2024 Q2 Changes

Learn about new features, behavior changes, and other updates in NI-DAQmx 2024 Q2.

Behavior Changes

NI-DAQmx support for VB6 is expected to be removed in NI-DAQmx 2024 Q3. NI-DAQmx

2024 Q2 will be the last release to support VB6.

NI-DAQmx 2022 Q4 Changes

Learn about new features, behavior changes, and other updates in DAQmx 2022 Q4.

New Features

Support has been added for the following devices:

- TS-15050 DIO P0
- TS-15100
- TS-15110
- TS-15120
- TS-15130
- TS-15200

NI-DAQmx Overview

What is DAQmx?

NI-DAQmx is the driver software you use to communicate with and control your NI data acquisition (DAQ) devices. It includes an extensive library of functions and VIs you can call from your application software, such as LabVIEW or LabWindows/CVI, to program your devices.

For information on getting started, refer to Getting Started with NI-DAQmx. Refer to Device Support in NI-DAQmx in the NI-DAQmx Readme for a list of devices supported in NI-DAQmx.

What is Measurement & Automation Explorer (MAX)?

MAX is an application that automatically installs with the NI-DAQmx driver. MAX informs other programs which devices you have in your system and how they are configured. With MAX, you can:

- Configure your NI hardware and software
- Create and edit channels, tasks, interfaces, scales, and virtual instruments
- Execute system diagnostics
- View devices and instruments connected to your system
- Update your NI software

For more information, refer to the Measurement & Automation Explorer Help for NI-DAQmx or Getting Started.

What is DAQ Assistant?

The DAQ Assistant is an application that automatically installs with the NI-DAQmx driver. You can launch the DAQ Assistant from MAX, or from your NI application software such as LabVIEW, SignalExpress, LabWindows/CVI, or Measurement Studio. With DAQ Assistant you can:

- Create and edit tasks and virtual channels
- Add virtual channels to tasks
- Create and edit scales
- Test your configuration
- Save your configuration
- Generate code in your NI application software for use in your application
- View connection diagrams for your sensors

For more information, refer to the DAQ Assistant Help or Getting Started.

How does DAQmx Work Together with Other Applications and Text Based Programming Environments?

DAQmx has an application programming interface (API), which is a library of VIs, functions, classes, attributes, and properties for creating applications for your device. For information on getting started, refer to Getting Started. For a brief description of using DAQmx with these applications, refer to the following topics.

- LabVIEW
- LabWindows/CVI
- Measurement Studio with Visual C++, Visual C#, or Visual Basic .NET
- ANSI C Application without LabWindows/CVI
- .NET Application without Measurement Studio
- SignalExpress

Related concepts:

- Getting Started with NI-DAQmx
- LabVIEW
- Creating an Application with LabWindows/CVI
- Measurement Studio with Visual C++, Visual C#, or Visual Basic .NET
- ANSI C Application without LabWindows/CVI
- Creating a .NET Application without Measurement Studio
- SignalExpress
- Finding Examples
- Troubleshooting

Related information:

- Using the DAQ Assistant in Measurement Studio

LabVIEW

If you program your NI-DAQmx-supported device in LabVIEW, you can interactively create virtual channels—both global and local—and tasks by launching the DAQ Assistant from MAX or from within LabVIEW. You also can create local virtual channels and tasks, and write your own applications using the NI-DAQmx API.

To learn about which NI-DAQmx VIs are most commonly used when creating a NI-DAQmx data acquisition application, see Learn 10 Functions in NI-DAQmx and Handle 80% of your Data Acquisition Applications.

For help with NI-DAQmx VIs, refer to DAQmx - Data Acquisition VIs and Functions.

For general help with programming in LabVIEW, refer to LabVIEW Help.

For help with using the DAQ Assistant with LabVIEW, refer to Using the DAQ Assistant to Automatically Generate LabVIEW Code.

Creating an Application with LabWindows/CVI

If you program your NI-DAQmx-supported device in LabWindows/CVI, you can interactively create global or local virtual channels and tasks by launching the DAQ Assistant from MAX or from within LabWindows/CVI. You can generate the configuration code based on your task or channel in LabWindows/CVI. Refer to the DAQ Assistant Help for additional information about generating code. You also can create local virtual channels and tasks, and write your own applications using the NI-DAQmx API.

For help with NI-DAQmx functions, refer to NI-DAQmx C Function Reference Help. For general help with programming in LabWindows/CVI, refer to LabWindows/CVI Help, accessible through **Start»All Programs»National Instruments»LabWindows CVI»LabWindows CVI Help**.

For help with using the DAQ Assistant with LabWindows/CVI, refer to Using the DAQ Assistant in NI LabWindows/CVI.

Related information:

- Using the DAQ Assistant in NI LabWindows™/CVI™

Measurement Studio with Visual C++, Visual C#, or Visual Basic .NET

If you program your NI-DAQmx-supported device in Measurement Studio using Visual C++, Visual C#, or Visual Basic .NET, you can interactively create channels and tasks by launching the DAQ Assistant from MAX or from within Visual Studio .NET. You can generate the configuration code based on your task or channel in Measurement Studio. Refer to the DAQ Assistant Help for additional information about generating code. You also can create channels and tasks, and write your own applications in your ADE using the NI-DAQmx API.

For help with NI-DAQmx methods and properties, refer to the NI-DAQmx .NET Class Library or the NI-DAQmx Visual C++ Class Library included in the NI Measurement Studio Help. For general help with programming in Measurement Studio, refer to the NI Measurement Studio Help, which is fully integrated with the Microsoft Visual Studio .NET help. To view this help file in Visual Studio. NET, select **Measurement Studio»NI Measurement Studio Help**.

For help with using the DAQ Assistant with Measurement Studio, refer to Using the DAQ Assistant in Measurement Studio.

Related information:

- Using the DAQ Assistant in Measurement Studio

ANSI C Application without LabWindows/CVI

NI-DAQmx has a C API that you can use to create applications. To create an application, follow these general steps:

1. Create a new project.
2. Open existing or new source files (`.c`), and add them to the project. Make sure you include the NI-DAQmx header file, `nidaqmx.h`, in your source code files. You can find this header file at `NI-DAQ\DAQmx ANSI C Dev\include`.
3. Add the NI-DAQmx import library, `nidaqmx.lib`, to the project. The import library files are located under `NI-DAQ\DAQmx ANSI C Dev\lib\`.
4. To view examples of NI-DAQmx applications, go to the `NI-DAQ\Examples\ DAQmx ANSI C` directory.
5. Build your application.

For help with NI-DAQmx functions, refer to the NI-DAQmx C Reference Help, which is installed by default at **Start»All Programs»National Instruments»NI-DAQ»Text-Based Code Support»NI-DAQmx C Reference Help**.

For help with using the DAQ Assistant with ANSI C, refer to Using NI-DAQmx in Text Based Programming Environments.

Creating a .NET Application without Measurement Studio

With the Microsoft .NET Framework version 1.1 or later, you can use NI-DAQmx to create applications using Visual C# and Visual Basic .NET without Measurement Studio. You need at least Microsoft Visual Studio .NET 2003 or Microsoft Visual Studio 2005 for the API documentation to be installed.

The installed documentation contains the NI-DAQmx API overview, measurement tasks and concepts, and function reference. This help is fully integrated into the Visual Studio .NET documentation. To view the NI-DAQmx .NET documentation, go to **Start»All Programs»National Instruments»NI-DAQ»NI-DAQmx .NET Reference Help**. Expand **NI Measurement Studio Help»Measurement Studio Support for NI-DAQmx Overview» NI-DAQmx .NET class library** to view the function reference. Expand **NI Measurement Studio Help»NI Measurement Studio .NET Class Library»Using the Measurement Studio .NET Class Libraries** to view conceptual topics for using NI-DAQmx with Visual C# and Visual Basic .NET.

To get to the same help topics from within Visual Studio, go to **Help»Contents**. Select **Measurement Studio** from the **Filtered By** drop-down list and follow the previous instructions.

For help with using the DAQ Assistant with Visual Studio .NET, refer to Using NI-DAQmx in Text Based Programming Environments.

SignalExpress

If you use your NI-DAQmx-supported device in SignalExpress, you can create a project that includes NI-DAQmx steps. With SignalExpress, you can log and analyze data. You can also add global virtual channels that you created in MAX to your NI-DAQmx steps in SignalExpress. Refer to the DAQ Assistant Help for additional information.

For help with using the DAQ Assistant with SignalExpress, refer to Taking an NI-DAQmx Measurement in SignalExpress. For general help with programming in SignalExpress, refer to SignalExpress Help.

Related information:

- Using the DAQ Assistant in Measurement Studio

Getting Started with NI-DAQmx

The National Instruments Getting Started with NI-DAQmx Series on ni.com is aimed at helping you learn NI-DAQmx programming fundamentals. Through video and text tutorials, this series will take you from verifying your device's operation in Measurement & Automation Explorer (MAX) to programming data acquisition applications using LabVIEW. It is intended for both the beginner who wants to learn how to use the DAQ Assistant, as well as the experienced user who wishes to take advantage of advanced NI-DAQmx functionality.

Go to Getting Started with NI-DAQmx: Main Page on ni.com.

The series includes:

- NI-DAQmx Frequently Asked Questions
- NI-DAQmx Software and Hardware Installation
- Basic Programming with NI-DAQmx
- Advanced Programming with NI-DAQmx

Related concepts:

- NI-DAQmx Overview
- Finding Examples
- Troubleshooting

Finding Examples

Each API includes a collection of programming examples to help you get started developing an application. You can modify example code and save it in an application. You can use examples to develop a new application or add example code to an existing application.

To run examples without hardware installed, you can use an NI-DAQmx simulated device. In MAX, refer to the Measurement & Automation Explorer Help for NI-DAQmx by

selecting **Help»Help Topics» NI-DAQmx** for information on NI-DAQmx simulated devices.

To find the locations of examples for your software application, refer to the following table.

Software Application	Example Location
LabVIEW or LabWindows/CVI	**Help»Find Examples**
SignalExpress	`Program Files\National Instruments\` `SignalExpress\Examples`
ANSI C	`*...NI-DAQ\Examples\DAQmx ANSI C`
MFC 7.0 C++	`*...NI-DAQ\Examples\MStudioVC2003`
Visual Basic .NET and C# for Visual Studio 2003	`*...NI-DAQ\Examples\DotNET1.1`
MFC 8.0 C++	`*...NI-DAQ\Examples\MStudioVC2005`
MFC 9.0 C++	`*...NI-DAQ\Examples\MStudioVC2008`
Visual Basic .NET and C# for Visual Studio 2005	`*...NI-DAQ\Examples\DotNET2.0`
Visual Basic .NET and C# for Visual Studio 2008	`*...NI-DAQ\Examples\DotNET3.5`

Note * For Windows, the default path is `<drive>:\Users\Public\`
`Public Documents\National Instruments\NI-DAQ\`
`Examples\....`

Note Visual Studio 2003 and later do not require Measurement Studio.

Troubleshooting

Installation and Configuration

Refer to the DAQ Getting Started Guide for general installation and configuration

instructions.

Use the following resources if you have problems installing your DAQ hardware and/or software:

- Refer to NI DAQ Setup and Support on `ni.com` for information on getting started with NI DAQ, support, drivers and code, and other resources.
- Refer to `ni.com/kb` for documents on troubleshooting common installation and programming problems and for answering frequently asked questions about NI products.
- If you think you have damaged your device and need to return your NI hardware for repair or calibration, refer to sending a NI board in for repair or calibration.

For LabWindows/CVI users, if the Data Acquisition function panel is disabled, you may need to uninstall NI-DAQmx and reinstall it, making sure that you add support for LabWindows/CVI. If you have installed LabWindows/CVI support and Data Acquisition is still dimmed, select **Library»Customize**. In the **Customize Library Menu** dialog box, check **Data Acquisition**, and restart LabWindows/CVI. You might also need to verify that the `dataacq.lib` is in the `bin` directory.

Programming

To help you get started programming, you can use the shipping examples for your ADE.

You can also visit NI's extensive library of technical support resources at `ni.com/support`.

You can interactively configure global virtual channels and tasks with the DAQ Assistant. For NI application software such as LabVIEW, you can use the DAQ Assistant to generate code.

You can use NI I/O Trace to analyze the functions you have called in the API with NI application software. With NI I/O Trace, you can watch the order of execution of the application and locate errors as they happen.

> **Note** I/O Trace is not supported by the NI-DAQmx .NET API.

Finally, the NI-DAQmx Help contains programming flowcharts for common applications such as measuring temperature, current, strain, position, and acceleration.

External Connections

In addition to the information on making signal connections in this help file, the Connection Diagram tab in the DAQ Assistant within MAX shows you how to connect signals.

Calibration

- For information on externally calibrating your device, including step-by-step calibration procedures, refer to ni.com/calibration.
- For an overview of calibration, including the difference between self-calibration and external calibration, refer to Device Calibration.
- For device-specific information required for calibration with NI-DAQmx, refer to Device-Specific Calibration.
- For information on channel calibration, refer to What Is Channel Calibration?

CPU Usage

NI-DAQmx tasks use 100% of the CPU if no other processes are running. However, as soon as another process requires the CPU, the NI-DAQmx task yields to that process.

Related concepts:

- Finding Examples

Common Applications

Measuring Acceleration
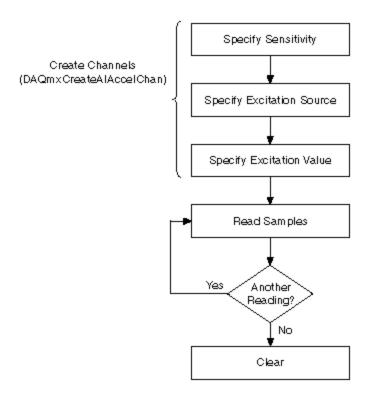
Acceleration is a change in velocity with respect to time. An accelerometer is a transducer that represents acceleration as a voltage. Accelerometers also can measure vibration and shock. Accelerometers typically convert acceleration measured in g's to voltage. For example, a sensor with a rated output of 10 mV/g should produce 50 mV when subjected to 5 g of acceleration.

Related concepts:

- Measuring Acceleration Programming Flowchart
- Finding Examples

Measuring Acceleration Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to measure acceleration. Alternatively, you can configure a task for measuring acceleration using the DAQ Assistant.

> 💡 **Tip** To increase performance, especially when multiple samples are read, include the Start function/VI and Stop function/VI in your application. In the previous flowchart, the Start function/VI would come just before you read samples, and Stop would come just before you clear the task.

Measuring acceleration is an example of an analog input measurement. Refer to Analog Input Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- Analog Input Programming Flowcharts

Measuring Analog Frequency

Some devices can measure analog frequency directly using frequency-to-voltage circuitry. Many devices, however, only measure voltage, and you must use software algorithms to convert those measurements to frequency.

Devices that measure analog frequency, such as DSA devices and the SCXI-1126, have circuitry that produces triggers of the same frequency as the measured signal. Every time the signal passes from threshold level minus hysteresis to threshold level, a trigger occurs. A pulse generator uses these triggers and produces a pulse once every frequency cycle. The input frequency range sets the width of this pulse. As the input frequency range increases, the pulse width grows smaller. This pulse train is then converted to a DC signal that has a level proportional to the duty cycle of the pulse train. The duty cycle is the fraction of a period of the pulse train when the pulse is occurring. The DC signal has a voltage that is proportional to the input frequency and can therefore be scaled to that frequency value.

For devices that cannot measure frequency directly, you need to use software algorithms, such as the Fast Fourier Transform (FFT), to convert voltage to frequency. LabVIEW Full and Professional Development Systems contains advanced analysis VIs that handle these transformations. The LabWindows™/CVI™ full development system also contains advanced analysis functions to help you measure analog frequency. Regardless of whether you use existing VIs or functions or create your own, you need to sample at least twice as fast as the highest frequency component in the signal you are acquiring.

Analog Frequency, Sample Rate, and the Nyquist Theorem

The Nyquist Theorem states that the highest frequency you can accurately represent is half the sampling rate. For instance, to measure the frequency of a 100 Hz signal, you need a sampling rate of at least 200 S/s. In practice, you should use sampling rates of 5 to 10 times the expected frequencies to improve accuracy of measurements.

In addition to sample rate, you need to determine the number of samples to acquire. You must sample a minimum of three cycles of the analog signal. For example, you need to collect at least 15 samples, or points, if you use a sampling rate of 500 S/s to measure the frequency of a 100 Hz signal. Because you sample about five times faster than the signal frequency, you sample about five points per cycle of the signal. You need data from three cycles, so 5 points x 3 cycles = 15 points. In practice, however, you should acquire 10 or more cycles to improve accuracy of measurements, so you should acquire 50 or more samples.

The number of points you collect determines the number of frequency bins that the samples fall into. The size of each bin is the sampling rate divided by the number of

points you collect. For example, if you sample at 500 S/s and collect 100 points, you have bins at 5 Hz intervals.

The Nyquist frequency is the bandwidth of the sampled signal and is equal to half the sampling frequency. Frequency components below the Nyquist frequency appear normally. Frequency components above the Nyquist frequency appear aliased between 0 and the Nyquist frequency. The aliased component is the absolute value of the difference between the actual component and the closest integer multiple of the sampling rate. For example, if you have a signal with a component at 800 Hz and you sample at 500 S/s, that component appears aliased at 200 Hz because |800–(2 x 500)| = 200(Hz).
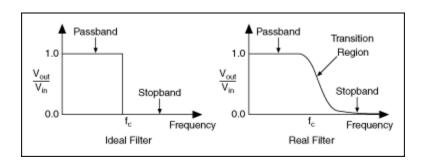
One way to eliminate aliased components is to use an analog hardware filter before you digitize and analyze the frequency information. If you want to perform all the filtering in software, you must first sample at a rate fast enough to correctly represent the highest frequency component the signal contains. For example, with the highest component at 800 Hz, the minimum sampling rate is 1,600 Hz, but you should sample 5 to 10 times faster than 800 Hz. If the frequency you want to measure is around 100 Hz, you can use a lowpass Butterworth filter with a cutoff frequency (f_c) of 250 Hz to filter out frequencies above 250 Hz and pass frequencies below 250 Hz.

> **Note** LabVIEW includes Butterworth filters with the LabVIEW Full and Professional Development Systems.

Measuring Frequency with Filtering

The following figure shows a lowpass filter.

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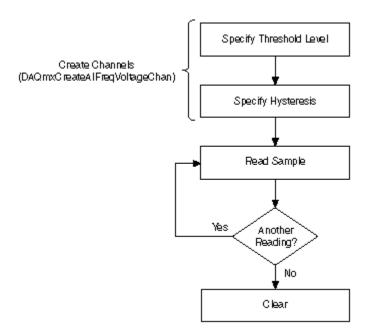
The Ideal Filter in the figure is optimal. All frequencies above the Nyquist frequency are rejected. The Real Filter in the figure is what you might actually be able to accomplish with a Butterworth filter. The passband is where V_{out}/V_{in} is close to 1. The stopband occurs where V_{out}/V_{in} is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

Related concepts:

- Measuring Analog Frequency Programming Flowchart
- Finding Examples

Measuring Analog Frequency Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to measure analog frequency. Alternatively, you can configure a task for measuring analog frequency using the DAQ Assistant.

Tip To increase performance, especially when multiple samples are read, include the Start function/VI and Stop function/VI in your application. In the previous flowchart, the Start function/VI would come just before you read

> samples, and Stop would come just before you clear the task.

Measuring frequency is an example of analog input measurement. Refer to Analog Input Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- Analog Input Programming Flowcharts

Hysteresis with Analog Frequency Measurements

For waveform repetitions, hysteresis adds a window below the threshold level. Hysteresis is typically used to avoid erroneous measurements due to noise or jitter in the signal. The signal must drop below the threshold level minus the hysteresis before NI-DAQmx recognizes a waveform repetition at the threshold level.

Measuring Angular Displacement

Angular displacement is movement around an axis, such as the angular motion of the shaft of a motor. An angular displacement sensor is a device whose output signal represents the rotation of the shaft; it cannot measure the physical displacement of the whole shaft. One type of sensor used to measure angular displacement is a rotary variable differential transformer (RVDT). Another type of sensor used to measure angular displacement is a resolver, which is a rotating transformer that can measure 360° of rotation.

On M Series devices, C Series devices, and NI-TIO-based devices, you can use the counters to perform displacement measurements with quadrature encoders, or angular encoders. You can measure angular position with X1, X2, and X4 angular encoders. You can choose to do either a single-point or a buffered sample clock

displacement measurement.

You also can measure velocity with angular encoders, but you need to use a sample clock with a fixed frequency. To measure velocity, use the following formula:

$V = D/T$

where V is the average velocity, D is the distance, and T is time.

The counter measures the position of the encoder using the A and B signals, which are offset by 90°. The counter also supports the Z index, which provides a precise reference point and is available on some encoders.

Related concepts:

- Measuring Position with an RVDT or LVDT Programming Flowchart
- Measuring Position with Encoders Programming Flowchart
- Finding Examples

Measuring Position with Encoders Programming Flowchart

The following flowchart depicts the main steps you must complete for measuring position with an encoder in an NI-DAQmx application. If you prefer, you can configure a task using the DAQ Assistant.

Measuring position with an encoder is an example of a counter measurement. Refer to Counter Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- Counter Programming Flowcharts

Measuring Position with an RVDT or LVDT Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to measure position with an RVDT or LVDT. Alternatively, you can configure a task for measuring position using the DAQ Assistant.

> 💡 **Tip** To increase performance, especially when multiple samples are read, include the Start function/VI and Stop function/VI in your application. In the previous flowchart, the Start function/VI would come just before you read samples, and Stop would come just before you clear the task.

Measuring position is an example of an analog input measurement. Refer to Analog Input Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- Analog Input Programming Flowcharts

Control

You can create an event response or control loop application in any operating system supported by NI-DAQmx. However, your application can only be deterministic if you have the LabVIEW Real-Time module and use your application on a real-time controller. This section assumes that you are using LabVIEW with NI-DAQmx to create a control application. It does not assume that you have the LabVIEW Real-Time module or the real-time controller.

Related concepts:

- Control Loops

Event Response

In a control application, an event is the same as an occurrence. This occurrence leads to an action, or a response. An example is monitoring the temperature of an engine. When the temperature rises too high, the engine slows down. The event, in this case, would be the temperature rising above a predetermined level, and the response would be the engine slowing down. Another example comes from manufacturing. In a manufacturing line, a system senses when a part is in front of a station (the event) and takes a reading or manipulates the part (the response). If the system does not sense and respond to the presence of that part in a set amount of time, the manufacturing line creates defective parts.

When creating an event response application, make sure you consider the amount of time needed to respond to the event. For example, if the device controls the temperature of your home, the time to react to events (changes in temperature) is less critical than if the device controls a nuclear reactor. If the application is not time critical, the application does not need to be deterministic, meaning that you do not need the LabVIEW Real-Time Module or a real-time controller.

The relative priority of the task is important as well. Because LabVIEW is multi-threaded, you can separate the application into tasks, each with its own priority. By setting priorities, time-critical tasks can take precedence over non-time-critical tasks. The time-critical task must periodically yield processor resources to the lower-priority tasks so they can execute. By properly separating the time-critical task from lower priority tasks, you can reduce application jitter. Refer to the **LabVIEW Real-Time Module Concepts** book in the LabVIEW Help for more information about assigning

priorities to tasks.

Related concepts:

- [Finding Examples](#)
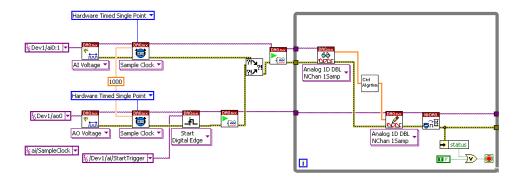
Control Loops

A control application monitors and controls a system. The application continuously loops by reading samples, processing data, and adjusting the output. You can use NI-DAQmx and DAQ devices to create a control application. With the LabVIEW Real-Time Module, you can create deterministic control applications.

Creating a Control Loop Application with NI-DAQmx

The following block diagram shows a typical deterministic control loop application. First, an analog value is read. This value corresponds to the process variable. This value is compared to the set point, which is specified in the Ctrl Algrthm VI in the diagram, and adjusted as necessary within the while loop, possibly using a PID algorithm. The adjusted value is then written. This value corresponds to the actuator output.

In the block diagram, the sampling rates are the same for analog input and output. Because the example shown assumes a single DAQ device, the Start Trigger synchronizes the analog input and analog output tasks. For multiple devices, synchronization works differently. Refer to Synchronization for more information. Notice also that the slave task—the analog output task—starts before the analog input task. Finally, within the loop, the Wait for Next Sample Clock VI checks to make sure that the loop executes within the specified sampling rate. If it does not, this VI returns an error.

Related concepts:
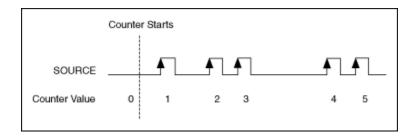
- Finding Examples
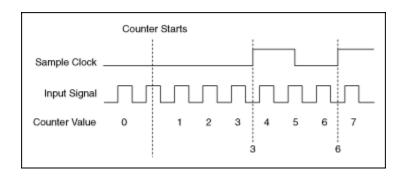
Edge Counting

Edge counting is when a device counts rising or falling edges using a counter channel. You can choose to do either single point or buffered sample clock edge counting.

The following figure shows an example of edge counting in which the counter in a device counts five edges on the input terminal.

With buffered edge counting, the device latches the number of edges counted onto each active edge of the sample clock and stores the number in the buffer. There is no built-in clock for buffered edge counting, so you must supply an external sample clock.

In NI-DAQmx, when doing on-demand edge counting, you first arm the counter by calling the Start function/VI. Each subsequent read returns the number of edges counted since the counter was started. If you perform multiple reads without first starting the counter, the counter implicitly starts and stops with each Read function/VI call, and the number of counted edges is not cumulative between read calls.

On devices that support gate configuration along with timing engine pause triggering, use the CI.CountEdges.Gate.Enable attribute/property to enable gate functionality.
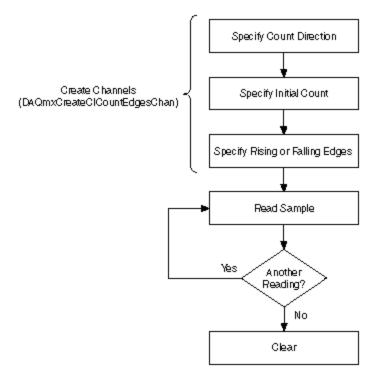
With the exception of the NI 9361, you also can pause counting with on-demand edge counting in NI-DAQmx by configuring a pause trigger. To configure a pause trigger, use the trigger attributes/properties to set the source terminal of the digital trigger as well as the level on which to pause.

Related concepts:

- Edge Counting Programming Flowchart
- Finding Examples

Edge Counting Programming Flowchart

The following flowchart depicts the main steps you must complete for counting edges in an NI-DAQmx application. If you prefer, you can configure a task for counting edges using the DAQ Assistant.

Edge counting is an example of a counter measurement. Refer to Counter Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- Counter Programming Flowcharts

Measuring Charge
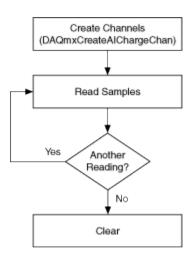
Electrical charge is a fundamental property of matter. Charge is a measurement of the net effect of protons and electrons being unequally distributed. Piezoelectric transducers produce a charge from physical stress, deformation, acceleration, or force. Charge is measured in coulombs.

Related concepts:

- Measuring Charge Programming Flowchart
- Finding Examples

Measuring Charge Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to measure charge. Alternatively, you can configure a task for measuring charge using the DAQ Assistant.

> **Tip** To increase performance, especially when multiple samples are read, include the Start function/VI and Stop function/VI in your application. In the previous flowchart, the Start function/VI would come just before you read samples, and Stop would come just before you clear the task.

You can use charge sensors to measure physical phenomenon such as acceleration and sound pressure with charge mode accelerometers and microphones. These measurements can be converted from coulombs to the appropriate engineering units using custom scales.

Measuring charge is an example of analog input measurement. Refer to Analog Input Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- Analog Input Programming Flowcharts

Measuring and Generating Current

Many measurement devices can measure and generate current. To measure or generate current with a DAQ device, you need a resistor. Current then can be measured through an analog input connector or generated through an analog output connector. The resistance must be placed in parallel with the connector and the current source. To measure voltage dropped across the resistor and convert it to current, use Ohm's Law.
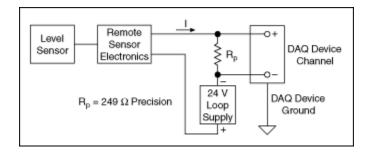
$$I_{(A)} = V_{(V)} / R_{(\Omega)}$$

where I is the current, V is the voltage, and R is the resistance.

4 to 20 mA Loops

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

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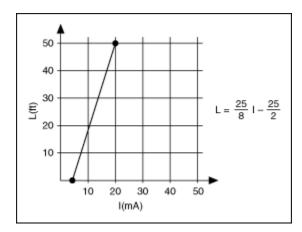
Current Loop Wiring

The DAQ device reads the voltage drop across the 249 Ω resistor R_p, using Ohm's Law.

Because the current is 4-20 mA and R_p is 249 Ω, V ranges from 0.996 V to 4.98 V, which is within the range that DAQ devices can read. Although the equation is useful for

calculating the current, the current typically represents a physical quantity you want to measure. In the following figure, the tank level measures 0 to 50 feet. 4 mA represents 0 feet, and 20 mA represents 50 feet. L is the tank level, and I is the current.

Linear Relationship between Tank Level and Current

Using the Ohm's Law equation and substituting 0.249 for the value of R_p, you can derive L in terms of measured voltage:
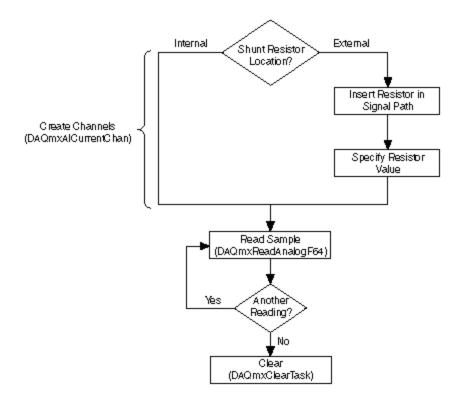
$$L = \frac{25 \times V}{8 \times 0.249} - \frac{25}{2}$$

Related concepts:

- Measuring Current Programming Flowchart
- Generating Current Programming Flowchart
- Finding Examples
- Tips on Measuring AC Current

Measuring Current Programming Flowchart

The following flowchart illustrates the main steps required in an NI-DAQmx application to measure current. Alternatively, you can configure a task for measuring current using the DAQ Assistant.

Tip To increase performance, especially when multiple samples are read, include the Start function/VI and Stop function/VI in your application. In the previous flowchart, the Start function/VI would come just before you read samples, and Stop would come just before you clear the task.

Measuring current is an example of an analog input measurement. Refer to Analog Input Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- Analog Input Programming Flowcharts
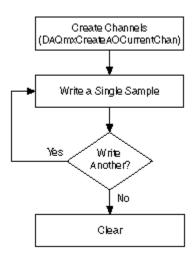
Tips on Measuring AC Current

To measure AC current, insert a precisely calibrated, low-value resistor into the signal path and measure the voltage drop across the resistor. You must then perform high-pass filtering on the resulting signal to remove the DC component. You can perform this filtering using an analog filter or digital signal processing techniques, such as the

filtering tools in the analysis library of LabVIEW.

Generating Current Programming Flowchart

The following flowchart illustrates the main steps required in an NI-DAQmx application to generate current. Alternatively, you can configure a task for generating current using the DAQ Assistant.

> **Tip** To increase performance, especially when multiple samples are written, include the Start function/VI and Stop function/VI in your application. In the previous flowchart, the Start function/VI would come just after you write samples, and Stop would come just before you clear the task.

Generating current is an example of an analog output measurement. Refer to Analog Output Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- Analog Output Programming Flowcharts

Measuring and Generating Digital Values

Signals that are read or measured are called input signals. Those signals that are

generated are called output, or standard output. Some specialized devices also support a Wired-OR output. Refer to the device documentation for more information about the types of input and output the device supports.

This section covers software-timed digital input/output operations—or unstrobed operations. These signals are controlled by software timing.
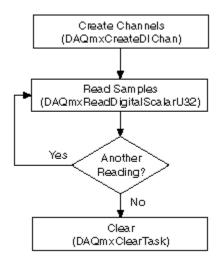
Measuring and generating digital values are used in a number of applications, including controlling relays and monitoring alarm states. Generally, measuring and generating digital values is used in laboratory testing, production testing, and industrial process monitoring and control.

Related concepts:

- [Measuring a Digital Value Programming Flowchart](#)
- [Generating a Digital Value Programming Flowchart](#)
- [Finding Examples](#)
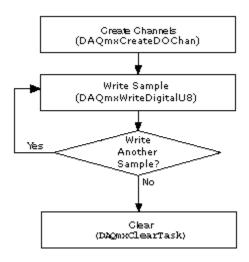
Measuring a Digital Value Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to measure digital values. If you prefer, you can configure a task for acquiring digital values using the DAQ Assistant.

Generating a Digital Value Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to generate digital values. If you prefer, you can configure a task for generating digital values using the DAQ Assistant.
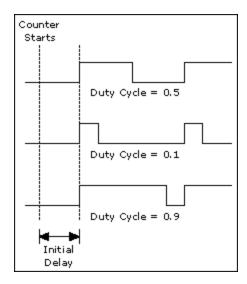
Measuring Duty Cycle

You can use the counters on your measurement device to measure duty cycle. Duty Cycle measurements measure the active time of a signal. Use the following equation to calculate the duty cycle of a pulse:

Duty Cycle = High Time/Pulse Period

where Pulse Period is high time plus low time.

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

Creating a Program

Measuring Pulses Programming Flowchart

Related concepts:

- Measuring Pulses Programming Flowchart

Measuring Pulses Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to measure the frequency/duty cycle, high/low ticks, or high/low time of digital pulses. Alternatively, you can configure a task for measuring digital pulses using the DAQ Assistant.

Digital frequency and period are examples of counter measurements. Refer to Counter Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- Counter Programming Flowcharts

Measuring Force

Force is an influence that changes the motion, size, or shape of an object. Many types of sensors exist for measuring force. Some are piezoelectric sensors, typically used for dynamic force measurements, such as impact testing. Others are bridge-based sensors, typically used for measuring static or slow-changing loads.

> **Note** NI-DAQmx supports only IEPE force and bridge-based sensors.
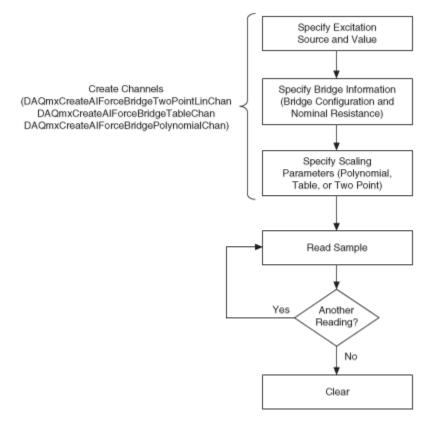
Related concepts:

- Measuring Force with a Piezoelectric Sensor Programming Flowchart
- Measuring Force with a Bridge-Based Sensor Programming Flowchart
- Finding Examples

Measuring Force with a Bridge-Based Sensor Programming

Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to measure force with a bridge-based sensor. Alternatively, you can configure a task for measuring force using the DAQ Assistant.

> 💡 **Tip** To increase performance, especially when multiple samples are read, include the Start function/VI and Stop function/VI in your application. In the previous flowchart, the Start function/VI would come just before you read samples, and Stop would come just before you clear the task.

When selecting the scaling type, choose the one that best matches the specifications for your sensor.

Measuring force is an example of analog input measurement. Refer to Analog Input Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- Analog Input Programming Flowcharts

Measuring Force with a Piezoelectric Sensor Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to measure force with a piezoelectric force sensor. Alternatively, you can configure a task for measuring force using the DAQ Assistant.

> 💡 **Tip** To increase performance, especially when multiple samples are read, include the Start function/VI and Stop function/VI in your application. In the previous flowchart, the Start function/VI would come just before you read samples, and Stop would come just before you clear the task.
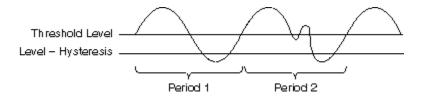
Measuring force is an example of analog input measurement. Refer to Analog Input Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- [Analog Input Programming Flowcharts](#)

Measuring Digital Frequency

The digital frequency of a signal is the inverse of the period of a signal. To get the frequency of the signal, take the inverse of the period. The formula for frequency is Frequency (in Hz) = Counter Timebase Rate (in Hz) / Count.

The Counter Timebase Rate is a known frequency and is usually a built-in time source. If the counter timebase rate is unknown, you only can make measurements only in terms of ticks of the counter timebase. This may be the case if you are using an external signal for the counter timebase, and the frequency of the external signal is unknown or aperiodic.

Digital frequency is an example of a time measurement. Refer to Configuring a Time Measurement in NI-DAQmx and Two Counter Measurement Method for more information about measuring time.

Related concepts:

- [Measuring Digital Frequency and Period Programming Flowchart](#)
- [Finding Examples](#)

Measuring Pulses Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to measure the frequency/duty cycle, high/low ticks, or high/low time of digital pulses. Alternatively, you can configure a task for measuring digital pulses using the DAQ Assistant.

Digital frequency and period are examples of counter measurements. Refer to Counter Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- Counter Programming Flowcharts

Measuring Digital Frequency and Period Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to measure digital frequency or period. Alternatively, you can configure a task for measuring digital frequency using the DAQ Assistant.

Digital frequency and period are examples of counter measurements. Refer to Counter Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- Counter Programming Flowcharts

Generic Programming Flowcharts

This section contains general programming flowcharts that you can use when creating an application. You also can find programming flowcharts for typical applications—such as measuring temperature, measuring current, and measuring strain—in the Common Applications section of this help file.

In the programming flowcharts, many applications also include explicit control functions to start, stop, and clear the task. For instance, for applications that use your counter/timer, such as finite counter input, you need to call the Start function/VI to arm the counter. In LabVIEW, clearing occurs automatically. For other ADEs, you must include these functions in your application.

Functions and VIs produce the core functionality of the NI-DAQmx API. For instance, NI-DAQmx includes functions for timing, triggering, reading, and writing samples.

However, for advanced functionality, Visual C++, Visual C#, Visual Basic .NET, and LabVIEW require properties. ANSI C and LabWindows/CVI employ the Get and Set Attribute functions. For more information, refer to the programming reference help for your ADE.

Analog Input Programming Flowcharts

This section contains general programming flowcharts that you can use when creating an application. You also can find programming flowcharts for typical applications—such as measuring temperature, measuring current, and measuring strain—in the Common Applications section of this help file.

Functions and VIs provide the core functionality of the NI-DAQmx API. For instance, NI-DAQmx includes functions for timing, triggering, reading, and writing samples. However, for advanced functionality, Visual C++, Visual C#, Visual Basic .NET, and LabVIEW require properties. ANSI C and LabWindows/CVI employ the Get and Set Attribute functions. For more information, refer to the programming reference help for your ADE.

Related concepts:
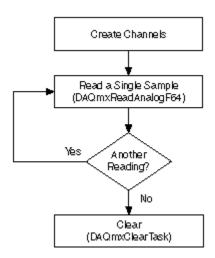
- Single Sample Analog Input Programming Flowchart
- Finite Analog Input Programming Flowchart
- Continuous Analog Input Programming Flowchart
- Triggered Acquisition Programming Flowchart

Single Sample Analog Input Programming Flowchart

Acquiring a single sample is an on-demand operation. In other words, NI-DAQmx acquires one value from an input channel and immediately returns the value. This operation does not require any buffering or hardware timing. For example, if you periodically needed to monitor the fluid level in a tank, you acquire single data points. You can connect the transducer that produces a voltage representing the fluid level to a single channel on your measurement device and initiate a single-channel, single-point acquisition when you want to know the fluid level.

With NI-DAQmx, you also can gather data from multiple channels. For instance, you

might want to monitor the fluid level in the tank as well as the temperature. In this case, you need two transducers connected to two channels on your device. The following flowchart depicts the steps to programmatically create a single sample analog input application. If you prefer, you can configure a task for acquiring a single sample using the DAQ Assistant.
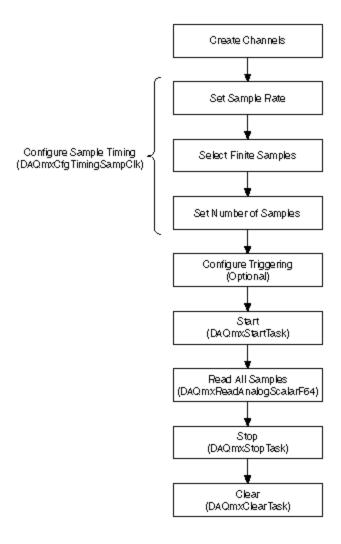
> **Tip** To increase performance, especially when multiple samples are read, include the Start function/VI and Stop function/VI in your application. In the previous flowchart, the Start function/VI would come just before you read samples, and Stop would come just before you clear the task.

Finite Analog Input Programming Flowchart

One way to acquire multiple samples for one or more channels is to acquire single samples in a repetitive manner. However, acquiring a single sample on one or more channels over and over is inefficient and time consuming. Moreover, you do not have accurate control over the time between each sample or channel. Instead, you can use hardware timing, which uses a buffer in computer memory to acquire data more efficiently. Programmatically, you need to include the timing function, specifying the **sample rate** and the **sample mode** (finite). As with other functions, you can acquire multiple samples for a single channel or multiple channels. You can configure a task for finite analog input using the DAQ Assistant.

```
                    ┌─────────────────────┐
                    │   Create Channels   │
                    └─────────────────────┘
                              │
                              ▼
                    ┌─────────────────────┐
                    │   Set Sample Rate   │
                    └─────────────────────┘
                              │
                              ▼
                    ┌─────────────────────┐
Configure Sample Timing │ Select Finite Samples │
(DAQmxCfgTimingSampClk) └─────────────────────┘
                              │
                              ▼
                    ┌─────────────────────┐
                    │ Set Number of Samples │
                    └─────────────────────┘
                              │
                              ▼
                    ┌─────────────────────┐
                    │ Configure Triggering │
                    │     (Optional)      │
                    └─────────────────────┘
                              │
                              ▼
                    ┌─────────────────────┐
                    │        Start        │
                    │   (DAQmxStartTask)  │
                    └─────────────────────┘
                              │
                              ▼
                    ┌─────────────────────────┐
                    │     Read All Samples    │
                    │(DAQmxReadAnalogScalarF64)│
                    └─────────────────────────┘
                              │
                              ▼
                    ┌─────────────────────┐
                    │        Stop         │
                    │   (DAQmxStopTask)   │
                    └─────────────────────┘
                              │
                              ▼
                    ┌─────────────────────┐
                    │        Clear        │
                    │   (DAQmxClearTask)  │
                    └─────────────────────┘
```

Continuous Analog Input Programming Flowchart

If you want to view, process, or log a subset of the samples as they are being acquired, you need to continually acquire samples. For these types of applications, set the **sample mode** to continuous. The following flowchart depicts the main steps required in an NI-DAQmx application for measuring voltage. Instead, you can configure a task for continuous analog input using the DAQ Assistant.

Analog Output Programming Flowcharts

This section contains general programming flowcharts that you can use when creating an application. You also can find programming flowcharts for typical applications—such as generating voltage and generating current—in the Common Applications section of this help file.

In the programming flowcharts, many applications also include explicit control functions to start, stop, and clear the task. In LabVIEW, clearing occurs automatically. For other ADEs, you must include these functions in your application.

Functions and VIs provide the core functionality of the NI-DAQmx API. For instance, NI-DAQmx includes functions for timing, triggering, reading, and writing samples. However, for advanced functionality, Visual C++, Visual C#, Visual Basic .NET, and

LabVIEW require properties. ANSI C and LabWindows/CVI employ the Get and Set Attribute functions. For more information, refer to the programming reference help for your ADE.

Related concepts:

- Generating Voltage
- Measuring and Generating Current

Single Sample Analog Output Programming Flowchart

Generating a single sample is an on-demand operation. In other words, NI-DAQmx generates one value from an input channel and immediately returns the value. This operation does not require any buffering or hardware timing.

With NI-DAQmx, you also can generate samples from multiple channels. If you prefer, you can configure a task for generating a single sample using the DAQ Assistant.
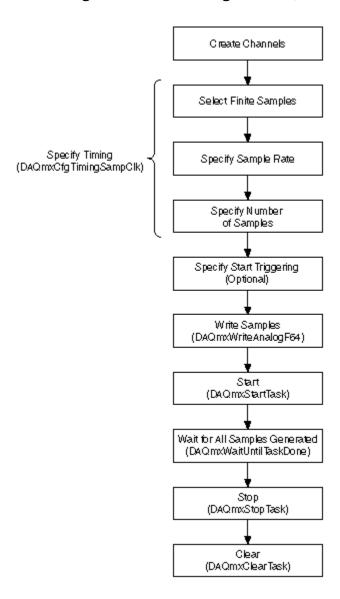
> **Tip** To increase performance, especially when multiple samples are written, include the Start function/VI and Stop function/VI in your application. In the previous flowchart, the Start function/VI would come just before you write samples, and Stop would come just before you clear the task.

Finite Analog Output Programming Flowchart
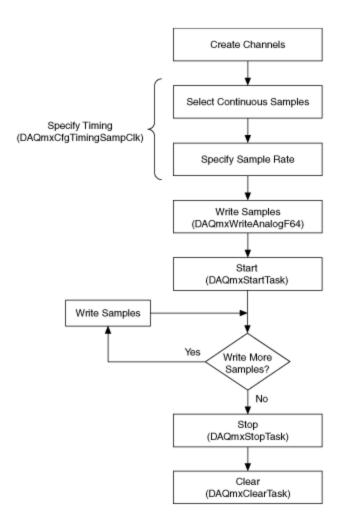
The following flowchart depicts the main steps required in an NI-DAQmx application to

generate a finite number of voltage samples in a buffered generation. If you prefer, you can configure this task using the DAQ Assistant.

Continuous Analog Output Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to continuously generate voltage samples. If you prefer, you can configure this task using the DAQ Assistant.

```
                          ┌─────────────────────┐
                          │   Create Channels   │
                          └──────────┬──────────┘
                                     │
                          ┌──────────▼──────────────────┐
                          │  Select Continuous Samples   │
                          └──────────┬──────────────────┘
  Specify Timing  ⎧                  │
(DAQmxCfgTimingSampClk) ⎨ ┌──────────▼──────────┐
                  ⎩       │  Specify Sample Rate │
                          └──────────┬──────────┘
                                     │
                          ┌──────────▼──────────┐
                          │    Write Samples    │
                          │(DAQmxWriteAnalogF64)│
                          └──────────┬──────────┘
                                     │
                          ┌──────────▼──────────┐
                          │       Start         │
                          │  (DAQmxStartTask)   │
                          └──────────┬──────────┘
    ┌─────────────┐                 │
    │Write Samples│────────────────►│
    └──────▲──────┘                 │
           │              ┌─────────▼─────────┐
           │    Yes       │    Write More     │
           └──────────────┤     Samples?      │
                          └─────────┬─────────┘
                                    │ No
                          ┌─────────▼─────────┐
                          │       Stop        │
                          │  (DAQmxStopTask)  │
                          └─────────┬─────────┘
                                    │
                          ┌─────────▼─────────┐
                          │       Clear       │
                          │ (DAQmxClearTask)  │
                          └───────────────────┘
```

Counter Programming Flowcharts

This section contains general programming flowcharts that you can use when creating an application. You also can find programming flowcharts for typical applications—such as counting edges and generating pulses—in the Common Applications section of this help file.

In the programming flowcharts, many applications also include explicit control functions to start, stop, and clear the task. For instance, for applications that use your counter, such as counting edges or measuring period, you need to call the Start function/VI to arm the counter. In LabVIEW, clearing occurs automatically. For other ADEs, you must include these functions in your application.

Functions and VIs provide the core functionality of the NI-DAQmx API. For instance, NI-DAQmx includes functions for timing, triggering, reading, and writing samples.

However, for advanced functionality, Visual C++, Visual C#, Visual Basic .NET, and LabVIEW require properties. ANSI C and LabWindows/CVI employ the Get and Set Attribute functions. For more information, refer to the programming reference help for your ADE.
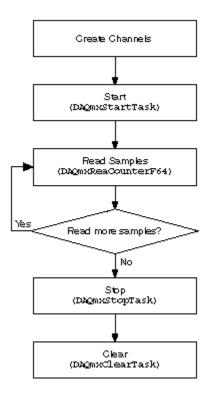
Related concepts:

- Edge Counting
- Generating Pulses
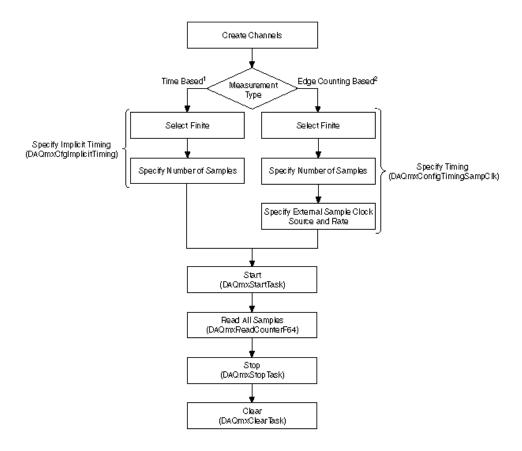
Single Point Counter Input Programming Flowchart

The following flowchart depicts the main steps you must complete for an on-demand counting application. If you prefer, you can configure this task using the DAQ Assistant.

Finite Counter Input Programming Flowchart

The following flowchart depicts the main steps you must complete for finite counter input in an NI-DAQmx application. If you prefer, you can configure this task using the DAQ Assistant.

[1]Time-based measurements include period, semi-period, pulse width, two-edge separation, and digital frequency.
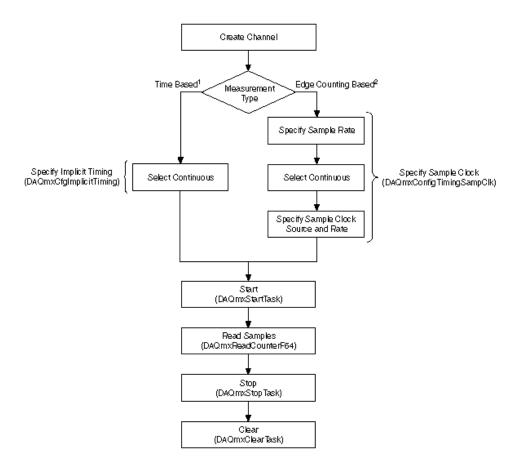
[2]Edge counting-based measurements include edge counting, encoder-based position measurements, and GPS timestamp measurements.

Continuous Counter Input Programming Flowchart

The following flowchart depicts the main steps you must complete for continuous counting in an NI-DAQmx application. If you prefer, you can configure this task using the DAQ Assistant.

[1]Time-based measurements include period, semi-period, pulse width, two-edge separation, and digital frequency.

[2]Edge counting-based measurements include edge counting, encoder-based position measurements, and GPS timestamp measurements.

Digital Input Programming Flowcharts

This section contains general programming flowcharts that you can use when creating an application. You also can find programming flowcharts for typical applications—such as measuring a digital value—in the Common Applications section of this help file.

In the programming flowcharts, many applications also include explicit control functions to start, stop, and clear the task. In LabVIEW, clearing occurs automatically. For other ADEs, you must include these functions in your application.

Functions and VIs provide the core functionality of the NI-DAQmx API. For instance, NI-DAQmx includes functions for timing, triggering, reading, and writing samples. However, for advanced functionality, Visual C++, Visual C#, Visual Basic .NET, and LabVIEW require properties. ANSI C and LabWindows/CVI employ the Get and Set Attribute functions. For more information, refer to the programming reference help for your ADE.

Related concepts:

- Measuring and Generating Digital Values

Single Sample Digital Input Programming Flowchart

Acquiring a single sample is an on-demand operation. In other words, NI-DAQmx acquires one value from an input channel and immediately returns the value. This operation does not require any buffering or hardware timing. For example, if you periodically needed to monitor the fluid level in a tank, you acquire single data points. You can connect the transducer that produces a voltage representing the fluid level to a single channel on your measurement device and initiate a single-channel, single-point acquisition when you want to know the fluid level.

With NI-DAQmx, you also can gather data from multiple channels. For instance, you might want to monitor the fluid level in the tank as well as the temperature. In this case, you need two transducers connected to two channels on your device. The following flowchart depicts the steps to programmatically create an application to measure digital values. If you prefer, you can configure a task for acquiring a single sample using the DAQ Assistant.
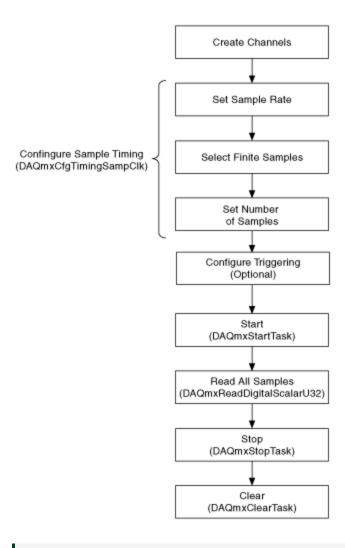
> 💡 **Tip** To increase performance, especially when multiple samples are read, include the Start function/VI and Stop function/VI in your application. In the previous flowchart, the Start function/VI would come just before you read samples, and Stop would come just before you clear the task.

Finite Digital Input Programming Flowchart

One way to acquire multiple samples for one or more channels is to acquire single samples in a repetitive manner. However, acquiring a single data sample on one or more channels over and over is inefficient and time consuming. Moreover, you do not have accurate control over the time between each sample or channel. Instead, you can use hardware timing, which uses a buffer in computer memory to acquire data more efficiently. Programmatically, you need to include the timing function, specifying the **sample rate** and the **sample mode** (finite). As with other functions, you can acquire multiple samples for a single channel or multiple channels. You can configure a task for measuring digital values using the DAQ Assistant.
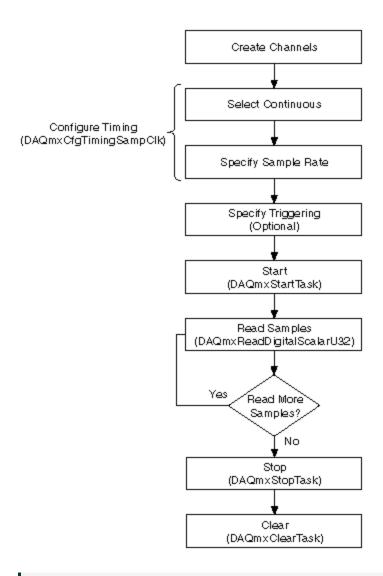
> **Note** Triggering and Sample Clock timing for Digital I/O are not supported on all devices.

Related concepts:

- Single Sample Digital Input Programming Flowchart

Continuous Digital Input Programming Flowchart

If you want to view, process, or log a subset of the samples as they are being acquired, you need to continually acquire samples. For these types of applications, set the **sample mode** to continuous. The following flowchart depicts the main steps required in an NI-DAQmx application for acquiring digital signals. You can configure a task for continuously acquiring digital values using the DAQ Assistant.

```
                    ┌─────────────────────┐
                    │   Create Channels   │
                    └─────────────────────┘
                               │
                               ▼
                    ┌─────────────────────┐
                    │  Select Continuous  │
                    └─────────────────────┘
    Configure Timing           │
  (DAQmxCfgTimingSampClk)      ▼
                    ┌─────────────────────┐
                    │ Specify Sample Rate │
                    └─────────────────────┘
                               │
                               ▼
                    ┌─────────────────────┐
                    │  Specify Triggering │
                    │     (Optional)      │
                    └─────────────────────┘
                               │
                               ▼
                    ┌─────────────────────┐
                    │       Start         │
                    │  (DAQmxStartTask)   │
                    └─────────────────────┘
                               │
                               ▼
                    ┌───────────────────────────────┐
                    │        Read Samples           │
                    │ (DAQmxReadDigitalScalarU32)   │
                    └───────────────────────────────┘
                               │
                               ▼
                          ╱ Read More ╲   Yes
                         ◄   Samples?   ►──────┐
                          ╲           ╱        │
                               │ No            │ (loops back)
                               ▼
                    ┌─────────────────────┐
                    │        Stop         │
                    │  (DAQmxStopTask)    │
                    └─────────────────────┘
                               │
                               ▼
                    ┌─────────────────────┐
                    │       Clear         │
                    │  (DAQmxClearTask)   │
                    └─────────────────────┘
```

Note Sample clock timing for Digital I/O is not supported on all devices.

Digital Output Programming Flowcharts

This section contains general programming flowcharts that you can use when creating an application. You also can find programming flowcharts for typical applications in the Common Applications section of this help file.

In the programming flowcharts, many applications also include explicit control functions to start, stop, and clear the task. For instance, for applications that use your counter/timer, such as counting edges or measuring period, you need to call the Start function/VI to arm the counter. In LabVIEW, clearing occurs automatically. For other

ADEs, you must include these functions in your application.

Functions and VIs provide the core functionality of the NI-DAQmx API. For instance, NI-DAQmx includes functions for timing, triggering, reading, and writing samples. However, for advanced functionality, Visual C++, Visual C#, Visual Basic .NET, and LabVIEW require properties. ANSI C and LabWindows/CVI employ the Get and Set Attribute functions. For more information, refer to the programming reference help for your ADE.

Single Sample Digital Output Programming Flowchart

Generating a single sample is an on-demand operation. In other words, NI-DAQmx generates one value on an output channel immediately after the Write function/VI is called. This operation does not require any buffering or hardware timing.

With NI-DAQmx, you also can generate samples from multiple channels. If you prefer, you can configure a task for generating a single sample using the DAQ Assistant.
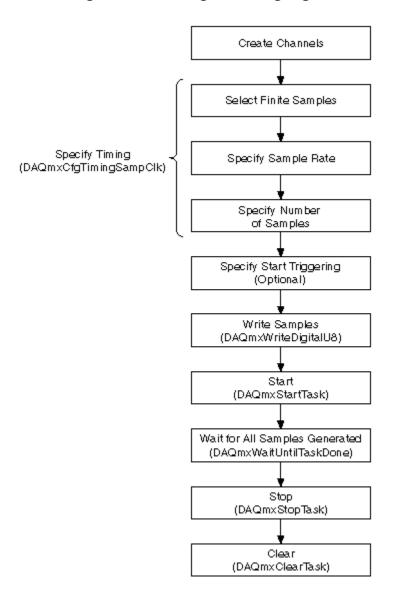
Tip To increase performance, especially when multiple samples are written, include the Start function/VI and Stop function/VI in your application. In the preceding flowchart, the Start function/VI would come just before you write samples, and Stop would come just before you clear the task.

Finite Digital Output Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to generate a finite number of digital values in a buffered generation. If you prefer, you can configure a task for generating digital values using the DAQ Assistant.
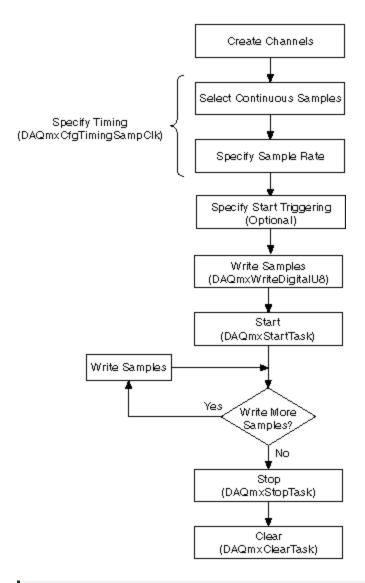
> **Note** Sample clock timing for Digital I/O is not supported on all devices.

Continuous Digital Output Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to continuously generate digital values. If you prefer, you can configure a task for generating digital values using the DAQ Assistant.
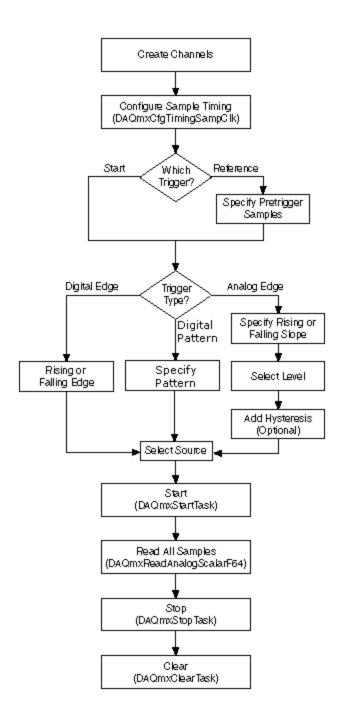
Note Sample clock timing for Digital I/O is not supported on all devices.

Triggered Acquisition Programming Flowchart

The following flowchart depicts the main steps you follow for adding triggering to an acquisition. If you prefer, you can configure triggering with the DAQ Assistant.

Measuring GPS Timestamp

You can take a GPS timestamp measurement with the NI PXI-6608. In a GPS timestamp measurement, the NI PXI-6608 determines the precise time of year using a specialized onboard counter. You can select a single point (on-demand) timestamp or a buffered (sample clock) timestamp.

You can synchronize the GPS timestamp counter to a GPS receiver signal by using a pulse per second (PPS) or an IRIG-B (timecode TTL) synchronization signal from the GPS receiver. PPS does not include any timing information; rather, the PPS accurately reports when the beginning of a second occurs. IRIG-B, on the other hand, has the time encoded in the signal from the beginning of the current year. The GPS counter can latch on the current time upon receiving a hardware gate signal. GPS does not provide year information; however, the time is stored in a 64-bit floating-point number that can be converted to seconds since January 1 of the current year.

When doing an on-demand GPS timestamp measurement, you must first arm the counter by calling the Start function/VI. Each subsequent read returns the number of seconds counted.
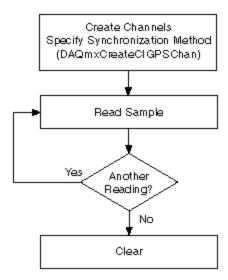
When doing a buffered GPS timestamp measurement, the current time is latched on each active edge of the sample clock and stored in the buffer. There is no built-in clock for buffered GPS timestamp measurements, so you must supply an external sample clock.

Related concepts:

- GPS Timestamp Programming Flowchart
- Finding Examples

GPS Timestamp Programming Flowchart

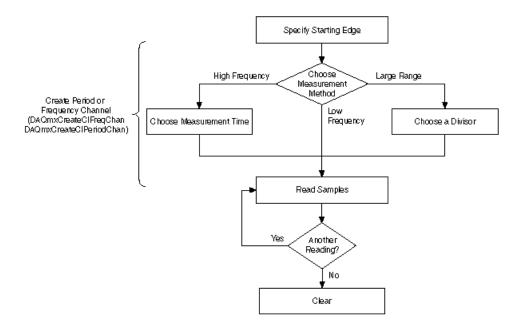The following flowchart depicts the main steps required in an NI-DAQmx application to take a GPS timestamp measurement with an NI PXI-6608.

GPS timestamp is an example of a counter measurement. Refer to Counter Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- Counter Programming Flowcharts

Measuring Linear Displacement

Linear displacement is movement and direction along a single axis. A position or linear displacement sensor is a device whose output signal represents the distance an object has traveled from a reference point. The linear variable differential transformer (LVDT) is a sensor that measures linear displacement.

On M Series devices, C Series devices, and NI-TIO-based devices, you can use the counters to perform displacement measurements with two-pulse encoders. Linear position can be measured with two-pulse encoders. You can choose to do either a single point or a buffered sample clock displacement measurement.

You also can measure velocity with two-pulse encoders, but you need to use a sample clock with a fixed frequency. To measure velocity, use the following formula:

$V = D/T$

where V is the average velocity, D is the distance, and T is time.

The counter measures the position of the encoder using the A and B signals, which are offset by 90°. The counter also supports the Z index, which provides a precise reference point and is available on some encoders.

Related concepts:

- [Measuring Position with an RVDT or LVDT Programming Flowchart](#)
- [Measuring Position with Encoders Programming Flowchart](#)
- [Finding Examples](#)

Measuring Position with Encoders Programming Flowchart

The following flowchart depicts the main steps you must complete for measuring position with an encoder in an NI-DAQmx application. If you prefer, you can configure a task using the DAQ Assistant.

Measuring position with an encoder is an example of a counter measurement. Refer to Counter Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- [Counter Programming Flowcharts](#)

Measuring Position with an RVDT or LVDT Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to measure position with an RVDT or LVDT. Alternatively, you can configure a task for measuring position using the DAQ Assistant.

> 💡 **Tip** To increase performance, especially when multiple samples are read, include the Start function/VI and Stop function/VI in your application. In the previous flowchart, the Start function/VI would come just before you read samples, and Stop would come just before you clear the task.

Measuring position is an example of an analog input measurement. Refer to Analog Input Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- Analog Input Programming Flowcharts

Measuring Period, Semi-Period, Pulse Width, and Two-Edge

Separation

You can measure period, semi-period, pulse width, and two-edge separation using counters, such as on a DAQ device, to determine the duration of an event or to determine the interval time between two events.

Period measurements measure the time between consecutive rising or falling edges of a pulse. Semi-period measurements measure the time between consecutive edges. Pulse width measurements measure the time between either a rising and falling edge, or a falling and rising edge. Two-edge separation measurements measure the time between the rising or falling edge of one digital signal and the rising or falling edge of another digital signal.

The formula for period, semi-period, pulse width, and two-edge separation is as follows:

Period, Semi-Period, Pulse Width, or Two-Edge Separation (in seconds) = Count / Counter Timebase Rate (in Hz).

where Count is the number of counter timebase ticks that elapse during one period, semi-period, pulse width, or two-edge separation of the measured input signal or signals.

The Counter Timebase Rate is a known frequency and is usually a built-in time source. If the counter timebase rate is unknown, you only can make measurements only in terms of ticks of the counter timebase. This may be the case if you are using an external signal for the counter timebase, and the frequency of the external signal is unknown or aperiodic.

Period, semi-period, pulse width, and two-edge separation are examples of time measurements. Refer to Configuring a Time Measurement in NI-DAQmx and Two Counter Measurement Method for more information about measuring time.
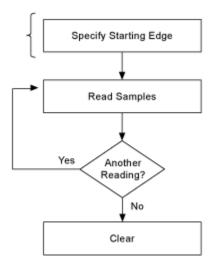
Related concepts:

- Generating Pulses
- Measuring Semi-Period, Two-Edge Separation, and Pulse Width Programming Flowchart
- Measuring Digital Frequency and Period Programming Flowchart
- Measuring Pulses Programming Flowchart
- Finding Examples

Measuring Pulses Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to measure the frequency/duty cycle, high/low ticks, or high/low time of digital pulses. Alternatively, you can configure a task for measuring digital pulses using the DAQ Assistant.

Digital frequency and period are examples of counter measurements. Refer to Counter Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- Counter Programming Flowcharts

Measuring Digital Frequency and Period Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to measure digital frequency or period. Alternatively, you can configure a task for measuring digital frequency using the DAQ Assistant.

Digital frequency and period are examples of counter measurements. Refer to Counter Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- Counter Programming Flowcharts

Measuring Semi-Period, Two-Edge Separation, and Pulse Width Programming Flowchart

The following flowchart demonstrates the main steps required in an NI-DAQmx application to measure semi-period and pulse width. Alternatively, you can configure a task for measuring semi-period and pulse width using the DAQ Assistant.

Period, semi-period, two-edge separation, and pulse width is an example of a counter measurement. Refer to Counter Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- Counter Programming Flowcharts

Measuring Pressure
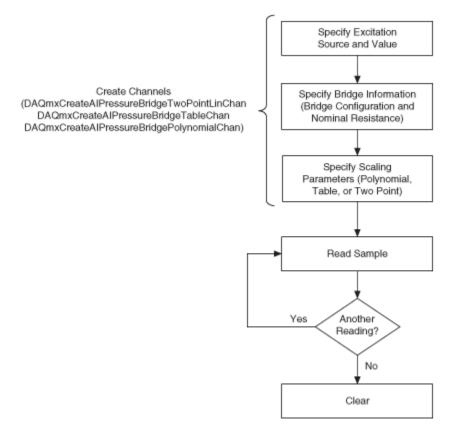
Pressure is a measure of force per unit area. You can use bridge-based sensors to measure pressure.

Related concepts:

- Measuring Pressure Programming Flowchart
- Finding Examples

Measuring Pressure Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to measure pressure. Alternatively, you can configure a task for measuring pressure using the DAQ Assistant.


Tip To increase performance, especially when multiple samples are read, include the Start function/VI and Stop function/VI in your application. In the previous flowchart, the Start function/VI would come just before you read samples, and Stop would come just before you clear the task.

When selecting the scaling type, choose the one that best matches the specifications for your sensor.

Measuring pressure is an example of analog input measurement. Refer to Analog Input Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- Analog Input Programming Flowcharts

Measuring Proximity
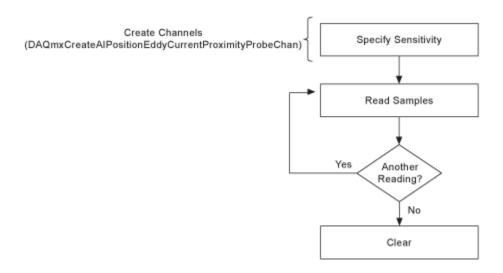
Proximity is the distance between two objects. An eddy current proximity probe is a transducer that uses changes in voltage to measure proximity. Eddy current proximity probes use a high-frequency radio signal to convert voltage to a proximity measurement. Measurements are recorded as milimeters, microns, mVolts/mil and Volts/mil where a mil is 1/1000 of an inch.

Related concepts:

- Measuring Proximity Programming Flowchart
- Finding Examples

Measuring Proximity Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to measure proximity using an eddy current proximity probe. Alternatively, you can configure a task for measuring proximity using the DAQ Assistant.

Measuring proximity is an example of an analog input measurement. Refer to Analog Input Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- Analog Input Programming Flowcharts

Generating Pulses

A pulse is a rapid change in the amplitude of a signal from its idle value to an active value for a short period of time. Pulses can have high or low idle states. A pulse with a low idle state starts at the low value (typically zero), pulses high, and returns to low. A pulse with a high idle state starts high, pulses to low, and returns to high.

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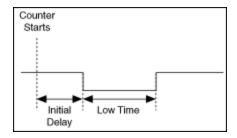
Each pulse or pulse train consists of three parts:

- **High Time**—The amount of time the pulse is at a high level.
- **Low Time**—The amount of time the pulse is at a low level.
- **Initial Delay**—The amount of time the output remains at the idle state before generating the pulse. The idle state always replaces high time or low time for the first pulse of a generation, depending on the idle state.
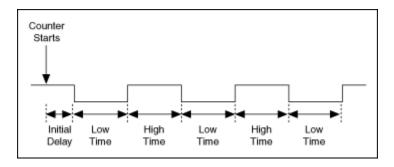
The pairing of high time and low time pair is a pulse specification.

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

The following illustration shows the parts of a pulse.

The following illustration shows the parts of a pulse train.
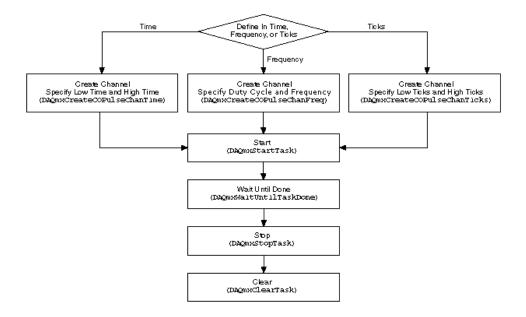
Before you generate a pulse, you need to determine if you want to output the pulse or pulse train in terms of frequency, time, or number of ticks of the counter timebase. For frequency, you need to determine the duty cycle. For time, you specify the high time and the low time. Use the number of ticks if you are using a counter timebase with an unknown rate. When you configure a pulse generation, the output appears at the counter output terminal.

Related concepts:

- Generating a Pulse Programming Flowchart
- Generating a Finite Pulse Train Programming Flowchart
- Generating a Continuous Pulse Train Programming Flowchart
- Finding Examples
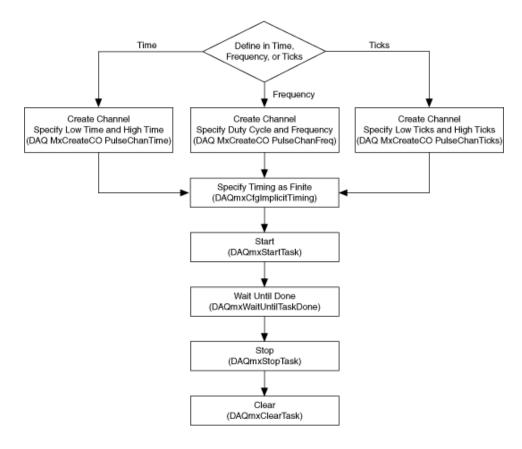
Generating a Pulse Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to generate a pulse. If you prefer, you can configure a task to generate a pulse using the DAQ Assistant.
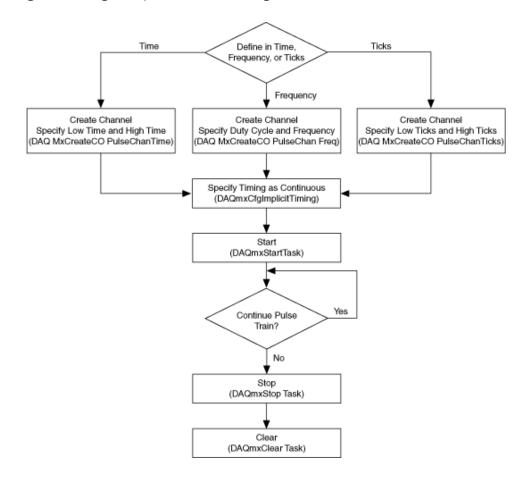
Generating a Finite Pulse Train Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application for generating a finite pulse train. Alternatively, you can configure a task for generating the pulse train using the DAQ Assistant.

Generating a Continuous Pulse Train Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application for generating a continuous pulse train. Alternatively, you can configure a task for generating the pulse train using the DAQ Assistant.

Measuring Resistance

Resistance is the opposition to passage of an electric current. One Ohm (Ω) is the resistance through which one volt (V) of electric force causes one ampere (A) to flow. Two common methods for measuring resistance are the 2-wire method and the 4-wire method. Both methods send a current through a resistor with a measurement device measuring the voltage drop from the signal before and after it crosses the resistor. The 2-wire method is easier to implement, but this method is less accurate than the 4-wire method for resistances below 100 Ω. For 3-wire resistors, there is also a 3-wire method. To calculate resistance, use the following equation.

$$R_{(\Omega)} = V_{(V)} / I_{(A)}$$
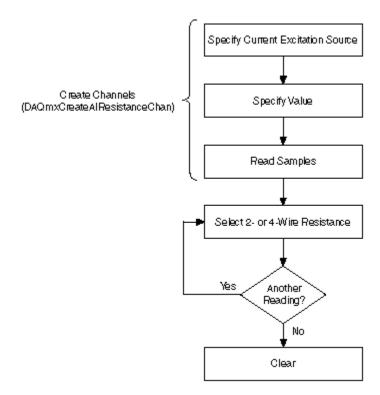
where R is the resistance, V is the voltage, and I is the current.

Related concepts:

- Measuring Resistance Programming Flowchart
- Finding Examples

Measuring Resistance Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to measure resistance. Alternatively, you can configure a task for measuring resistance using the DAQ Assistant.


Tip To increase performance, especially when multiple samples are read, include the Start function/VI and Stop function/VI in your application. In the preceding flowchart, the Start function/VI would come just before you read samples, and Stop would come just before you clear the task.

Measuring resistance is an example of an analog input measurement. Refer to Analog Input Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- Analog Input Programming Flowcharts

Measuring Sound Pressure

Sound pressure is the dynamic variation of the static pressure of air and is measured in force per unit area (Pa). The instantaneous sound pressure is typically averaged over a certain duration to give sound pressure level. Sound pressure level is usually represented on a logarithmic amplitude scale, which is similar to the human perception of hearing. Typical values on this logarithmic scale are a sound level of 0 dB, which is the average threshold of human hearing, 60 to 70 dB for normal conversation, 110 dB at an extremely loud concert, and 150 dB for the noise of a rocket takeoff or a jet engine at close range.

The Sound Pressure Level (SPL or LP) in decibels is defined by the following equation.

$$SPL = 20 \log 10 \ (p/pref)$$

where p is the instantaneous sound pressure in Pa and pref is 20 μPa, the internationally accepted reference for sound pressure measurements, which roughly corresponds to the threshold of human hearing.
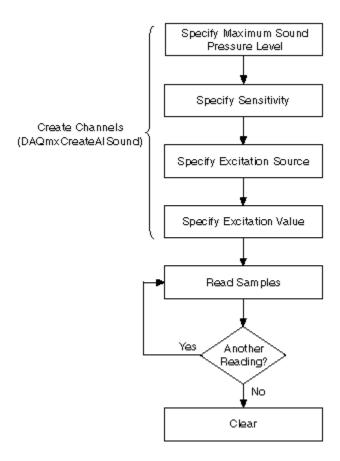
You use a microphone to measure sound pressure. The microphone acts as a transducer, creating a voltage signal that is proportional to the instantaneous sound pressure.

Related concepts:

- Measuring Sound Pressure Programming Flowchart
- Finding Examples

Measuring Sound Pressure Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to measure sound pressure. Alternatively, you can configure a task for measuring sound pressure using the DAQ Assistant.

> 🔅 **Tip** To increase performance, especially when multiple samples are read, include the Start function/VI and Stop function/VI in your application. In the previous flowchart, the Start function/VI would come just before you read samples, and Stop would come just before you clear the task.
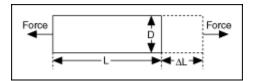
Measuring sound pressure is an example of an analog input measurement. Refer to Analog Input Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- Analog Input Programming Flowcharts

Measuring Strain

Strain (ε) is the amount of deformation of a body due to an applied force. Specifically, strain is the fractional change in length, as shown in the following figure.

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 (με).

When a uniaxial force strains a bar, as in the preceding figure, a phenomenon known as Poisson Strain causes the girth of the bar, D, to contract in the transverse direction, which is perpendicular to the force. The magnitude of this transverse contraction is a material property indicated by its Poisson's Ratio. The Poisson's Ratio of a material is the negative ratio of the strain in the transverse direction to the strain in the axial direction, which is parallel to the force. Poisson's Ratio for steel, for example, ranges from 0.25 to 0.3.
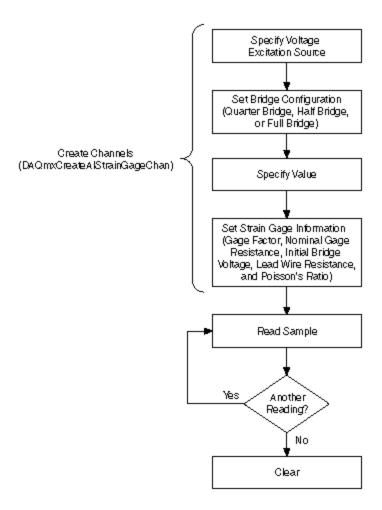
To measure strain, you can use one or more strain gages in a Wheatstone bridge in one of several bridge configurations. Refer to Signal Conditioning Requirements for Bridge-Based Sensors for more information about strain gages and bridge configurations.

Related concepts:

- Measuring Strain Programming Flowchart
- Finding Examples

Measuring Strain Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to measure strain. Alternatively, you can configure a task to measure strain with a strain gage using the DAQ Assistant.

> 💡 **Tip** To increase performance, especially when multiple samples are read, include the Start function/VI and Stop function/VI in your application. In the previous flowchart, the Start function/VI would come just before you read samples, and Stop would come just before you clear the task.

Measuring strain is an example of an analog input measurement. Refer to Analog Input Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- Analog Input Programming Flowcharts

Measuring Torque
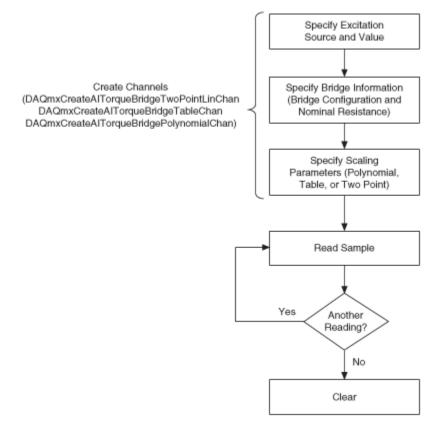
Torque is a measure of the tendency of a force to rotate an object. Torque is the cross product of a force and the distance of the force from the fulcrum. You can use bridge-based sensors to measure torque.

Related concepts:

- Measuring Torque Programming Flowchart
- Finding Examples

Measuring Torque Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to measure torque. Alternatively, you can configure a task for measuring torque using the DAQ Assistant.

> :bulb: **Tip** To increase performance, especially when multiple samples are read, include the Start function/VI and Stop function/VI in your application. In the previous flowchart, the Start function/VI would come just before you read

> samples, and Stop would come just before you clear the task.

When selecting the scaling type, choose the one that best matches the specifications for your sensor.

Measuring torque is an example of analog input measurement. Refer to Analog Input Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- Analog Input Programming Flowcharts
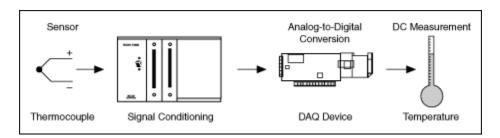
Measuring Temperature

> ✏️ **Note** Temperature measurements may require you to condition the signal. The conditioning requirements depend on your sensor. Refer to the Overview of Temperature Sensor Types for an explanation of sensor types and conditioning requirements.
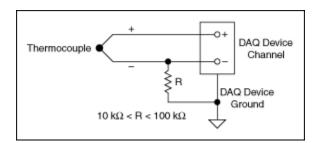
Using a Thermocouple to Measure Temperature

A popular way to measure temperature with a DAQ device is to use a thermocouple, as shown in the following figure, because thermocouples are inexpensive, easy to use, and easy to obtain. Thermocouples produce a voltage that varies based on temperature. Using a thermocouple, you can measure a voltage and use a formula to convert the voltage measurement to temperature.

The typical wiring for a thermocouple, as shown in the following figure, uses a resistor,

R, only if the thermocouple is not grounded at any other point. If, for example, the thermocouple tip were already grounded, using a resistor would cause a ground loop and result in erroneous readings.
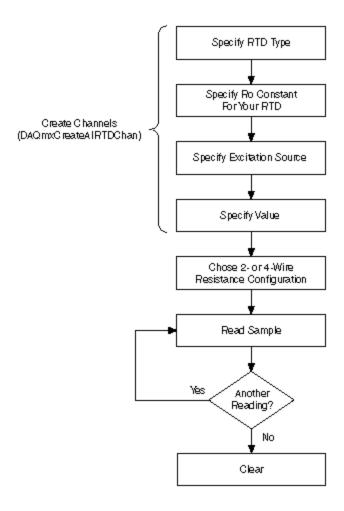
You also can measure temperature using Resistance Temperature Detectors (RTD) and Thermistors.

Related concepts:

- Measuring Temperature with an RTD Programming Flowchart
- Measuring Temperature with a Thermistor Programming Flowchart
- Measuring Temperature with a Thermocouple Programming Flowchart
- Finding Examples

Measuring Temperature with an RTD Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to measure temperature with an RTD. Alternatively, you can configure a task for measuring temperature using the DAQ Assistant.

> 💡 **Tip** To increase performance, especially when multiple samples are read, include the Start function/VI and Stop function/VI in your application. In the previous flowchart, the Start function/VI would come just before you read samples, and Stop would come just before you clear the task.

Measuring temperature is an example of analog input measurement. Refer to Analog Input Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- Analog Input Programming Flowcharts

Measuring Temperature with a Thermistor Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to measure temperature with a thermistor. Alternatively, you can configure a task for measuring temperature using the DAQ Assistant.

Tip To increase performance, especially when multiple samples are read, include the Start function/VI and Stop function/VI in your application. In the previous flowchart, the Start function/VI would come just before you read samples, and Stop would come just before you clear the task.

Measuring temperature is an example of analog input measurement. Refer to Analog Input Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- [Analog Input Programming Flowcharts](#)

Measuring Temperature with a Thermocouple Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to measure temperature with a thermocouple. Alternatively, you can configure a task for measuring temperature using the DAQ Assistant.

💡 **Tip** To increase performance, especially when multiple samples are read, include the Start function/VI and Stop function/VI in your application. In the previous flowchart, the Start function/VI would come just before you read samples, and Stop would come just before you clear the task.

Measuring temperature is an example of analog input measurement. Refer to Analog Input Programming Flowcharts for additional flowcharts that can help you create an application.
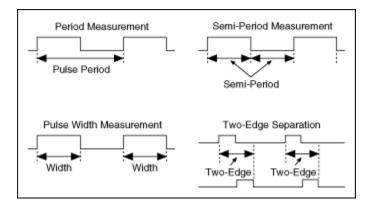
Related concepts:

- Analog Input Programming Flowcharts

Measuring Velocity

Velocity is the rate of change in position with respect to time.

Measuring Velocity with an IEPE Velocity Transducer
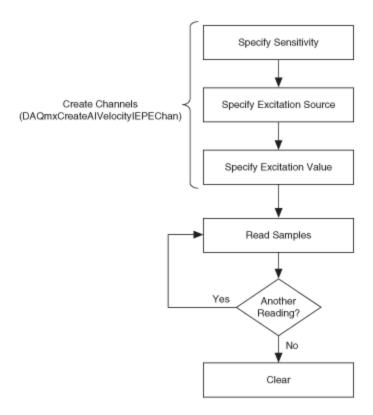
A velocity transducer is an IEPE sensor that converts velocity to voltage. Velocity transducers are typically used to measure vibration.

Related concepts:

- Measuring Velocity Programming Flowchart
- Finding Examples

Measuring Velocity Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to measure velocity with a piezoelectric sensor. Alternatively, you can configure a task for measuring velocity using the DAQ Assistant.

Tip To increase performance, especially when multiple samples are read, include the Start function/VI and Stop function/VI in your application. In the previous flowchart, the Start function/VI would come just before you read samples, and Stop would come just before you clear the task.

Measuring velocity is an example of analog input measurement. Refer to Analog Input Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- Analog Input Programming Flowcharts

Measuring Angular Velocity (Encoder)

On devices that support it, you can use the counters to perform velocity measurements with encoders. An encoder is a device that converts linear or rotary displacement into digital or pulse signals. Angular velocity can be measured with X1, X2, and X4 quadrature encoders. You can choose to do either single-point (on-demand)

velocity measurement or buffered (sample clock) velocity measurement.

The counter measures the velocity of the encoder using the A and B signals, which are offset by 90°. The velocity is calculated using the A and B input signal transitions and the amount of time that elapses between the changes in the encoder count value.

The decoding type attribute/property specifies how to count and interpret the pulses the encoder generates on signal A and signal B.

Measuring Linear Velocity (Encoder)

On devices that support it, you can use the counters to perform velocity measurements with encoders. An encoder is a device that converts linear or rotary displacement into digital or pulse signals. Linear velocity can be measured with X1, X2, and X4 quadrature encoders. You can choose to do either single-point (on-demand) velocity measurement or buffered (sample clock) velocity measurement.

The counter measures the velocity of the encoder using the A and B signals, which are offset by 90°. The velocity is calculated using the A and B input signal transitions and the amount of time that elapses between the changes in the encoder count value.

The pulse in the distance per pulse attribute/property is one full period of both signal A and signal B in the units specified by the length portion of the Units input. The decoding type attribute/property specifies how to count and interpret the pulses the encoder generates on signal A and signal B.

Generating Voltage

You can generate single sample DC signals or time-varying multiple sample signals.

Single Samples—Including Steady Signals

Use single samples if the signal level is more important than the generation rate. For instance, generate one sample at a time if you need to generate a constant, or DC, signal. You can use software or hardware timing if the device supports hardware timing to control when the device generates a signal.

Time-Varying Multiple Samples

Use multiple samples if the generation rate is just as important as the signal level, as in an AC sine wave. Function generators are a common type of device that you can program to produce certain types of waveforms, such as sine, triangle, and square waves. You also can use a DAQ device as a function generator. You do this by generating one cycle of a sine wave, such as with the Sine Generation VI in LabVIEW, storing one cycle of sine wave data in a waveform, and programming the device to generate the values continuously from the waveform one point at a time at a specified rate.

Also called buffered analog output, generating multiple samples involves the following steps:
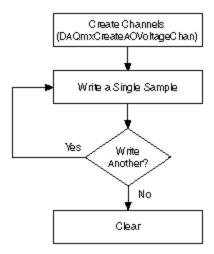
1. Your application writes multiple samples into a buffer.
2. All the samples in the buffer are then sent to your device according to the timing you specify. You can use software or hardware timing (if your device supports hardware timing) to control when your device generates a signal.

Related concepts:

- Generating Voltage Programming Flowchart
- Finite Analog Output Programming Flowchart
- Finding Examples

Generating Voltage Programming Flowchart

The following flowchart illustrates the main steps required in an NI-DAQmx application to generate voltage. Alternatively, you can configure a task for generating voltage using the DAQ Assistant.

> 💡 **Tip** To increase performance, especially when multiple samples are written, include the Start function/VI and Stop function/VI in your application. In the previous flowchart, the Start function/VI would come just after you write samples, and Stop would come just before you clear the task.
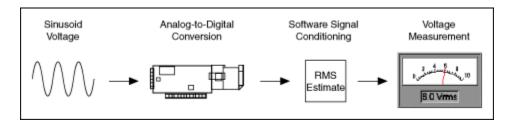
Generating voltage is an example of an analog output measurement. Refer to Analog Output Programming Flowcharts for additional flowcharts that can help you create an application.

Related concepts:

- [Analog Output Programming Flowcharts](#)

Measuring Voltage

Most measurement devices can measure, or read, voltage. Two common voltage measurements are direct current (DC) and alternating current (AC).

Measuring DC Voltage

DC voltage is useful for measuring phenomena that change slowly with time, such as temperature, pressure, or strain. With DC signals, you want to accurately measure the amplitude of a signal at a given point in time.
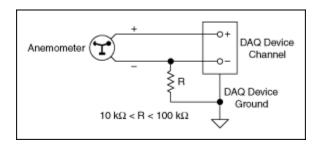
Wind Speed Example

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

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

Using this equation, a measurement of 3 V would correspond to a wind speed of 60 mph (3 V × 20 mph/V = 60 mph).
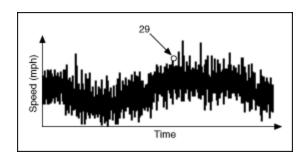
Notice that the wiring diagram in the following figure uses a resistor, R, because an anemometer is usually not a grounded signal source. If the anemometer transducer were already grounded, using a resistor would cause a ground loop and result in erroneous readings.

Averaging

Averaging can improve measurement accuracy for noisy and rapidly changing signals.

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

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

Measuring AC Voltage

AC voltage is a waveform that constantly increases, decreases, and reverses polarity. AC voltage is common in household, lab, and industrial devices because most power lines deliver AC voltage. You can measure AC voltages to determine the maximum, minimum, and peak-to-peak values of a signal. The peak-to-peak value of a signal is the maximum voltage swing, from maximum to minimum.

AC Voltage and Root Mean Square (RMS)
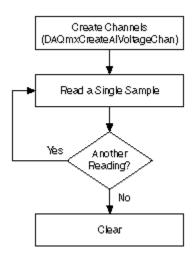
Voltage, current, and power are not constant values because AC signals alternate. However, you can use V_{rms} (root mean square) to measure voltage, current, and power such that a load connected to a 120 volts AC (VAC) source develops the same amount of power as that same load connected to a 120 volts DC (VDC) source. With RMS, the power formula for DC also works for AC. For sinusoidal waveforms, V_{rms} = Vpeak/ square root of 2. Because voltmeters read V_{rms}, the 120 VAC of a typical U.S. wall outlet actually has a peak value of about 170 V.

Related concepts:

- [Measuring Voltage Programming Flowchart](#)
- [Finding Examples](#)

## Measuring Voltage Programming Flowchart

The following flowchart depicts the main steps required in an NI-DAQmx application to measure voltage. Alternatively, you can configure a task for measuring voltage using the DAQ Assistant.

> 🔆 **Tip** To increase performance, especially when multiple samples are read, include the Start function/VI and Stop function/VI in your application. In the previous flowchart, the Start function/VI would come just before you read samples, and Stop would come just before you clear the task.

Measuring voltage is an example of an analog input measurement. Refer to Analog Input Programming Flowcharts for additional flowcharts that can help you create an application.

To measure AC voltages, you generally use a hardware timed acquisition, such as the ones shown in Finite Voltage Measurements and Continuous Voltage Measurements. To measure DC voltages, you often do not need a buffer or hardware timing, so you can use a simple application such as the one shown in Acquiring a Single Sample.

Related concepts:

- [Analog Input Programming Flowcharts](#)
- [Finite Analog Input Programming Flowchart](#)
- [Continuous Analog Input Programming Flowchart](#)
- [Single Sample Analog Input Programming Flowchart](#)

NI-DAQmx Key Concepts

NI-DAQmx Key Concepts covers important concepts in NI-DAQmx such as channels and tasks. The ways that NI-DAQmx handles timing, triggering, buffering, and signal routing are also central in the NI-DAQmx API.

Channels and Tasks in NI-DAQmx

Virtual channels and tasks are fundamental components of NI-DAQmx.

Virtual channels, or sometimes referred to generically as channels, are software entities that encapsulate the physical channel along with other channel specific information—range, terminal configuration, and custom scaling—that formats the data. Tasks are collections of one or more virtual channels with timing, triggering, and other properties.

Related concepts:

- Channels: Physical, Virtual, Local Virtual, and Global Virtual
- Tasks in NI-DAQmx

Channels: Physical, Virtual, Local Virtual, and Global Virtual

A physical channel is a terminal or pin at which you can measure or generate an analog or digital signal. A single physical channel can include more than one terminal, as in the case of a differential analog input channel or a digital port of eight lines. Every physical channel on a device has a unique name (for instance, `SC1Mod4/ai0,` `Dev2/ao5, and Dev6/ctr3`) that follows the NI-DAQmx physical channel naming convention.

Virtual channels are software entities that encapsulate the physical channel along with other channel specific information—range, terminal configuration, and custom scaling—that formats the data. To create virtual channels, use the DAQmx Create

Virtual Channel function/VI or the DAQ Assistant.

Virtual channels created with the DAQmx Create Virtual Channel function/VI are called local virtual channels and can only be used within the task. With this function/VI, you choose the name to assign for the virtual channel, which is used in the rest of the NI-DAQmx software framework to refer to the physical channel.

If you create virtual channels with the DAQ Assistant, you can use them in other tasks and reference them outside the context of a task. Because these channels can apply to multiple tasks, they are called global virtual channels. You can select global virtual channels with the NI-DAQmx API or DAQ Assistant and add them to a task. If you add a global virtual channel to several tasks and modify that global virtual channel with the DAQ Assistant, the change applies to all tasks that use that global virtual channel. You must save the changes before they become globally available.

Related concepts:

- Physical Channel Syntax
- Tasks in NI-DAQmx
- Creating Channels and Tasks with the DAQ Assistant
- Physical Channels
- Internal Channels

Related tasks:

- Creating Virtual Channels with the API

Creating Virtual Channels with the API

The following example illustrates the difference between physical and virtual channels and demonstrates how to create virtual channels with the API.

Problem

Create an NI-DAQmx virtual channel to measure temperature in the range 50° C to 200° C using a J-type thermocouple wired to channel 0 on an M Series device configured as Device 1. Use LabVIEW or LabWindows™/CVI™ to write your application.

Solution

1. Call the AI Temp TC instance of the DAQmx Create Virtual Channel VI in LabVIEW (DAQmxCreateAIThrmcplChan function in LabWindows/CVI).
2. Use `Dev1/ai0` as the physical channel on the device to which the thermocouple signal is connected.
3. Specify `myThermocoupleChannel` as the name to assign to your virtual channel.
4. Select the appropriate values for the thermocouple type and range inputs. NI-DAQmx applies these attributes to the virtual channel.

You have now created a virtual channel.
Related concepts:

- Choosing Whether to Use the API or the DAQ Assistant
- Physical Channels
- Internal Channels

Types of Virtual Channels

You can create a number of different types of virtual channels, depending on the signal type—analog, digital, or counter—and direction (input or output). These channels can be either global virtual channels or local virtual channels. For information on specific functions/VIs, refer to the NI reference help for your ADE.

- **Analog Input Channels**—Analog input channels measure different physical phenomena using a variety of sensors. The type of channel to create depends on the type of sensor and/or phenomenon you want to read. For instance, you can create channels for measuring temperature with a thermocouple, measuring current, measuring voltage, and measuring voltage with excitation.
- **Analog Output Channels**—NI-DAQmx supports two types of phenomena, voltage and current. You can use custom scales if the output from the device relates to another unit of measure.
- **Counter Input/Output Channels**—NI-DAQmx supports several types of counter input and output channels for different types of counter measurements and generations. To find out more about counter measurements and terminals used for common applications, refer to Counter Parts in NI-DAQmx.
- **Digital Input/Output Channels**—For digital channels, you can create both line-

based and port-based digital channels. A line-based channel can contain one or
more digital lines from one or more ports on a device. Reading or writing to a line-
based channel does not affect other lines on the hardware. You can split lines in a
particular port among multiple channels and use those channels simultaneously
within one or multiple tasks, but the lines in a given channel must all be input lines
or all be output lines. Additionally, all channels in a task must be either input
channels or output channels. Some devices also require that the lines of a given
port all be input lines or output lines. Check your device documentation for the
capabilities of your device.

A port-based channel represents a fixed collection of lines on the device. Reading
or writing to a port affects all the lines on the port. The number of lines in the port
(commonly referred to as port width) is hardware dependent and typically varies
from 8 lines (MIO device) to 32 lines (SCXI digital modules).

- **Power Channels**—Power channels source voltage and current and also provide
  measurements of those sources. You can change the voltage and current setpoints
  dynamically at runtime. You can use a power channel to control (enable or disable)
  the power output as well as to detect various error states that impact power
  output.

Related concepts:

- Channels: Physical, Virtual, Local Virtual, and Global Virtual
- Digital Lines, Ports, and Port Width
- Tasks in NI-DAQmx
- Counter Parts in NI-DAQmx

Physical Channel Syntax

Use this syntax to refer to physical channels and groups of physical channels in NI-
DAQmx.

Related concepts:

- Physical Channels
- Internal Channels
- Multidevice Tasks

Physical Channel Names

Physical channel names consist of a device identifier and a slash (/) followed by a channel identifier. For example, if the physical channel is `Dev1/ai1`, the device identifier is `Dev1`, and the channel identifier is `ai1`. MAX assigns device identifiers to devices in the order they are installed in the system, such as `Dev0` and `Dev1`. You also can assign arbitrary device identifiers with MAX.

For analog I/O and counter I/O, channel identifiers combine the type of the channel, such as analog input (ai), analog output (ao), and counter (ctr), with a channel number such as the following:

`ai1`

`ctr0`

For digital I/O, channel identifiers specify a port, which includes all lines within a port:

`port0`

Or, the channel identifier can specify a line within a port:

`port0/line1`

All lines have a unique identifier. Therefore, you can use lines without specifying which port they belong to. For example, `line31` is equivalent to `port3/line7` on a device with four 8-bit ports.

Physical Channel Ranges

To specify a range of physical channels, use a colon between two channel numbers or two physical channel names:

`Dev1/ai0:4`

`Dev1/ai0:Dev1/ai4`

For digital I/O, you can specify a range of ports with a colon between two port numbers:

```
Dev1/port0:1
```

You also can specify a range of lines:

```
Dev1/port0/line0:4
```

```
Dev1/line0:31
```

You can specify channel ranges in reverse order:

```
Dev1/ai4:0
```

```
Dev1/ai4:Dev1/ai0
```

```
Dev1/port1/line3:0
```

Physical Channel Lists
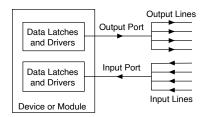
Use commas to separate physical channel names and ranges in a list as follows:

```
Dev1/ai0, Dev1/ai3:6
```

```
Dev1/port0, Dev1/port1/line0:2
```

Digital Lines, Ports, and Port Width

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

- **Line**—A line is an individual signal. It refers to a physical terminal. The data that the line carries are called bits, binary values that are either 1 or 0. 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 port is a collection of digital lines. Usually, the lines are grouped into an 8-bit or 32-bit port.
- **Port Width**—The port width refers to the number of lines in a port. For example, a device with one port with eight lines has a port width of eight.

Channel Name Generation

NI-DAQmx assigns names to local virtual channels that you create programmatically with the NI-DAQmx API when you do not provide a name for each local virtual channel.

Physical Channel Names	Name To Assign	Generated Local Virtual Channel Names
Dev1/ai0:1	—	Dev1/ai0, Dev1/ai1
Dev1/ai0:7	"foo"	foo0, foo1, ..., foo7
Dev1/ai0:7	"foo31"	foo31, foo32, ..., foo38
Dev1/ai0:7	"foo 123"	foo123, foo124, ..., foo130
Dev1/ai0:7	"a0:3, b"	a0, a1, a2, a3, b0, b1, b2, b3

Naming Channels, Tasks, and Scales

Use the following guidelines to name your channels, tasks, and scales:

- Use any alphanumeric characters.
- Do not use nonalphanumeric characters with the following exceptions:
  - In NI-DAQmx 7.4 or later, dashes are allowed in channel, task, and scale names.
  - Spaces are allowed.
  - You can use underscores within the channel, task, or scale name, but you cannot use leading underscores, such as `_Dev1`.

> **Note** You can use other nonalphanumeric characters when creating channels, tasks, and scales, but exporting that configuration to another system might not work correctly, especially if the operating system is in a different language.

- You must use no more than 256 characters.

Cold-Junction Compensation Channels

On devices with built-in cold-junction compensation (CJC) channels, the CJC channel is read once per sample clock edge.

Related concepts:

- [Sample Clock](#)

Tasks in NI-DAQmx

A task is a collection of one or more virtual channels with timing, triggering, and other properties. Conceptually, a task represents a measurement or generation you want to perform. All channels in a task must be of the same I/O type, such as analog input or counter output. However, a task can include channels of different measurement types, such as an analog input temperature channel and an analog input voltage channel. For most devices, only one task per subsystem can run at once, but some devices can run multiple tasks simultaneously. With some devices, you can include channels from multiple devices in a task. To perform a measurement or a generation with a task, follow these steps:

1. Create or load a task. You can create tasks interactively with the DAQ Assistant or programmatically in your ADE such as LabVIEW or LabWindows/CVI.
2. Configure the channel, timing, and triggering properties as necessary.
3. Optionally, perform various task state transitions to prepare the task to perform the specified operation.
4. Read or write samples.
5. Clear the task.

If appropriate for your application, repeat steps 2 through 4. For instance, after reading or writing samples, you can reconfigure the virtual channel, timing, or triggering properties and then read or write additional samples based on this new configuration.

If properties need to be set to values other than their defaults for your task to be successful, your program must set these properties every time it executes. For example, if you run a program that sets property A to a nondefault value and follow that with a second program that does not set property A, the second program uses the

default value of property A. The only way to avoid setting properties programmatically each time a program runs is to use virtual channels and/or tasks created in the DAQ Assistant.

Related concepts:

- Simultaneous Tasks
- Multidevice Tasks
- Task State Model
- Creating Channels and Tasks with the DAQ Assistant

Related tasks:

- Creating Tasks with the API

Creating Tasks with the API

The following example illustrates how to create a task with the API:

Problem

Create an NI-DAQmx task to measure temperature in the range 50°C to 200°C using a J-type thermocouple that is wired to channel 0 on an M Series device configured as Device 1. Sample the temperature 10 times per second, and acquire 10,000 samples. Use LabVIEW or LabWindows/CVI to write your application.

Solution

1. Call the AI Temp TC instance of the DAQmx Create Virtual Channel VI in LabVIEW (DAQmxCreateAIThrmcplChan function in LabWindows/CVI).
2. Specify `Dev1/ai0` as the physical channel for the device connected to the thermocouple signal.
3. Specify `myThermocoupleChannel` as the name to assign to your virtual channel.
4. Select the appropriate values for the thermocouple type and range inputs. NI-DAQmx applies these attributes to the virtual channel.
5. Call the Sample Clock instance of DAQmx Timing VI in LabVIEW (or DAQmxCfgSampClkTiming function in LabWindows/CVI), specifying a `rate` of 10

Hz and a `sample mode` of finite.

6. Call the DAQmx Start Task VI (DAQmxStartTask in LabWindows/CVI).

7. Call the Analog 1D DBL 1Chan NSamp instance of DAQmx Read VI (DAQmxReadAnalogF64 in LabWindows/CVI), specifying number of `samples per channel` as 10,000.

8. Call the DAQmx Stop Task VI (DAQmxStopTask function in LabWindows/CVI) after the desired number of samples have been acquired.

9. Call the DAQmx Clear Task VI (DAQmxClearTask function in LabWindows/CVI).

You have now created a task called myTemperatureTask that uses a local virtual channel called myThermocoupleChannel.

> **Note** You also can use the DAQ Assistant to create the same task and generate the code to run the task.

Related concepts:

- Creating Channels and Tasks with the DAQ Assistant
- Choosing Whether to Use the API or the DAQ Assistant

Using the Start Task function/VI

To explicitly start a task, call the Start Task function/VI. You auto-start a task when you perform some other operation that implicitly starts the task. For instance, calling a Read function/VI or a Write function/VI might implicitly start the task if one is not already started. How to specify this behavior depends on the operation that your task performs. By default, the Read function/VI and the Write function/VI for a single sample automatically starts a task.

Related concepts:

- Committed State
- Running State
- Task State Model
- Verified State

Starting a Finite Measurement Task

If you have specified a task to perform a finite measurement, you do not need to call the Start Task function/VI, nor do you need to change the default behavior of the DAQmx Read function/VI. Calling the Read function/VI starts your task, performs the finite measurement, and stops the task after the last sample is read. The task returns to its state before you called the read operation. However, if you need to perform additional read operations after the task has been stopped (in other words, if you want to read earlier locations in the buffer), the default behavior is insufficient for two reasons:

1. The task is returned to the Verified state and the samples are no longer accessible.
2. Future calls of the Read function/VI start new read operations rather than reading from the completed operation.

For this situation, explicitly commit the task by calling the Control Task function/VI with the `Action` parameter set to Commit. Then, after performing the initial read operation and before performing the subsequent read operations, set the Auto-Start Read attribute/property to False.

Starting a Continuous Measurement Task

For a continuous measurement, explicitly call the Start Task function/VI, perform the desired read operations, and call the Stop Task function/VI to stop the continuous measurement. When you perform a read operation in a loop—regardless if the read operation performs a single-sample, on-demand read, or a multiple-sample, hardware-timed read—call the Start Task function/VI before entering the loop and call the Stop Task function/VI after leaving the loop.

Starting an Analog Output Task

The behavior of the Write function/VI is more complicated. Calling the Write function/VI always results in the task transitioning to at least the Committed state. Whether the task transitions to the Running state depends on the value of the `Auto-Start` parameter.

For single-sample write operation, call a single-sample version of the Write function/VI. This call implicitly starts the task, writes the single sample, and stops the task. For a

multiple-sample, on-demand write operation, call the Write function/VI, but also set the `Auto-Start` parameter to True, which by default is set to False. This call implicitly starts the task, writes the multiple samples, and stops the task.

For a multiple-sample, hardware-timed write operation, first call the Write function/VI to write the samples to generate, explicitly call the Start Task function/VI, wait for the samples to be generated by calling the Wait Until Done function/VI, and then explicitly call the Stop Task function/VI.

If you attempt to perform a hardware-timed generation with the `Auto-Start` parameter of the Write function/VI set to True either because you explicitly set it to true or because you are using a single-sample Write function/VI, the operation might fail because the samples that you write are not transferred to the device in time to generate the waveform. As a result, when performing hardware-timed generations, always write at least part of the waveform to generate before starting the task.

Improving Performance with the Start Task function/VI

There are other situations in which you should explicitly call the DAQmx Start Task function/VI and the DAQmx Stop Task function/VI, even though you are not required to do so. When you call the Read function/VI or the Write function/VI in a loop, you can significantly improve performance if you explicitly call the Start Task function/VI before entering the loop and call the Stop Task function/VI after exiting the loop. Without explicitly calling the Start Task function/VI before entering the loop, the task must implicitly transition from its current state to the Running state before performing the read or write operation. After the read or write operation is complete, the task must implicitly transition from the Running state back to its previous state. These implicit state transitions occur for every iteration of the loop, which is inefficient.

Aborting a Task

Several conditions cause a task to abort:

- To explicitly abort a task, call the DAQmx Control Task function/VI with the `Action` parameter set to Abort. In general, aborting a task is not a normal operation. It is intended for exceptional situations.
- In LabVIEW, you also can abort a task by clicking the **Abort Execution** button. Doing so results in all tasks created in that VI hierarchy to be aborted and then

cleared.
- If you remove a device from the system, all tasks currently using the resources of that device are aborted.
- If you call the DAQmx Reset Device function/VI to restore a device to its initial configuration, all tasks currently utilizing the resources of that device are aborted.

When a task is aborted, it is returned to the Verified state. If the task is running, it is stopped as soon as possible and is then unreserved. After a task has been aborted, you can continue to use the task. However, you might need to transition the task back to its previous state before continuing the specified operation.

Related concepts:

- [Verified State](#)

Using Is Task Done

You can use the Is Task Done function/VI for applications in which you need to monitor the progress of a task running in one section of your application from another section of your application.

In general, use the Is Task Done function/VI with continuous measurements and generations when you are not actively reading or writing samples but want to monitor for errors.

Related concepts:

- [When Is A Task Done?](#)

Using Wait Until Done

You might need to call the Wait Until Done function/VI to ensure that the specified operation is complete before you stop the task.

The most common example is a finite generation. If you start a task that performs a finite generation and then immediately stop the task, the generation probably has not completed when you stop the task. As a result, the generation does not complete as expected. To ensure that the finite generation completes as expected, call the Wait

Until Done function/VI before stopping the task. After the Wait Until Done function/VI executes, the finite generation has been completed, and you can stop the task.

In general, use the Wait Until Task Done function/VI with finite measurements and generations.

Related concepts:

- When Is A Task Done?
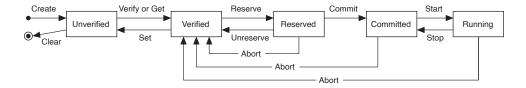
When Is A Task Done?

If the measurement or generation is finite, the task is done when you acquire or generate the final sample or when you call the Stop Task function/VI. If the measurement or generation is continuous (including on-demand timing), or if retriggering is enabled, the task is not done until you call the Stop Task function/VI. In addition, the task is done if a fatal error is generated while performing the measurement or generation, or you abort the measurement or generation. Check for errors and warnings to verify the task completed successfully.

Task State Model

NI-DAQmx uses a task state model to improve ease of use and speed up driver performance.
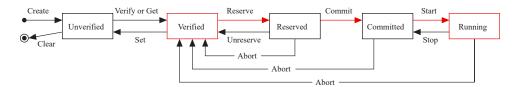
The task state model consists of five states—Unverified, Verified, Reserved, Committed, and Running. You call the Start Task function/VI, Stop Task function/VI, and Control Task function/VI to transition the task from one state to another. The task state model is very flexible. You can choose to interact with as little or as much of the task state model as your application requires.
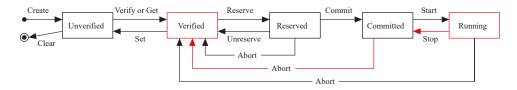
If you explicitly invoke a state transition that has already occurred, it is not repeated

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 Control Task function/VI with the `Action` parameter set to Reserve does not reserve the resources again.

Sometimes, calling a function/VI may require multiple state transitions, such as calling the Start Task function/VI while in the Verified state. In these cases, the task will implicitly transition between each of the necessary states to get to the final desired state, as shown in the following diagram.

Transitioning backwards in the Task State Model will undo any implicit forwards transitions in addition to the requested explicit transition. Continuing with the example above, calling the Stop Task function/VI after implicitly transitioning to the Running State from the Verified State will cause the task to return to the Verified State, as shown in the following diagram.

Unverified State

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 attributes/properties of the task.

Verified State

NI-DAQmx checks the timing, triggering, and channel attributes/properties for correctness when the task transitions from the Unverified to the Verified state. You can explicitly perform this transition by calling the Control Task function/VI with `Action` set to Verify. While NI-DAQmx detects and verifies some invalid values for attributes/properties immediately when you set the attribute/property, NI-DAQmx cannot verify other values immediately because they depend on other attributes/properties and the

devices being used. NI-DAQmx checks the value of these attributes/properties during the verify transition and reports any invalid values at that time. If NI-DAQmx finds no invalid values, the task is successfully verified and transitions to the Verified state. Otherwise, it remains in the Unverified state.

In certain cases, NI-DAQmx will coerce the values of attributes/properties when successfully verifying a task rather than generating an error. This is done when the value set on the attribute/property cannot be met exactly as specified and coercing it to a legal value has little functional impact on the task.

Related concepts:

- Coercion

Reserved State

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 Control Task function/VI with `Action` 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 State

NI-DAQmx programs some of the settings for the resources when the task is committed. These settings might be the rate of a 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 is started. When a task is committed, it transitions from the Reserved state to the Committed state. You can explicitly perform this transition by invoking the Control Task function/VI with `Action` 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 State

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 invoking the Start Task function/VI. Notice that starting a task does not necessarily start acquiring samples or generating a waveform. You might have specified the timing and triggering attributes/properties such that a sample is not acquired until you call the Read function/VI or a waveform is not generated 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 Committed state. If the task begins to perform the specified operation, the task is successfully started and transitions to the Running state.

Running to Committed State

The task ceases to perform the specified operation when the task transitions from the Running state to the Committed state. To explicitly perform this transition, call the Stop Task function/VI. Notice that you might have specified the timing and triggering attributes/properties such that all the samples are acquired before this transition occurs. For output operations, the last value written will typically continue to be generated after the task is stopped. In this situation, despite the fact that no additional samples are acquired, the task is still in the Running state until this transition occurs. In general, the stop transition does not fail. If it does, it is an exceptional condition, and the task is returned to the Reserved state. If the task is stopped, the task successfully transitions back to the Committed state.

Committed to Verified State

When the task resources that perform the specified operation are released, the task transitions from the Committed state to the Verified state. These resources may be clocks or channels on a device, trigger lines on a PXI chassis, or buffer memory in the computer. To explicitly perform this transition, call the Control Task function/VI with `Action` set to Unreserve. After the task releases all of its resources, it successfully transitions back to the Verified state.

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 attributes/properties, explicitly verify the task occasionally to inform the users if they have set an attribute/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 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 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. For more information, refer to Using the Start Task Function/VI.
- **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 time the 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.

Related concepts:

- Using the Start Task function/VI

Implicit Task State Transitions

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

- Moving the task through multiple states at the same time
- Operations that require state transitions

Related concepts:

- Task State Model
- Task Moves Through Multiple States at the Same Time
- Operations That Require State Transitions

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 Control Task function/VI, setting `Action` 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 is implicitly transitioned to the required state. Some operations that require state transitions include the following:

- Querying the value of an attribute/property implicitly verifies the task. This verification is required to return accurate coerced values of attributes/properties. Because the coerced value of a attribute/property often depends on the values of other attributes/properties, 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 an attribute/property, NI-DAQmx may return an error specifying that the value of attribute/property is invalid.
- Calling the Read function/VI implicitly commits the task if the task is not already committed. If the value of the DAQmx Read `Auto Start` attribute/property is True and the task has not been started, the task also is implicitly started. For more information regarding the auto-start behavior of read operations, refer to Using the Start Task Function/VI.
- Calling the Write function/VI commits the task. If the value of the `Auto-Start` parameter is True, the task also is started. For more information regarding the auto-start behavior of write operations, refer to Using the Start Task Function/VI.

For example, if the task is in the Reserved state, the value of the DAQmx Read `Auto Start` attribute/property is True, and you call the Read function/VI, the task is implicitly committed and started. The task transitions from the Reserved state to the Committed state and to the Running state before the read operation is performed.

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

Related concepts:

- [Using the Start Task function/VI](#)

Transitioning the State Backwards

When a task is implicitly transitioned backwards, it returns to the state of the task prior to the last operation that resulted in a forward state transition. For example, if the task was in the Verified state and you called the Start Task function/VI to start the task, the task is reserved, committed, and started, transitioning to the Reserved state and to the Committed state before transitioning to the Running state. When you invoke the Stop Task function/VI, the task is not just stopped and transitioned from the Running state to the Committed state. If this were the case, the result is unexpected because the task still has its resources reserved despite the fact that you never explicitly reserved them.

Instead, the task is stopped, uncommitted, and unreserved, returning to the Verified state, its state immediately before you performed the last operation that resulted in the state transition, calling the Start Task function/VI.

As another example, suppose the task is in the Reserved state, and you call the Read function/VI to perform a finite measurement. This results in the task implicitly transitioning from the Reserved state to the Committed state and then to the Running state before performing the read operation. When the read operation completes, the task does not remain in the running state. If this were the case, the result is unexpected behavior, because you need to stop the task and unreserve its resources despite the fact you never explicitly reserved the resources or started the task. Instead, after the finite read operation completes, the task is implicitly transitioned from the Running state to the Committed state to the Reserved state. This results in the task returning to the state before you performed the read operation.

Keep in mind that setting the value of a channel, timing, or triggering attribute/ property does not implicitly transition the task back to the Unverified state. Instead, the task remains in its current state and is implicitly verified when the next state transition occurs. For example, if the task is in the Reserved state and you set the value of timing attribute/property, the task remains in the Reserved state. The next time the task, either implicitly or explicitly, is committed, the task is verified. Because the task is implicitly verified when the next state transition occurs, NI-DAQmx can return an error specifying that the value of attribute/property is invalid.

Creating Channels and Tasks with the DAQ Assistant

You can launch the DAQ Assistant from your NI application software or from MAX. The DAQ Assistant is a graphical interface for configuring channels, tasks, and scales.

After you launch the DAQ Assistant, follow the wizard instructions to create your new task or channel. When the wizard is done, you can configure measurement-specific settings, scaling, and, if necessary, timing and triggering.

Related concepts:

- [Choosing Whether to Use the API or the DAQ Assistant](#)

LabVIEW

In LabVIEW, there are several ways to open the DAQ Assistant. A couple of common ones are the following:

- Drop the DAQ Assistant Express VI from the Express Input palette.
- Use the DAQmx Task Name control to open the DAQ Assistant.
- Open the DAQ Assistant from within a LabVIEW Project as described in ***Using NI-DAQmx with LabVIEW Project*** in the ***LabVIEW Help***.

LabWindows/CVI

In LabWindows/CVI, select **Tools»Create/Edit DAQmx Tasks**. You also can launch the DAQ Assistant by clicking the **Task Name** control of the DAQmx LoadTask function panel and selecting **New Task**.

Measurement Studio

In Measurement Studio, open Visual Studio .NET and select **Project»Add New Item** to open the Add New Item dialog box. In the Categories pane, select **Measurement Studio»Assistants**. In the Templates pane, select **DAQmx Task Class**.

MAX

In MAX, right-click **Data Neighborhood**, and select **Create New** from the shortcut menu. Select **NI-DAQmx Task** or **NI-DAQmx Global Virtual Channel** in the **Create New** window, and click **Next**.

Signal Express

In SignalExpress, add a DAQmx Acquire or DAQmx Generate step.

Choosing Whether to Use the API or the DAQ Assistant

When creating a new application, you can choose to use DAQ Assistant or the API.

Advantages of Using the DAQ Assistant:

- The DAQ Assistant requires no programming. You can configure channels, timing, triggering, and scales interactively.
- The DAQ Assistant can decrease development time. You can create a complete application in a matter of minutes.
- If you create your application using the DAQ Assistant and later need functionality that it doesn't expose, you can easily generate the equivalent API code from your DAQ Assistant task if you use an NI ADE such as LabVIEW, LabWindows/CVI, or Measurement Studio.

Advantages of Using the API:

- The API contains advanced features not exposed by the DAQ Assistant.
- The API provides additional flexibility, allowing you to customize your application to suit your needs.
- The API gives you tighter control over the performance of your application.

Timing and Triggering

Timing and triggering are important in NI-DAQmx. The clocks section explains clocks and handshaking. The triggering section goes over the triggers—such as a Start Trigger and a Reference Trigger—and common trigger types—such as an analog edge trigger or a digital edge trigger.

Related concepts:

- [Clocks](#)
- [Triggering](#)

Timing, Hardware Versus Software

You can use software timing or hardware timing to control when a signal is generated. With hardware timing, a digital signal, such as a clock on your device, controls the rate of generation. With software timing, the rate at which the samples are generated is determined by the software and operating system instead of by the measurement device. A hardware clock can run much faster than a software loop. A hardware clock is also more accurate than a software loop.

In NI-DAQmx, select hardware timing with the Sample Clock Timing function/VI or by setting the Sample Timing Type attribute/property to Sample Clock. If you do neither of these things, or you set the Sample Timing Type attribute/property to On Demand, you are selecting software timing.

> **Note** Some devices do not support hardware timing. Refer to your device documentation if you are unsure whether your device supports hardware timing.

Clocks

Periodic digital edges measure time and are called clocks. Clocks such as a sample timebase clock and the 20 MHz timebase clock mark the passing of time or are used to align other signals in time. Clocks usually do not cause actions in the sense that triggers do. The names of clocks usually do not refer to actions. The sample clock is a notable exception.

The following are some common clocks used by DAQ devices. Refer to your device documentation for all the clocks on your device.
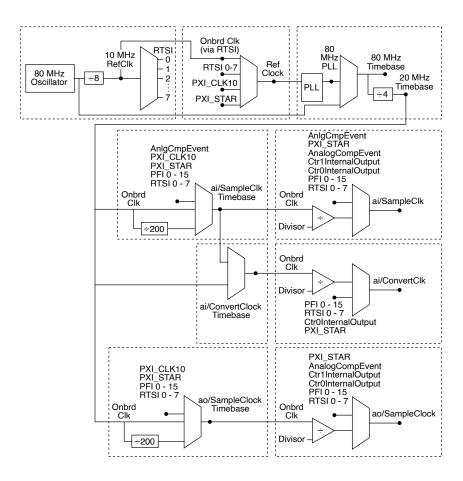
- **AI Convert Clock**—The clock on a multiplexed device that directly causes ADC conversions. The default AI Convert Clock rate uses 10 μs of additional settling time between channels, compared to the fastest AI Convert Clock rate for the device. When the Sample Clock rate is too high to allow for 10 μs of additional settling time, the default AI Convert Clock rate uses as much settling time as is allowed by the Sample Clock rate. If there are multiple devices in the same task, the same amount of additional settling time is used for all devices in the task, even if their maximum AI Convert Clock rates differ.
- **AI Convert Clock Timebase**—The clock that is divided down to produce the AI convert clock.
- **AI Sample Clock**—The clock that controls the time interval between samples. Each time the sample clock ticks (produces a pulse), one sample per channel is acquired.
- **AI Sample Clock Timebase**—The onboard clock used as the source of the AI sample clock. The AI Sample Clock Timebase is divided down to produce the AI sample clock.

- **Counter Timebase**—The clock connected to the source terminal of a counter (Ctr0Source, for example).
- **DI Sample Clock**—The clock that controls the time interval between samples. Each time the sample clock ticks (produces a pulse), one sample per channel is acquired.
- **DO Sample Clock**—The clock that controls the time interval between samples. Each time the sample clock ticks (produces a pulse), one sample per channel is acquired.
- **DO Sample Clock Timebase**—The onboard clock used as the source of the DO sample clock. The DO Sample Clock Timebase is divided down to produce the DO sample clock.
- **Master Timebase**—An onboard clock used by other counters on the device. The master timebase is divided down to produce a slower clock or to measure elapsed time. This timebase is the onboard clock used as the source of the AI Sample Clock timebase, the AO Sample Clock timebase, and the counter timebases, for example.
- **12.8 MHz Timebase**—The onboard clock source for the master timebase from which other timebases are derived. This timebase is often used to synchronize tasks across chassis.
- **13.1072 MHz Timebase**—The onboard clock source for the master timebase from which other timebases are derived. This timebase is often used to synchronize tasks across chassis.
- **20 MHz Timebase**—The onboard clock source for the master timebase from which other timebases are derived, if the device does not support an 80 MHz Timebase. Otherwise, the clock produced by dividing the 80 MHz Timebase by 4.
- **80 MHz Timebase**—The onboard clock source for the master timebase from which other timebases are derived.
- **100 MHz Timebase**—The onboard clock source for the master timebase from which other timebases are derived.
- **100 kHz Timebase**—The clock produced by dividing the 20 MHz Timebase by 200.
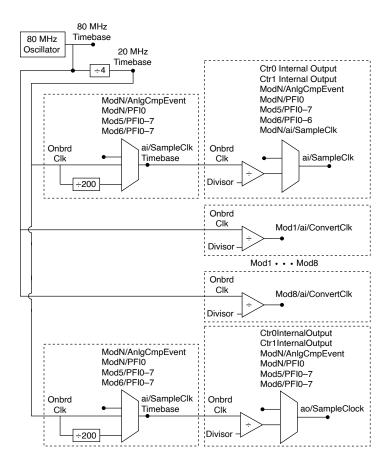
> **Note** M Series, C Series, and X Series devices do not have a master timebase of an arbitrary frequency. These devices use the 20 MHz/80 MHz/100 kHz timebase directly.
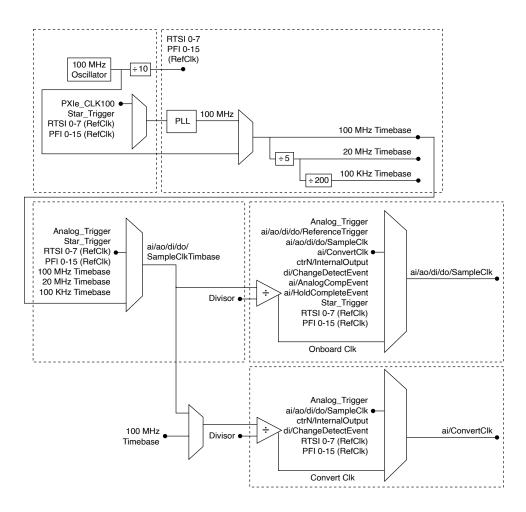
The following diagram illustrates the M Series clocks that comprise analog input and analog output timing. The black circles in the diagram represent terminals.
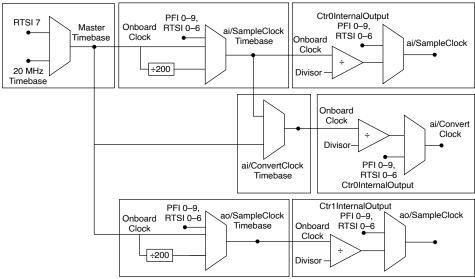
The following diagram illustrates the C Series clocks that comprise analog input and analog output timing.

The following diagram illustrates the X Series clocks that comprise analog input, analog output, digital input, and digital output timing. The black circles in the diagram represent terminals.

The following diagram illustrates the E Series clocks that comprise analog input and analog output timing. The black circles in the diagram represent terminals.

Related concepts:

- Multiplexed Versus Simultaneous Sampling

Trigger and Clock Distinction

The distinction between triggers and clocks is blurred when the digital edges used as a trigger are periodic. In such a case, a clock causes the device to perform an action. The sample clock is the primary example. The stimulus for the action of producing a sample is so often a clock that NI-DAQmx configures the sample clock instead of the sample trigger. The distinction is made clear when you consider the sample clock is in fact just one way of providing the source of a sample trigger.

Sample Timing Types

NI-DAQmx introduces the concept of a ***sample timing type.*** Each sample timing type is a different stimulus for triggering the action of producing a sample. When you select a Timing function/VI, you select your sample timing type. There also is an attribute/property for setting the following sample timing types:

- **Sample Clock**—A digital edge produces each sample. Nearly all devices have an onboard clock that is dedicated to producing these edges periodically. Even when the edges are not periodic, as they might be when the clock source is something other than the dedicated onboard clock, you still use sample clock timing. Sample clock timing is a type of hardware timing.
- **On Demand**—Every time the Read or Write function/VI executes, the device produces the requested samples as fast as possible. In this mode, the Sample Quantity attributes/properties are ignored. On-demand timing is a type of software timing.
- **Change Detection**—Change detection timing captures samples from digital physical channels when NI-DAQmx detects a change-a rising edge, a falling edge, or both rising and falling edges-on one or more digital lines or ports. Change detection timing reduces the digital data an application has to process. One issue to be aware of with change detection on some devices is overflow. Overflow occurs when NI-DAQmx cannot read a sample prior to the next change detection event. The effect is that one or more samples can be missed.

  Programmatically, you include the Change Detection Timing function/VI, specifying the physical channels for rising and falling edges on which to detect

changes. You can query for an overflow by using the Overflow attribute/property in your application after the task starts.

- **Handshake**—The handshake sample timing type is used to acquire or generate digital data with the 8255 protocol. Many devices have an 8255 chip, and other devices emulate the 8255 protocol by default with the handshake timing type.
- **Burst Handshake**—Burst handshake timing acquires or generates digital data on the data lines with a clocked protocol. This timing type involves three control signals: the sample clock, the Pause Trigger, and the Ready for Transfer Event. Data is transferred on each active sample clock edge if the peripheral device deasserts the Pause Trigger and the DAQ device asserts the Ready for Transfer Event.

  There are separate Burst Handshake Timing functions/VIs based on whether you import or export a sample clock. Using the appropriate function/VI is important because there are timing restrictions (such as setup and hold times) when sharing a clock between the two devices.

- **Implicit**—The implicit sample timing type is used for acquiring period or frequency samples using counters. It is also used for generating pulses. This timing type is called *implicit* because the signal being measured is itself the timing signal or the timing is implicit in the rate of the generated pulse train.

Related concepts:

- Timing, Hardware Versus Software
- Change Detection Considerations for NI 6527 Devices
- Sample Clock
- Pause Trigger
- Events
- Setup and Hold Times

Sample Clock

Your device uses a sample clock to control the rate at which samples are acquired and generated. This sample clock sets the time interval between samples. Each tick of this clock initiates the acquisition or generation of one sample per channel. You also can connect an outside source as your clock. In software, you can specify the interval (how fast the clock acquires or generates signals) by specifying the sample rate. You can limit the sample rate by the signal conditioning you apply to the signals or the number

of channels in your application. However, the number of channels affects your measurement only if you are sampling close to the maximum sample rate for your device.

> 📝 **Note** Sample clock timing for digital I/O is not supported on all devices.

Related reference:

- [Sample Clock Timing for Digital I/O](#)

Handshaking

If you want to communicate with an external device using an exchange of signals to request and acknowledge each data transfer, use handshaking.

For example, you might want to acquire an image from a scanner. The process involves the following steps:

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

This process repeats until all the samples are transferred.

> 📝 **Note** Not all devices support handshaking. Refer to 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 devices that have an additional 8255 chip onboard—support handshaking.

Related concepts:

- [Handshaking Signals for Devices That Emulate the 8255 Protocol](#)

Burst Handshaking Signals

For devices that support burst handshake timing, three signals are used:

- Pause Trigger (formerly called REQ)
- Ready for Transfer Event (formerly called ACK)
- sample clock

For digital input tasks, when the Pause Trigger signal is logic low and the Ready for Transfer Event is logic high, the samples are sent to the measurement device. For digital output tasks, when the Pause Trigger signal is logic low and the Ready for Transfer Event is logic high, the NI-DAQmx device sends the samples to a peripheral device. The sample clock, either onboard or external, controls the timing. Data is transferred or acquired on either the rising or falling edge of the sample clock.

The default terminals used for burst handshaking signals vary from device to device.

Related concepts:

- [Burst Handshaking Timing Defaults for NI 653x Devices](#)

Handshaking Signals for Devices That Emulate the 8255 Protocol

Devices that emulate the 8255 protocol support two handshaking signals:

- **Handshake Trigger**—Also called Strobe Input (STB) and Acknowledge Input (ACK)
- **Handshake Event**—Also called Input Buffer Full (IBF) and Output Buffer Full (OBF)

For input tasks, when the Handshake Trigger signal is low, the samples are sent to the measurement device. After the samples have been sent, Handshake Event is high, which tells the peripheral device that the data has been read. For digital output, Handshake Event is low while the NI-DAQmx device sends the samples to a peripheral device. After the peripheral device receives the samples, it sends a low pulse back on the Handshake Trigger line. Refer to your device documentation to determine which digital ports you can configure for handshaking signals.

The default terminals used for handshaking signals vary from device to device.

Related concepts:

- [Handshake Timing Defaults](#)

Handshaking Signals for 8255-Based Devices

8255-based devices that perform handshaking support four handshaking signals:
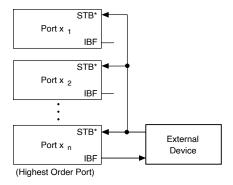
- Strobe Input (STB)
- Input Buffer Full (IBF)
- Output Buffer Full (OBF)
- Acknowledge Input (ACK)

Use the STB and IBF signals for digital input operations and the OBF and ACK signals for digital output operations. When the STB line is low, the samples are sent to the measurement device. After the samples have been sent, IBF is high, which tells the peripheral device that the data has been read. For digital output, OBF is low while the software sends the samples to a peripheral device. After the peripheral device receives the samples, it sends a low pulse back on the ACK line. Refer to your device documentation to determine which digital ports you can configure for handshaking signals.
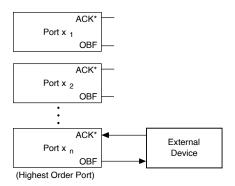
Digital Data on Multiple Ports

For 8255-based devices, the ports in the task affect which handshaking lines are used. Always use the handshaking lines associated with the highest order port in the task. For instance, if you want to group ports 1 and 2 into a single task, use the handshaking lines associated with port 2.

Connect all the STB lines together if you are grouping ports for digital input, as shown in the following figure. Connect only the IBF line of the highest order port in the task to the other device. No connection is needed for the IBF signals for the other ports.

If you group ports for digital output on an 8255-based device, connect only the handshaking signals of the last port in the port list, as shown in the following figure.

When performing handshaking, some lines are automatically reserved for control purposes and are unavailable for use. The control lines used depend on the ports you are using and whether you are handshaking with input or output channels. The remaining lines in the port not used for control are still available for use. If you are transferring data across any line in a port in a handshaking task, the entire port is reserved for handshaking data and the remaining lines in the port are unavailable for use.

Hardware-Timed Single Point Sample Mode

In hardware-timed single-point sample mode, samples are acquired or generated continuously using hardware timing and no buffer. You must use the sample clock or change detection timing types. No other timing types are supported.

Use hardware-timed single-point sample mode if you need to know if a loop executes in a given amount of time, such as in a control application.

Because there is no buffer if you use hardware-timed single-point sample mode, ensure that reads or writes execute fast enough to keep up with hardware timing. If a read or write executes late, it returns a warning.
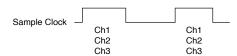
Continuous Pulses (HW Timed Updates) is hardware-timed single point for counter output.

Related concepts:
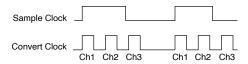
- Timing, Hardware Versus Software
- Hardware-Timed Counter Tasks

Multiplexed Versus Simultaneous Sampling

Devices use either multiplexed or simultaneous sampling. Simultaneous sampling devices have an ADC for each analog channel and can sample from all channels at the same time, as shown in the following figure.

Multiplexed sampling devices have a single ADC for all analog input channels. These devices use both a sample clock and a convert clock. The sample clock initiates the acquisition of a sample from all channels in the scan list. The convert clock causes the ADC conversion for each individual channel. The following figure depicts a three-channel analog input task on a device that uses multiplexed sampling. Notice that, unlike S Series devices, the samples are not digitized simultaneously.

The convert clock must run faster than the sample clock to achieve the specified sample rate. For instance, if you specify a sample rate of 10 S/s for 8 analog input channels, the convert clock must run at least eight times the sample rate (80 Hz) to ensure that each channel is sampled 10 times a second. At faster sampling rates, you

must also take settling time between channels into account.

Related concepts:

- Clocks
- Device-Specific Sampling Methods

Setup and Hold Times

When a DAQ device samples a digital signal, the signal must remain stable for a period of time before and after the assertion of the clock edge used for timing. The amount of time before the assertion of the clock is called the ***setup time***. The amount of time after the assertion of the clock edge is called the ***hold time***. Refer to your device documentation for minimum setup and hold times.

Simultaneous Analog Output On-Demand Timing

Typically, when you use software timing to output samples on multiple AO channels, NI-DAQmx writes a sample to the first DAC, and the sample is generated. Then, NI-DAQmx writes a sample to the second DAC, and that sample is generated, and so on. However, with the simultaneous single-point on-demand timing, all of the data is generated at the same time after NI-DAQmx writes to each DAC. You set this timing with the Simultaneous Analog Output Enable attribute/property.

Timing Response Modes

Digital I/O and DAQ devices typically use the single-cycle timing response mode, meaning the device responds to an external signal by the next active sample clock edge.

Devices that support the pipelined timing response mode, such as the PCIe-6536 and PCIe-6537, can respond to an external signal a few sample clock edges later. This mode uses a source-synchronous clock scheme, which simultaneously returns the clock and data to the acquiring device. With a source-synchronous data transfer, you can acquire and generate data at much higher rates than with single-cycle timing response mode.

With the pipelined timing response mode, you can configure external sample clocks,

but the sample clock must be free-running and started before the task commits. If you export the sample clock, the export occurs during a task commit. As with other events, when the task uncommits, the signal remains exported.

Triggering

When a device controlled by NI-DAQmx does something, it performs an action. Two very common actions are producing a sample and starting a waveform acquisition. Every NI-DAQmx action needs a stimulus or cause. When the stimulus occurs, the action is performed. Causes for actions are called triggers. Triggers are named after the actions they cause:

- Advance Trigger
- Expiration Trigger
- Handshake Trigger
- Pause Trigger
- Reference Trigger
- Start Trigger
- Arm Start Trigger

In addition to specifying the action you want a trigger to cause, you must select the type of trigger to use, which determines how the trigger is produced.

Advance Trigger

An Advance Trigger causes a switch device to execute the next entry in its instruction (scan) list. You can configure this trigger to occur on a digital edge or when the Send Software Trigger function/VI runs.

Arm Start Trigger

When you configure an Arm Start Trigger, a counter task does not respond to any Start Triggers until after the Arm Start Trigger occurs. You can configure this trigger to occur on a digital edge or at a specified time (for devices that support time triggering). The Arm Start Trigger is separate from a Start Trigger and is typically used in advanced counter/timer applications. You might use an Arm Start Trigger to synchronize multiple tasks, such as counting edges and pulse generation. The Start Trigger then would be

used to start the acquisition or generation.

Expiration Trigger

An Expiration Trigger expires a watchdog task. You can use this trigger instead of the watchdog timer to signal an expiration. You can configure this trigger to occur on a digital edge.

Related concepts:

- [Watchdog Timers](#)

Handshake Trigger

A Handshake Trigger is a control signal from a peripheral device. The peripheral device asserts the Handshake Trigger to indicate to the DAQ device that it has acquired a sample (for output tasks) or generated a sample (for input tasks). For input tasks, the DAQ device latches data, by default, at the trigger position specified by the Sample Input Data When attribute/property, or when the peripheral device asserts the Handshake Trigger.

Pause Trigger

With sample clock timing or burst handshake timing, the Pause Trigger pauses an ongoing acquisition or generation. Deasserting this trigger resumes an acquisition or generation. Depending on your device, there are some additional issues you need to remember.
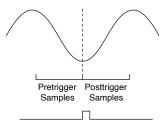
Related concepts:

- [Pause Triggering](#)

Reference Trigger

A Reference Trigger establishes the reference point in a set of input samples. You can configure this trigger to occur on a digital edge, a digital pattern, an analog edge, or when an analog signal enters or leaves a window. Data acquired up to the reference

point is pretrigger data. Data acquired after this reference point is posttrigger data.

Pretrigger Samples Posttrigger Samples

Start Trigger

A Start Trigger begins an acquisition or generation. You can configure this trigger to occur on a digital edge, a digital pattern, an analog edge, when an analog signal enters or leaves a window, or at a specified time (for devices that support time triggering).
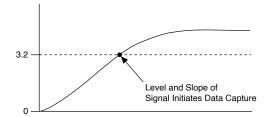
Trigger Types

In addition to specifying the action you want a trigger to cause, you must select the type of trigger to use, which determines how the trigger is produced. 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, choose a digital edge trigger with the source typically being one of the PFI pins.

Analog Edge Triggering
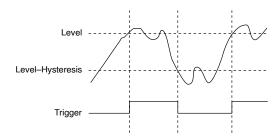
For analog edge triggering, you configure the measurement device to look for a certain signal level and slope (either rising or falling). After the device identifies the trigger condition, the device performs the specified action associated with the trigger, such as starting the measurement or marking which sample was acquired when the trigger occurred. You connect analog trigger signals to any analog input channel or terminal capable of accepting analog signals. Refer to the device-specific analog triggering considerations for your device for additional information.

In the following figure, the trigger is set to capture data for a rising edge signal when the signal reaches 3.2.

3.2

0

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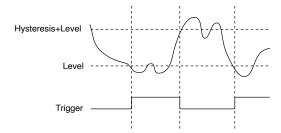
Hysteresis adds a window above or below the trigger level and often is used to reduce false triggering due to noise or jitter in the signal. When using hysteresis with a rising slope, the trigger asserts when the signal starts below `level` (or `threshold level`) minus `hysteresis` and then crosses above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

For example, if you add a `hysteresis` of 1 to the previous example, which used a `level` of 3.2, the signal must start at or drop below 2.2 for triggering to occur. The trigger then asserts as the signal rises above 3.2 and deasserts when it falls below 2.2.

Level

Level–Hysteresis

Trigger

When using hysteresis with a falling slope, the trigger asserts when the signal starts or rises above `level` (or `threshold level`) plus hysteresis and then crosses below `level`. The trigger deasserts when the signal crosses above `level` plus `hysteresis`. If you instead trigger on a falling edge at 3.2 with a hysteresis of 1, the signal must start at or rise above 4.2 and then fall below 3.2 for triggering to occur. The trigger will then assert as the signal falls below 3.2 and deassert when it rises above 4.2.

Hysteresis+Level

Level

Trigger

Related concepts:
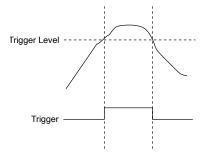
- Analog Triggering

Analog Level Triggering

An analog level trigger is similar to an analog edge trigger. With both trigger types, you specify the edge—rising or falling—and the trigger level. With an analog edge trigger, you are interested in the point at which the trigger condition is met. With an analog level trigger, on the other hand, you are interested in the *duration* that the signal remains above or below the trigger level. An analog level trigger is typically used with a Pause Trigger. The Pause Trigger asserts or deasserts when the trigger condition is met. In the following illustration, a trigger asserts when the signal crosses above the trigger level and deasserts when it drops below it. The deassertion of the trigger could correspond to a Pause Trigger.
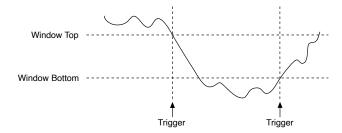
Analog Multi Edge Triggering
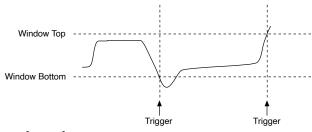
An analog multi edge trigger functions similar to an analog edge trigger. The analog multi edge trigger differs in that you can configure multiple channel sources acquiring in the same task to each have their own analog trigger condition. When any of the multiple channel sources satisfy their respective condition, a trigger is generated and the device will perform the specified action associated with the trigger.

Analog Window Triggering

A window trigger occurs when an analog signal either passes into (enters) or passes out of (leaves) a window defined by two voltage levels. Specify the levels by setting the window top value and the window bottom value. The following image demonstrates a trigger that acquires data when the signal enters the window.

The following image demonstrates a trigger that acquires data when the signal leaves the window.
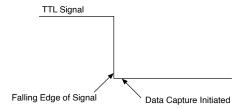
Related concepts:

- Analog Triggering Considerations for TestScale Modules and C Series, E Series, M Series, and S Series Devices
- Analog Triggering Considerations for SC Express Devices

Digital Edge Triggering

A digital trigger is usually a TTL signal with two discrete levels: a high and a low level. When the signal moves from high to low or from low to high, a digital edge is created. There are two types of edges: rising and falling. You can produce Start or Reference Triggers from the rising or falling edge of your digital signal.

In the following figure, the acquisition begins after the falling edge of the digital trigger signal. Usually, digital trigger signals are connected to PFI pins on your measurement device.

TTL Signal

Falling Edge of Signal   Data Capture Initiated

Digital Level Triggers

Digital level triggering starts, stops, or pauses an acquisition or measurement based on the value read on a digital line.
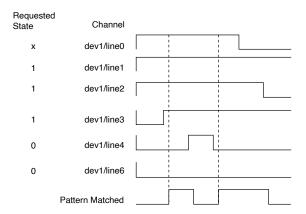
Digital Pattern Triggering

For digital pattern triggering, you configure the device to detect a specific digital pattern on specific physical channels. After detecting this condition, the device performs the action associated with the trigger, such as starting the task or marking which sample was acquired when the trigger occurred.

The digital pattern is specified using the following characters:

- X: ignore the physical channel
- 0: Match on a logic low level on the physical channel
- 1: Match on a logic high level on the physical channel
- R: Match on rising edge on the physical channel
- E: Match on either rising or falling edge on the physical channel
- F: Match on falling edge on the physical channel

For instance, if you specify a pattern of "X11100" and a source of "dev1/line0:4,dev1/line6," the pattern match occurs when physical channels "dev1/line1," "dev1/line2," and "dev1/line3" are logic high and when physical channels "dev1/line4" and "dev1/line6" are logic low. "dev1/line0" is ignored.

For pattern triggers on ports, the pattern match occurs in reverse order. For instance, if you specify a pattern of "11000000" and a source of "dev1/port0," the pattern match occurs when physical channels "dev1/line0" and "dev1/line1" are logic high and the other six lines are logic low.

Software Triggers

Software triggering starts, stops, or pauses an acquisition or measurement or advances a scan list based on a software trigger command being sent. You generate a software trigger command with the DAQmx Send Software Trigger function/VI.

Time Triggering

For devices that support it, a time trigger starts an acquisition or measurement at a specific time. If the specified time has already elapsed, you will get an error message indicating the time has already elapsed.

The timestamp timescale can be configured using the Timestamp.Timescale attribute/property. Time triggers and timestamps can be specified in I/O Device Time or Host Time, depending on the needs of your application.

- I/O Device Time

  Shared by all network-synchronized devices on your 802.1AS subnet. I/O Device Time is most useful for synchronizing events across multiple chassis or correlating timestamps from multiple chassis, because even though it may be in an obscure time scale (for example, related to a point in the distant past, such as the Linux 1970 epoch), it removes other sources of skew related to Windows system time or other systems that are not network-synchronized to the same 802.1AS subnet. In

that way, the I/O Device Time provides the best precision and relative accuracy but may reduce usability if it is not correlated to a recognizable global time. I/O Device Time also has the advantage of being monotonically increasing, so time triggers and timestamps spread across multiple devices or tasks accurately maintain their offsets from each other.

- Host Time

The timescale your PC or NI Linux Real-Time controller uses. In cases where the NI Linux Real-Time controller is the Grand Master of your 802.1AS subnet, Host Time and I/O Device Time are the same. However, Host Time is typically synchronized to a local Real Time Clock or a Network Time Protocol server, and it is usually traceable to global time. Using Host Time is more intuitive because triggers and timestamps on the chassis are specified in times that are easily correlated to your local system time. However, this usability comes at the cost of reduced relative accuracy between time triggers and timestamps that are spread across multiple devices or tasks, because using the calculated offset between the two timescales is not as accurate as using I/O Device Time directly. To help account for this loss of accuracy in a specific and common use-case, NI-DAQmx guarantees that two events that are scheduled for the same Host Time are guaranteed to start at the same I/O Device Time, preserving precise synchronization between chassis.

Whether a device supports time triggers or not can be queried using `TimeTrigSupported` attribute/property.

Network-synchronized devices include the cDAQ-9185, 9189; FD-11601, FD-11603, FD-11605, FD-11613, FD-11614, FD-11634, FD-11637; cRIO-9040, 9041, 9042, 9043, 9045, 9046, 9047, 9048, 9049, 9053, 9054, 9055, 9056, 9057, 9058; and sbRIO-9603, 9608, 9609, 9628, 9629, and 9638.

Related concepts:

- [Time-Based Features for Network-Synchronized Devices](#)

Synchronization

Synchronized operations are created by routing timing and control signals.

Synchronization can be within a single device—for instance, synchronizing analog input and analog output on an M Series device—or on multiple devices. Timing and control signals that synchronize operations fall into three categories: clocks, triggers, and events.

These timing and control signals are routed by connecting two terminals together. Selecting a terminal as the source of a clock or a trigger constructs a route. On PCI devices, the RTSI bus provides the pathways for signal routing. On PXI devices, the PXI trigger bus provides the same pathways. For NI-DAQmx to find a free PXI trigger line, you must perform a PXI chassis identification in MAX. For NI-DAQmx to find a free RTSI line, you must create a RTSI cable in MAX and populate it with the devices connected by the cable. You can discover what routes are possible by referencing a table of possible routes in MAX.

On some devices, you synchronize analog input, analog output, and digital input/output channels from multiple modules by including those channels in the same task. All channels within a task must be of the same channel type, such as analog input or counter output.

Related concepts:

- Clocks
- Triggering
- Events
- Multidevice Tasks

Types of Synchronization, Lockstep and Handshaked

Lockstep synchronization involves two or more similar devices sharing the same timing and triggering and essentially acting as a single device. Sharing a sample clock between analog input and analog output operations on a single device is also considered lockstep synchronization. The goal of lockstep synchronization is to eliminate skew as much as possible. In lockstep synchronization, clocks and triggers are typically shared.

Handshaked synchronization (or stimulus/response) is two or more devices acting in sequence. In handshaked synchronization, triggers and events are typically shared. A simplified DAC test is an example of this type of synchronization. A digital device sends

- Sample Clock Synchronization
- Reference Clock Synchronization

Sources of Error

There are several sources of error when synchronizing measurements:

Jitter

Jitter is small variations in the period of the clock (from sample to sample). It shows up as noise in the digitized signal and affects higher-frequency signals more. Each component added to the clock's path adds additional jitter. You can control jitter but not eliminate it by using an accurate clock source.

Stability

Stability describes how well the clock frequency resists fluctuations. Factors that can cause the frequency to fluctuate include variations in temperature, time (aging), supply voltage, shock, vibration, and capacitive load that the clock must drive. Temperature is often the dominant factor that affects crystal oscillator stability.

Some oscillators are housed inside small ovens with controlled temperature to provide stability that can be orders of magnitude better than with other techniques. These oscillators are known as oven controlled crystal oscillators (OCXOs). For example, the NI 6608 contains an OCXO.

Accuracy

Clock accuracy describes how well the actual frequency of the clock matches the specified frequency. An oscillator generates a clock. However, an oscillator never generates a perfect frequency. The accuracy of the oscillator-generated clock is affected by the quality of the crystal and the oscillator's assembly.

You can describe timing errors in several different ways. Some common units of timing error are parts per million (ppm) and parts per billion (ppb). Parts per million gives you a fractional value of error. For example, to find the error in Hertz of an 80 MHz oscillator with 5 ppm error, you multiply the frequency of the oscillator—80,000,000—by 5 divided by 1,000,000 or [80,000,000 Hz (5 Hz/1,000,000 Hz) = 400 Hz].

From this equation, you see that the oscillator can be off by as much as 400 Hz. Therefore, the actual frequency of the oscillator can be anywhere between 79,999,600 Hz and 80,000,400 Hz. Parts per billion is similar to parts per million, and it is used to describe more accurate clocks.

Skew

Skew is a propagation delay that is caused when a signal arrives at two places at different times. For instance, a signal is sent by a controlling device at time T0. A receiving device A acts upon the signal at time T1. A receiving device B acts upon the signal at time T2. If T1 is not equal to T2, the difference between T1 and T2 is the skew. The distance between devices and the cabling between your devices and signal paths within the devices themselves all affect signal arrival times.

Methods of Synchronization

Several methods exist for synchronizing devices, depending on the devices involved and the requirements of the application.

Start Trigger Synchronization

For synchronizing multiple tasks on a single device, even at different rates and on different subsystems, you do not need to synchronize any clocks. Because the device derives those clocks from the same internal oscillator, you need to share only the Start Trigger among the tasks so the clocks start at the same time.

To perform Start Trigger synchronization, configure start triggering on all slave tasks, setting the trigger source to the internal Start Trigger terminal from the master task, such as ai/StartTrigger. You do not have to configure start triggering on the master task. All tasks include an implicit Start Trigger, which occurs immediately when the task starts.

> **Note** You must start any tasks on slave devices before starting the task on the master device.

Related concepts:

- Master and Slave Devices

Sample Clock Synchronization

Sample Clock synchronization allows you to synchronize tasks on multiple devices at the same rate.

When using Sample Clock synchronization, slave devices replace the Sample Clock for a given subsystem with the Sample Clock from another device.

To perform Sample Clock synchronization, configure all devices to use Sample Clock timing. Set the source of the Sample Clock on all slave devices to the internal Sample Clock terminal from the master device. For example, the name of the Sample Clock terminal from the AI subsystem is `ai/SampleClock`.

> ✎ **Note** You must start any tasks on slave devices before starting the task on the master device.

Sample Clock synchronization results in skew due to the time required for the clock to travel between devices. On multiplexed devices, Sample Clock synchronization results in jitter because, even though you synchronized the Sample Clock, the devices do not synchronize the AI Convert Clock.

Use Master Timebase synchronization or Reference Clock synchronization to synchronize devices at different rates.

Related concepts:

- Skew
- Jitter
- Master Timebase Synchronization
- Reference Clock Synchronization
- Master and Slave Devices

Reference Clock Synchronization

Reference Clock synchronization is the most flexible and powerful synchronization

method available on supported devices. Reference Clock synchronization allows you to synchronize all timing for the synchronized devices, even at different rates and regardless of subsystem, in that clocks derived from the Reference Clock start and remain in phase. Derived clocks with a slower frequency than the Reference Clock are not in phase. For counter operations, Reference Clock synchronization ensures the counter timebases remain synchronized without drift, or in phase if the application requires different counter timebase frequencies.

When using Reference Clock synchronization, a device does not directly use a clock from another device in place of an onboard clock. Instead, all devices synchronize their onboard oscillators to a common reference signal using a phase-locked loop. Each device then derives other clocks from the synchronized oscillators. You must share a Start Trigger for the derived clocks to start in phase.

For PXI devices, the reference signal is typically a 10 MHz clock on the chassis backplane (PXI_Clk10). For PXI Express devices, the reference signal is typically a 100 MHz clock on the chassis backplane (PXIe_Clk100).

> **Note** Always use one of the PXI or PXI Express chassis backplane clocks, if possible. Using a clock from another device results in skew due to the time required for the signal to travel from one device to the other.

PXI or PXI Express chassis backplane clocks might provide different accuracy than the onboard oscillator of a device. For example, the PXIe-1062Q chassis has 25 ppm clock accuracy, while the PXI-6259 has 50 ppm clock accuracy.

For PCI and PCI Express devices, that reference signal is a clock from another device (typically 10MHzRefClk). Use the RefClk.Src attribute/property to specify the terminal of the reference signal for a given task. Set RefClk.Src to `OnboardClock` on the master device to lock to the onboard oscillator, rather than use it directly. Locking to the onboard oscillator helps to equalize skew between the master and slave devices.

Even though Reference Clock synchronization minimizes or eliminates skew in the clocks, the shared Start Trigger must travel from the master device to the slave devices, resulting in skew. Some devices allow you to correct for that skew.

Some devices use a Master Timebase instead of a Reference Clock, thus they use

Master Timebase synchronization. Reference Clock synchronization also requires you to share multiple signals and reserve multiple RTSI or PXI trigger lines for those signals. For Sample Clock-timed applications where all devices run at the same rate, you can use Sample Clock synchronization to eliminate the need for a shared Start Trigger, thus the additional RTSI/PXI line. You can also use Sample Clock synchronization to synchronize devices that use a Master Timebase with devices that use a Reference Clock.

Related concepts:

- Synchronization
- Skew
- Accuracy
- Trigger Skew Correction
- Master Timebase Synchronization
- Sample Clock Synchronization
- Master and Slave Devices

Master Timebase Synchronization

Master Timebase synchronization is the most flexible and powerful synchronization method available on supported devices. It allows you to synchronize all timing for the synchronized devices, even at different rates and regardless of subsystem, in that clocks derived from the Master Timebase start and remain in phase. For counter operations, Master Timebase synchronization ensures the counter timebases remain synchronized without drift, or in phase if the application requires different counter timebase frequencies.

When using Master Timebase synchronization, slave devices replace their onboard oscillator (the Master Timebase) with the Master Timebase from a master device. Each device then derives other clocks from the synchronized oscillators. You must share a Start Trigger for the derived clocks to start in phase.

> ✏️ **Note** You must start any tasks on slave devices before starting the task on the master device.

You must set two properties on the slave devices to synchronize the Master Timebase.

Use the MasterTimebase.Src attribute/property on the slave devices to specify the terminal of the Master Timebase from the master device. Because the slave devices then use an external timebase, you must also use the MasterTimebase.Rate property to specify the rate of the Master Timebase. Instead of hard-coding those values, you can query the Master Timebase rate and source from the master device and set the rate and source on the slave devices to the same values.

Master Timebase synchronization results in skew due to the time required for the clock to travel between devices.

Most devices use a Reference Clock instead of a Master Timebase, thus they use Reference Clock synchronization. Master Timebase synchronization also requires you to share multiple signals and reserve multiple RTSI or PXI trigger lines for those signals. For Sample Clock-timed applications where all devices run at the same rate, you can use Sample Clock synchronization to eliminate the need for a shared Start Trigger, thus the additional RTSI/PXI line. You can also use Sample Clock synchronization to synchronize devices that use a Master Timebase with devices that use a Reference Clock.

Related concepts:

- Synchronization
- Skew
- Reference Clock Synchronization
- Sample Clock Synchronization
- Master and Slave Devices

Sample Clock Timebase Synchronization

Sample Clock Timebase synchronization allows you to synchronize devices at different rates. Use Sample Clock Timebase synchronization to synchronize a combination of devices for which Reference Clock, Master Timebase, or Sample Clock synchronization is not an option.

When using Sample Clock Timebase synchronization, slave devices replace the Sample Clock Timebase for a given subsystem with the Sample Clock Timebase from another device. Each device derives its Sample Clock from the synchronized Sample Clock Timebase. You must share a Start Trigger for the derived clocks to start in phase.

> ✎ **Note** You must start any tasks on slave devices before starting the task on the master device.

You must set two properties on the slave devices to synchronize the Sample Clock Timebase. Use the SampClk.Timebase.Src attribute/property to specify the terminal of the Sample Clock Timebase from the master device. Because the slave devices then use an external timebase, you must also use the SampClk.Timebase.Rate property to specify the rate of that timebase. Instead of hard-coding those values, you can query the Sample Clock Timebase rate and source from the master device and set the rate and source on the slave devices to the same values.

Sample Clock Timebase synchronization results in skew due to the time required for the clock to travel between devices.

Related concepts:

- Skew
- Master and Slave Devices

Mixed-Clock Synchronization

For some applications, the other synchronization methods might not be sufficient. In such cases, you might be able to synchronize by using a clock from one device as a different clock on another device. For example, to synchronize devices that use a Reference Clock with devices that use a Master Timebase, such as to synchronize X Series devices with E Series devices, you can perform Reference Clock synchronization, but instead of locking the X Series reference clock to a PXI backplane clock, lock it to the Master Timebase of the E Series device.

Mixing clocks in this manner is often much more complicated than using the same clocks. You might have to manually configure several other timing parameters, such as delays, active edges, or clock divisors. Use mixed-clock synchronization only as a last resort if the other synchronization methods are not an option.

Related concepts:

- Methods of Synchronization

Counter Synchronization

You cannot synchronize counter input applications using implicit timing in the same sense as analog input or output applications. These types of counter input applications cannot be programmed to make their measurements at the same time because the signals being measured themselves determine when the measurements are made, and there is no reason to set up multiple devices to measure the same signal. You also cannot use Start Triggers for these applications.

You can, however, ensure that all counters are using the same timebase for their input measurements by sharing the CI Counter Timebase signal. Program all devices to use the same signal (usually the 20MHzTimebase from one of the devices) as their CI Counter Timebase. More generally, one device can be queried for its CI Counter Timebase source and that terminal can be set as the source of the CI Counter Timebase for the other devices.

To synchronize Sample Clock-timed buffered counter input applications, use Sample Clock synchronization. The Sample Clock must be externally supplied to one of your devices. The other synchronized devices are programmed to use as their Sample Clock the CtrnGate signal, where n is the number of the counter.

To synchronize pulse generation counter output applications, share the CO Counter Timebase and Start Trigger signal. Program all devices to use the same signal (usually the 20MHzTimebase from one of the devices) as their CO Counter Timebase. More generally, one device can be queried for its CO Counter Timebase source and that terminal can be set as the source of the CO Counter Timebase for the other devices. Program all devices to use the same signal as their Digital Edge Start Trigger. This is typically the CtrnGate signal from one of the devices, where n is the number of the counter.

Related concepts:

- Sample Clock Synchronization

Trigger Skew Correction

When sharing a trigger across multiple devices, the master device must respond to and export the trigger, and the trigger signal must travel from the master device to slave

devices. This results in skew in the trigger signal. On some devices using Reference Clock synchronization, you can compensate for that skew by locking the trigger to the Reference Clock or to a clock derived from the Reference Clock. When you lock triggers to a clock, the device responds to those triggers on a subsequent edge of that clock, rather than immediately. Therefore, skew correction results in increased latency.

To enable trigger skew correction for an application, specify which device is the master and which devices are slaves using the SyncType DAQmx Trigger property.

Related concepts:

- Reference Clock Synchronization

Subsystem

A **subsystem** is the circuitry (ADCs, DACs, clocks, triggers, timing signals, timing engines, and so on) that a device uses to acquire or generate samples. On DAQ devices, there are separate subsystems for analog input, analog output, each counter, and digital I/O. A subsystem is not synonymous with I/O type, however. For instance, each counter's input and output circuitry make up one subsystem. The trigger bus can also function as a subsystem.

Timing Engines

A timing engine is the circuitry a device uses to control an acquisition or generation. Controlling an acquisition or generation includes:

- Utilizing timing signals to control when the device acquires or generates each sample.
- Determining when to start and stop the acquisition or generation, using triggers and other control signals.
- Producing clocks, triggers, and other control signals.
- Routing those signals to other devices or other parts of the same device.

Timing engines for different devices and subsystems provide different functionality. The timing engine for an analog input or output subsystem uses a timebase to produce a Sample Clock. The capabilities of a digital I/O timing engine depend on the

device. Some can use change detection and/or handshaking for sample timing, while others can also produce a Sample Clock. Some timing engines cannot produce a Sample Clock, but can perform Sample Clock timing if the clock comes from an external source, such as from another subsystem or from a source external to the device. For example, counters produce no inherent timing signals, but can utilize a Sample Clock from an external source. Some devices can perform Sample Clock timing for digital I/O, but require an external clock.

All timing engines can respond to and route control signals, such as triggers. Not all triggers are supported by all devices and measurement types.

Most devices have a single timing engine per subsystem. When a task reserves resources from a timing engine, another task cannot reserve those same resources. However, another task can use signals routed by that timing engine. For example, a counter task can use the Sample Clock from an analog input task. One task can use the Start Trigger from another task to synchronize when the tasks start. Reserving one part of a timing engine reserves the entire timing engine. For example, if an analog input task uses the Sample Clock from an analog output task, the analog input task must reserve the analog input timing engine to utilize that external Sample Clock. As a result, another task can not use the Sample Clock from the analog input timing engine.

> **Note** On-demand analog input tasks must reserve resources from a timing engine.

Some devices have multiple timing engines available for some subsystems. Multiple timing engines allow a device to run multiple tasks simultaneously on the same subsystem or to use different terminals for handshaking.

Related concepts:

- Multiple Timing Engines
- Simultaneous Tasks
- Handshaking Line Configuration

Related reference:

- Sample Clock Timing for Digital I/O

Events

Triggers and clocks are input signals. Exportable triggers and clocks, such as the sample clock, also can be output signals. Output signals that do not have a trigger or clock counterpart are called events. Events are emitted to signify a device state change, the arrival of a certain kind of sample, the production of a certain amount of samples, or the passage of time.

NI-DAQmx includes the following events:

- **Advance Complete Event**—A signal emitted by a switch when it has finished executing an instruction in its scan list.
- **AI Hold Complete Event**—A signal emitted by a multiplexed analog input circuit when the analog signal at the physical channel being measured has been latched or held. The AI Hold Complete Event is designed to signal an external multiplexer to switch to the next channel. This signal was previously known as SCANCLK, which is the legacy name of the external terminal where this signal can be emitted.
- **Change Detection Event**—A signal a DIO device generates after it detects a change- a rising edge, a falling edge, or both rising and falling edges-on the data lines.
- **Counter Output Event**—A signal produced by a counter when it reaches terminal count.
- **Handshake Event**—A signal generated by a DAQ device that is used for handshaking. The assertion and deassertion times for this event are configurable within a handshaking cycle for some devices. For these devices, the default configuration is to mimic the 8255 protocol, which means that for input tasks, this event asserts after the device has space available in its FIFO; for output tasks, it asserts after valid data has been driven on the data lines; and in both input and output tasks, the event deasserts after the Handshake Trigger has been asserted.
- **Ready For Start Event**—A signal produced when a device is ready to accept a Start Trigger.
- **Ready For Transfer Event**—A signal sent to the peripheral device. The signal indicates that the DAQ device is ready for a transfer. For burst handshake output tasks, this means that the data is on the data lines. For input tasks, this means that there is space available in the device FIFO. This event is used by devices that support burst handshake timing.
- **Sample Complete Event**—A signal produced when the device acquires a sample from every channel in a task.

- **Watchdog Timer Expired Event**—A signal produced when a watchdog timer expires. Watchdog timers are hardware features that can detect failures in the software controlling the device.

> ✎ **Note** The Sample Complete Event is not exportable.

> ✎ **Note** Some event signals might be missed when you run an application. This typically happens with high-speed event sources. NI-DAQmx does not generate an error if this occurs.

Related concepts:

- Handshake Trigger
- Start Trigger
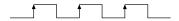
Related reference:

- Burst Handshake Timing for Digital I/O

Exported Signal Behaviors

You can export clocks, triggers, and events. The exported signal can exhibit one of three behaviors. It can rapidly change from its current state and then back again (pulse), change from its current state and remain at that state (toggle), or change from its current state and remain at that new state for a period of time determined by the configuration of the task before reverting back to the initial state (level). Most exported signals have pulse behavior, but some signals have programmable output behavior. For example, the Counter Output Event supports toggle as well as pulse behaviors. The Sample Clock supports pulse and level behaviors. You specify the behavior through the Output Behavior attribute/property for the exported signal.

Most exported signals exhibit the pulse behavior. When the event occurs, a finite pulse is generated. The pulse width of some exported triggers and events is configurable. The polarity of a signal exported as a pulse is also sometimes configurable. In the following illustration, the polarity is set to active high, meaning the initial state change of the signal is from low to high. When an event is exported as a pulse, each time the event occurs, the exported signal pulses.

When an event is exported as a toggle, each time the event occurs the exported signal changes state just once and remains at its new state until the next occurrence of the event. You can also set the initial state. In the following illustration, the initial state is set to high. The Counter Output Event is an example of a signal that can toggle.

For level behavior, the signal changes state and remains at that state for a period of time that is dependent on some configurable aspect of your task. If you are exporting the Sample Clock, the exported signal goes high at the beginning of the sample and goes low when the last AI Convert Clock pulse begins, as shown in the illustration.

Exported Sample Clock

Convert Clock

> **Note** On some devices, the exported signal can go low at the beginning of the sample and then high when the last AI Convert Clock pulse begins. Refer to your device documentation for additional information.

Software Events

Software events provide an asynchronous notification mechanism for a set of DAQ events. Unlike hardware events, software events do not require you to use a thread to wait until data is available. Using event-based programming, you can write an application that continues to perform work while waiting for data without resorting to developing a multi-threaded application.

NI-DAQmx includes the following software events:

- **Every N Samples Acquired Into Buffer Event**—Occurs when the user-defined number of samples is written from the device to the PC buffer. This event works only with devices that support buffered tasks.

> **Note** Some devices—such as AO Series, E Series, and M Series devices—require that the sample interval divide evenly into the buffer size when using DMA as your data transfer mechanism. For instance, if the buffer size is 1,000 samples, specifying 102 for this software event generates an error. Specifying 100, however, would not generate an error. If you are using IRQ as the data transfer method, the value does not need to be evenly divisible. With IRQ, however, the Data Transfer Request Condition attribute/property can affect when this software event occurs.

- **Every N Samples Transferred From Buffer Event**—Occurs when the user-defined number of samples is written from the PC buffer to the device. This event works only with devices that support buffered tasks.

> **Note** Some devices—such as AO Series, E Series, and M Series devices—require that the sample interval divide evenly into the buffer size when using DMA as your data transfer mechanism. For instance, if the buffer size is 1,000 samples, specifying 102 for this software event generates an error. Specifying 100, however, would not generate an error. If you are using IRQ as the data transfer method, the value does not need to be evenly divisible. With IRQ, however, the Data Transfer Request Condition attribute/property can affect when this software event occurs.

- **Done Event**—Occurs when the task completes execution or when an error causes the task to finish. Recoverable errors that do not cause the task to finish do not cause this event to fire. Calling the Stop Task function/VI to complete execution similarly does not cause this event to fire.
- **Signal Event**—Occurs when the specified hardware signal occurs. Supported signals include the counter output event, change detection event, sample complete event, and the sample clock.

Related concepts:

- Events
- Buffering
- How Is Buffer Size Determined?
- Data Transfer Mechanisms
- When Is A Task Done?
- Sample Clock

Reading and Writing Data

This section covers buffering and selecting data formats and organization.

Related concepts:

- Buffering
- Selecting Read and Write Data Format and Organization

Selecting Read and Write Data Format and Organization

NI-DAQmx provides multiple VIs and functions for reading and writing data. In many cases, you can use multiple options. This section outlines the options and provides some guidelines to follow to select the best option. Some data formats and organizations are not supported in all ADEs.

The read and write VIs have two major selection criteria: data format and data organization. Data format deals with the type of the data that is returned. For example, counter reads can return integers or floating-point formats. The second category, data organization, deals with the structure the data is returned in. For example, analog reads have a variety of array and scalar organizations.

Related concepts:

- Data Formats in NI-DAQmx
- Data Organization

Data Formats in NI-DAQmx

Data format deals with the type of the data that is read or written.

Related concepts:

- Waveform Timing Limitations
- Digital Data (Integer Format)
- Raw Data

Analog and Power Channel Data Formats

- **Waveform** —The waveform data format includes the channel name, timing, and unit information with the actual 64-bit scaled floating-point data. Your ADE provides a mechanism for extracting and setting individual parts of the waveform.

  For input tasks, you can use the additional information for a variety of purposes. For example, you can update graphs to show the timing information and include labels with the channel names. Analysis routines can use the timing information for calculations such as FFTs. Because there is overhead associated with including this additional information, NI-DAQmx allows you to configure the information you want to include.

  For output tasks, the timing information is the primary field that is useful. A waveform generated by a library may include timing information that you can use to set up the timing for your output task.

  When reading data, the waveform data includes the time when the first sample in the waveform was acquired, t0, and the amount of time that elapsed between each sample, dt. However, there are limitations on these two values.

- **64-Bit Floating-Point Numbers**—The 64-bit floating-point number format allows you to read or write scaled data with no additional information. Use this format to work with scaled data that requires higher performance than the waveform format provides. You might also use this format because it is a better match for the libraries you plan to use.
- **Unsigned and Signed Integers** —The unsigned and signed integer format reads or writes data in the native format of the device. Use this format for maximum performance. The tradeoff is that your application has to understand how to interpret and manipulate data that is not in engineering units.

Digital Channel Data Formats

- **Waveform**—The waveform data format includes the channel name and timing information with the actual data represented in a dedicated digital format. Your ADE provides a mechanism for extracting and setting individual parts of the waveform.

  The dedicated digital format represents digital data similar to logic analyzers and

digital simulation tools. Each channel has no limits on the number of lines. In addition, the digital format allows for additional states beyond basic 1s and 0s. The ADE can take advantage of this format by tailoring data and graph displays for the digital data.

For input tasks, you can use the additional information for a variety of purposes. For example, you can update graphs to show the timing information and include labels with the channel names. Because there is overhead associated with including this additional information, NI-DAQmx allows you to configure the information you want to include.

For output tasks, the timing information is the primary field that is useful. A waveform generated by a library may include timing information that you can use to set up the timing for your output task.

When reading data, the waveform data includes the time when the first sample in the waveform was acquired, t0, and the amount of time that elapsed between each sample, dt. However, there are limitations on these two values.

- **Line Format (Boolean)**—The line format represents each line within a channel as a single Boolean value (a single byte). The states of the data are limited to 1s (true) and 0s (false). Line formats are only provided for single sample reads and writes.

  Use the line format when it is convenient for manipulating or displaying the digital data. A typical application is controlling or reading back relay states. For high-speed digital applications, you should generally not use the line format.

- **Port Format (Integer)**—The port format matches the native format of digital devices that can represent only two digital states and organize individual lines into collections known as ports. For more information, refer to Digital Data-Integer Format.

  The port format is the most efficient in terms of space, as it requires only a bit of memory per line. In addition, the port format is often the most efficient in time as it matches the native format of many devices.

  The largest integer supported is 32 bits; therefore, you can read and write digital channels with no more than 32 lines when using the port format.

Counter Channel Data Formats

- **64-Bit Floating-Point Numbers**—The 64-bit floating-point number format reads scaled data. This format is best when you want to work with data in engineering units.
- **Unsigned Integers**—The unsigned integer format reads data in the native format of the device. Use this format for maximum performance. The tradeoff is that your application will have to understand how to interpret and manipulate data that is not in engineering units.

Raw Data Formats

The raw data format is defined by the native data format of the device.

Data Organization

The number of channels and the number of samples being read generally affect data organization. For example, if 100 samples are read for eight analog channels using 64-bit floating-point format, a two-dimensional array is used with one index selecting the channel and the second index selecting the sample. On the other hand, a simple floating-point scalar value is sufficient to read one sample for one analog channel using the 64-bit floating-point format. In general, the data organization for a particular read or write call is the simplest reasonable format that can handle the number of channels and samples requested.

There are often multiple legal data organizations to choose from. The main tradeoff to consider for data organization is difficulty in manipulation of the data. You can use data organizations that can handle multiple channels and multiple samples, but they are generally the most complicated to manipulate.

Performance is similar for equivalent operations used to read or write data with different data organizations.

- **Waveform Data Organization**—A waveform can contain one or more samples.
- **1D Waveform Array Data Organization**—The single dimension of a waveform array selects the channel. Each waveform can contain multiple samples, so a second dimension is not required.
- **Scalar Data Organization**—Use scalars when you read or write a single sample on a single channel. Scalar data is easy to manipulate. It is a good match when data is

read and/or written to individual channels as needed.

Scalar data is generally not a good match for high-speed multiple sample applications.

Scalar data also is not a good choice if multiple channels are acquired or generated simultaneously. Using a multiple channel organization is easier and in the case of output operations is actually a requirement.

- **Array Data Organization**—Array formats allow you to read or write data for multiple channels and/or multiple samples at the same time. If you acquire or generate on multiple channels simultaneously, reading and writing them at the same time is easiest. Reading and writing multiple samples in one call is more efficient than reading and writing samples one at a time.
- **Raw Data Organization**—The raw data organization is defined by the native organization of the device.

Related concepts:

- [Raw Data](#)

Digital Data (Integer Format)

You typically use an integer format to read or write entire ports. In integer format, each digital channel you read or write must fit into one integer. For example, if the largest channel in a task consists of one 8-line port, you can use the 8-bit, 16-bit, or 32-bit format. If you have more than one 8-line port or a port with more than eight lines in any channel within a task, you must use the 32-bit format.

Each byte in the integer maps to a port in the channel, in the order in which you added the ports to the channel. The least significant byte maps to the first port added to the channel, with all unused bytes zeroed out. Therefore, if a channel contains two 8-line ports, `port0` and `port1`, and you added `port0` to the channel before `port1`, the channel uses a 32-bit representation:

unused	unused	port1	port0

Within a particular byte, each bit in the integer maps to a line in the corresponding

port. NI-DAQmx orders the bits by line number, with the least significant bit mapping to the lowest line number. Therefore, with these values assigned to the lines in the channel, you might get the following:

port0/line0	0
port0/line1	0
port0/line2	1
port0/line3	0
port0/line4	1
port0/line5	1
port0/line6	0
port0/line7	1
port1/line0	1
port1/line1	1
port1/line2	1
port1/line3	0
port1/line4	1
port1/line5	0
port1/line6	0
port1/line7	1

The 32-bit binary representation of the channel is the following:

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 1 1 0 1 1 0 1 0 0
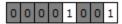
with an integer value of 38836.

If you specify only certain lines in a port to read or write, the full length of the integer is still used, but all unused bits are zeroed out. Therefore, the following lines and values:

port0/line0	1

`port0/line3`	1

yield the following 8-bit representation:

0	0	0	0	1	0	0	1

with an integer value of 9.

Interleaving

Interleaved samples prioritize samples before channels, such that the array lists the first sample from every channel in the task, then the second sample from every channel, up to the last sample from every channel.

Channel 0—Sample 1
Channel 1—Sample 1
Channel 2—Sample 1
Channel 0—Sample 2
Channel 1—Sample 2
Channel 2—Sample 2
...
Channel 0—Sample N
Channel 1—Sample N
Channel 2—Sample N

Non-interleaved samples prioritize channels before samples, such that the array lists all samples from the first channel in the task, then all samples from the second channel, up to all samples from the last channel.

Channel 0—Sample 1
Channel 0—Sample 2
...
Channel 0—Sample N
Channel 1—Sample 1
Channel 1—Sample 2
...
Channel 1—Sample N
Channel 2—Sample 1

Channel 2—Sample 2
...
Channel 2—Sample N

Raw Data

Raw data is in the native format and organization of the device, read directly from the device or buffer without scaling or reordering. The native format of a device can be an 8-, 16-, or 32-bit integer, signed or unsigned.

If you use a different integer size than the native format of the device, one integer can contain multiple samples or one sample can stretch across multiple integers. For example, if you use 32-bit integers, but the device uses 8-bit samples, one integer contains up to four samples. If you use 8-bit integers, but the device uses 16-bit samples, a sample might require two integers. This behavior varies from device to device. Refer to your device documentation for more information.

NI-DAQmx does not separate raw data into channels. It returns data in an interleaved or non-interleaved 1D array, depending on the raw ordering of the device. Refer to your device documentation for more information.

> **Note** If your device supports software calibration, NI-DAQmx does not calibrate raw samples. Refer to calibration to find out if your device uses software or hardware calibration.

Related concepts:

- Interleaving
- Device Calibration Considerations

Unscaled Data

Unscaled data is in the native format of the device, read directly from the device or buffer without scaling. The native format of a device can be an 8-, 16-, or 32-bit integer, signed or unsigned.

> ✏️ **Note** If your device supports software calibration, NI-DAQmx does not calibrate unscaled samples. Refer to calibration to find out if your device uses software or hardware calibration.

Related concepts:

- [Device Calibration Considerations](#)

Waveform Timing Limitations

The limitation on t0 is that NI-DAQmx calculates the starting time for the task when data is read the first time. At this time, NI-DAQmx calculates the starting time for the task by reading the current system time and subtracting the number of samples acquired × dt from the system. Therefore, if you call read after the acquisition is complete, the calculated start time for the task is not accurate. This inaccuracy is reflected in the t0 returned with the waveform data.

The limitation on dt is that for certain timing types, NI-DAQmx cannot calculate the value of dt. When you use sample clock timing, NI-DAQmx calculates dt based on the rate of the clock. Because NI-DAQmx does not know the rate when handshake, implicit, on demand, or change detection timing is specified, NI-DAQmx returns dt as 0. Waveforms with a dt of 0 often do not work with the waveform analysis functions. However, you can always update the value of dt in your application if you know the expected rate of the timing source. Your ADE has an interface to update the value of dt.

> ✏️ **Note** The waveform data only supports symmetric timing between samples. If your timing is not symmetric such as if each sample has a time stamp, the waveform data format cannot contain the timing information. However, you can use your ADE's analysis library to resample the data using a constant dt. You can the use the resampled data with the waveform based analysis library.

Buffering

A buffer is a temporary storage in computer memory for acquired or to-be-generated

samples. Typically this storage is allocated from your computer's memory and is also called the task buffer. For input operations, a data transfer mechanism transfers samples from your device into the buffer where they wait for a call to the Read function/VI to copy the samples to your application. For output operations, the Write function/VI copies samples into the buffer where they wait for the data transfer mechanism to transfer them to your device.

When Is a Buffer Created?

If you use the Timing function/VI and set the `sample mode` to finite or continuous, NI-DAQmx creates a buffer. If you set `sample mode` to Hardware Timed Single Point, NI-DAQmx does not create a buffer.

If you set the Data Transfer Mechanism to Programmed I/O or set the buffer size to zero by using either the Input or Output Buffer Config function/VIs, NI-DAQmx does not create a buffer (even if you also used the Timing function/VI). A data transfer mechanism of programmed I/O means there is no buffer.

Related concepts:

- Data Transfer Mechanisms
- How Is Buffer Size Determined?
- Reference Triggering Impact on Buffers
- Continuous Acquisition and Generation with Finite Buffer Size
- Controlling Where in the Buffer to Read Samples
- Read Status Attributes/Properties and Buffers
- Controlling Where in the Buffer to Write Samples
- Write Status Attributes/Properties and Buffers

How Is Buffer Size Determined?

Input Tasks

If your acquisition is finite (`sample mode` on the Timing function/VI set to Finite Samples), NI-DAQmx allocates a buffer equal in size to the value of the samples per channel attribute/property. For example, if you specify samples per channel of 1,000 samples and your application uses two channels, the buffer size would be 2,000 samples. Thus, the buffer is exactly big enough to hold all the samples you want to

acquire. If the acquisition is continuous (`sample mode` on the Timing function/VI set to Continuous Samples), NI-DAQmx allocates a buffer equal in size to the value of the samples per channel attribute/property, unless that value is less than the value listed in the following table. If the value of the samples per channel attribute/property is less than the value in the table, NI-DAQmx uses the value in the table.

Sample Rate	Buffer Size
No rate specified	10 kS
0-100 S/s	1 kS
101-10,000 S/s	10 kS
10,001-1,000,000 S/s	100 kS
>1,000,000 S/s	1 MS

Note For performance reasons, the default buffer size for continuous acquisitions differs slightly when logging is enabled.

You can override the default buffer size by calling the Input Buffer Config function/VI.

NI-DAQmx does not create a buffer when the `sample mode` on the Timing function/VI is set to Hardware Timed Single Point.

Note Using very large buffers may result in diminished system performance due to excessive reading and writing between memory and the hard disk. Reducing the size of the buffer or adding more memory to the system can reduce the severity of these problems.

Output Tasks

For generations, the amount of data you write before starting a generation determines the size of the buffer. The first call to a Multiple Samples version of the Write function/VI creates a buffer and determines its size.
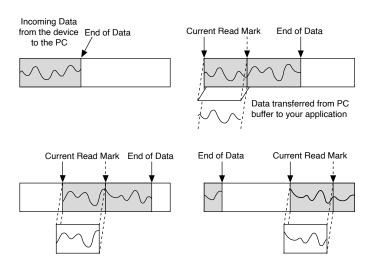
You also can use the Output Buffer Config function/VI to create an output buffer. If you use this function/VI, you must use it before writing any data.

The samples per channel attribute/property on the Timing function/VI does not determine the buffer size for output. Instead it is the total number of samples to generate. If n is your buffer size, setting samples per channel to 3×n generates the data in the buffer exactly three times. To generate the data exactly once, set samples per channel to n.

NI-DAQmx does not create a buffer when the `sample mode` on the Timing function/VI is set to Hardware Timed Single Point.

Continuous Acquisition and Generation with Finite Buffer Size

The NI-DAQmx API uses circular buffers as shown in the following figure. For input operations, portions of data are read from the buffer while the buffer is filled. Likewise for output operations, portions of the buffer can be written to while the buffer is emptied. Using a circular buffer, you can set up your device to continuously acquire data in the background while NI-DAQmx retrieves the acquired data.

When a continuous operation reaches the end of the buffer, it returns to the beginning and fills up (or in the case of output operations, reads from) the same buffer again. Your input application must retrieve data in blocks, from one location in the buffer, while the data enters the circular buffer at a different location, so newer data does not overwrite unread data.

While a circular buffer works well in many applications, two possible problems can occur with this type of acquisition: Your application might try to retrieve data from the

buffer faster than data is placed into it, or your application might not retrieve data from the buffer before NI-DAQmx overwrites the data into the buffer. When your application tries to read data from the buffer that has not yet been collected, NI-DAQmx waits for the data to be acquired and then returns the data. If your application does not read the data from the circular buffer fast enough, you receive an error, stating that some data has been overwritten and lost. If losing data in this way is not important to you, change the setting of the OverWrite Mode attribute/property.

Reference Triggering Impact on Buffers

Even though you have set the `sample mode` parameter on the Timing function/VI to Finite Samples, the acquisition runs continuously until the Reference Trigger occurs. The number of posttrigger samples in your buffer after the acquisition has finished is equal to the value of the `samples per channel` parameter from the Timing function/VI minus the number of pretrigger samples from the Trigger function/VI. When using a Reference Trigger, the default read position is Relative To First Pretrigger Sample with a read Offset of 0.

Controlling Where in the Buffer to Read Samples

Default read behavior depends on if a Reference Trigger is configured. If there is no Reference Trigger, NI-DAQmx reads samples beginning with the first sample acquired with each subsequent read beginning where the previous one left off. If there is a Reference Trigger, NI-DAQmx reads samples beginning with the first pretrigger sample and cannot begin reading until the acquisition has finished. This default behavior can be changed by using the Relative To and Offset attributes/properties.

The place where a read begins is called the Current Read Position. Each time data is read, the Current Read Position is computed based on the settings of the Relative To and Offset attributes/properties. When there is no Reference Trigger, the default for Relative To is Current Read Position. When there is a Reference Trigger, the default for Relative To is First Pretrigger Sample. In either case, the default for Offset is 0. Changing the settings of these two attributes/properties controls where in the buffer data is read.

During a continuous acquisition, for example, you can always read the most recent 1000 points by setting Relative To to Most Recent Sample and Offset to -1000. Even

when a Reference Trigger is configured, you can begin reading samples immediately by setting Relative To to First Sample.

Read Status Attributes/Properties and Buffers

The three Read Status attributes/properties are useful for observing the progress of your acquisition.

- The Current Read Position is the place in the buffer where the next read begins if the Relative To attribute/property is Current Read Position and the Offset is 0. In any case, the Current Read Position is always where the last read left it.
- Total Samples per Channel Acquired is the total number of samples per channel acquired by the device and transferred into the buffer.
- Available Samples per Channel is computed by first calculating the Current Read Position based on the settings of the Relative To and Offset attributes/properties and then subtracting this number from Total Samples per Channel Acquired.

Controlling Where in the Buffer to Write Samples

By default, NI-DAQmx writes samples sequentially beginning with the first sample in the buffer, and each write begins where the previous one left off. The sample where a write begins is called the Current Write Position. Each time data is written, the Current Write Position is computed based on the settings of the Relative To and Offset attributes/properties. The default write behavior results from the default settings of these two attributes/properties. The default for Relative To is Current Write Position and the default for Offset is 0. Changing the settings of these two attributes/properties controls where in the buffer data is written.

Write Status Attributes/Properties and Buffers

The three Write Status attributes/properties are useful for observing the progress of your generation.

- The Current Write Position is the place in the buffer where the next write begins if the Relative To attribute/property is Current Write Position and the Offset is 0. In any case, the Current Write Position is always where the last write left it.
- Total Samples per Channel Generated is the total number of samples per channel

generated by your device since the task started.

- Space Available in Buffer is computed by first calculating the Current Write Position based on the settings of the Relative To and Offset attributes/properties and then subtracting this number from the sum of Total Samples per Channel Generated and the buffer size. If regeneration is allowed, the Space Available in Buffer value is capped at the buffer size and grows from 0 to the buffer size repeatedly.

Glitching
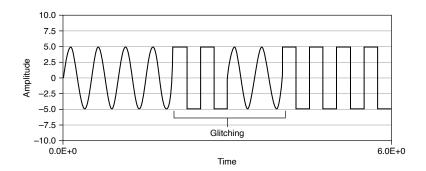
Glitching refers to the generation of a waveform in which, when transitioning from old samples in the buffer to new samples, a mixture of old and new samples is generated rather than just the new samples. This situation may occur when continuously generating samples if the Regeneration Mode write attribute/property is set to Allow Regeneration. Glitching occurs when, while you write new samples, a subset of these new samples are generated and then, since you have not finished writing all of the new samples, a subset of the old samples is generated. After your write operation completes, only the new samples are generated.

NI-DAQmx reduces the likelihood of glitching by ensuring that the writing of new samples does not overtake the generation. This glitching protection works by pausing the write until the total samples generated is more than one buffer ahead of the current write position. However, NI-DAQmx does not ensure that the generation does not overtake the new samples being written. If this occurs, a glitch results, and NI-DAQmx reports the kWarningPotentialGlitchDuringWrite warning (error 200015). The following suggestions can help you to avoid generating glitches:
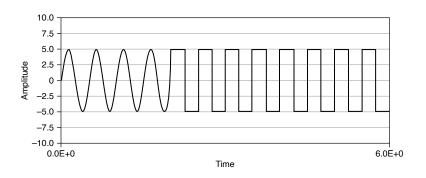
- Write new samples that are almost one buffer ahead of the total samples generated. By writing the new samples almost one buffer ahead of the total samples generated, there is less of a chance that the generation overtakes the new samples that are being written. If you are updating the entire buffer at a time, wait to write the new samples until the total samples generated attribute/property is one sample greater than an integral number of buffer sizes. For example, if the buffer size is 1000 samples, wait to write new samples until the total samples generated is either 1001, 2001, 3001, and so on.
- Increase the buffer size. If the buffer size is larger, there is less of a chance that the generation overtakes the new samples that are being written.
- Decrease the sample clock rate. If the sample clock rate is slower, there is less of a

chance that the generation overtakes the new samples that are being written.

In the following graphs, the sine wave is generated from old samples and the square wave is generated from the new samples. The first graph depicts glitching.

The second graph depicts the same waveforms without glitching.

Data Transfer Mechanisms

There are four primary ways to transfer data across the PCI bus: Direct Memory Access (DMA), Interrupt Request (IRQ), Programmed I/O, and USB Bulk.

Direct Memory Access (DMA)

DMA is a mechanism to transfer data between the device and computer memory without the involvement of the CPU. This mechanism makes DMA the fastest available data transfer mechanism. NI uses DMA hardware and software technology to achieve high throughput rates and to increase system utilization. DMA is the default method of data transfer for DAQ devices that support it.

> **Note** DAQCard and USB devices do not support DMA .

Interrupt Request (IRQ)

IRQ transfers rely on the CPU to service data transfer requests. The device notifies the CPU when it is ready to transfer data. The data transfer speed is tightly coupled to the rate at which the CPU can service the interrupt requests. If you are using interrupts to acquire data at a rate faster than the rate the CPU can service the interrupts, your systems may start to freeze.

Programmed I/O

Programmed I/O is a data transfer mechanism in which a buffer is not used and instead the computer reads and writes directly to the device. Software-timed (on-demand) operations typically use programmed I/O.

USB Bulk

USB Bulk is a buffered, message-based streaming mechanism for data transfer. This high-speed method is the default transfer mechanism for USB devices.

Memory Mapping

Memory mapping is a technique for reading and writing to a device directly from your program, which avoids the overhead of delegating the reads and writes to kernel-level software. Delegation to the kernel is safer, but slower. Memory mapping is less safe because an entire 4 KB page of memory must be exposed to your program for this to work, but it is faster. Memory mapping is set by default if your device supports it.

Changing Data Transfer Mechanisms between DMA and IRQ

There are a limited number of DMA channels per device (refer to your device documentation). Each operation (AI, AO, and so on) that requires a DMA channel uses that mechanism until all of the DMA channels are used. After all of the DMA channels are used, you receive an error if you try to run another operation requesting a DMA channel. If appropriate, you can change one of the operations to use interrupts. For NI-DAQmx, use the Data Transfer Mechanism channel attribute/property.

Regeneration

Generating the same data more than once is called regeneration. You can configure NI-DAQmx to allow or disallow regeneration by setting the Regeneration Mode attribute/property. By default, NI-DAQmx allows regeneration for sample clock timing and disallows it for handshaking or burst handshaking timing. When regeneration is disallowed, new data must be continuously written to the device.

Allowing Regeneration and Using Onboard Memory

When the Use Only Onboard Memory attribute/property is true, NI-DAQmx transfers data only once to the device and that data is continually regenerated from there. Attempting to write new data to the device after starting the task returns an error. In addition, the amount of data written to the device before starting the task must fit in the onboard memory of the device.

When the Use Only Onboard Memory attribute/property is false, NI-DAQmx continuously transfers data from the host memory buffer to the device even though this data is not changing. Thus, if you write new data to the device after starting the task, that new data is generated and regenerated until you write more new data. This type of regeneration is sometimes called PC memory or user buffer regeneration.

When this attribute/property is false, you can also set the Data Transfer Request Condition attribute/property to specify when to transfer data from the host buffer to the device.

TDMS Logging

Technical Data Management Streaming (TDMS) is a binary file format that allows for high-speed data logging. When you enable TDMS data logging, NI-DAQmx can stream data directly from the device buffer to the hard disk. NI-DAQmx improves performance and reduces the disk footprint by writing raw data to the TDMS file, including the scaling information separately for use when reading back the TDMS file. You can also read data while logging to disk.

For optimal performance, follow these tips:

- Use a PCI Express or PXI Express device with a RAID array.
- Log data only. For very high-speed acquisitions, logging and reading data can slow performance.
- If logging and reading data, ensure the number of samples per channel to read is evenly divisible by the sector size of the hard disk.
- If manually configuring the buffer size, choose a multiple of eight times the sector size of the hard disk. For instance, if your sector size is 512 bytes, your buffer size might be 4,096 samples.

NI provides a number of mechanisms for reading TDMS files, including software support in LabVIEW, LabWindows/CVI, ANSI C, DIAdem, and Measurement Studio. In addition, NI provides a Microsoft Excel plug in. Refer to ni.com/tdms for additional information. For information about high-speed data streaming, refer to ni.com/streaming.

Related information:

- The NI TDMS File Format
- Streaming Architecture of the Industry's Highest Performance PXI Express Platform

Logging Across Multiple Files

You can split TDMS logs across multiple files by using either the DAQmx Start New File function/VI or the Logging.SampsPerFile attribute/property.

Start New File

The DAQmx Start New File function/VI starts a new TDMS file with the specified file name the next time data is written to disk. If the logging mode is set to Log, data is written to disk on multiples of the Logging.FileWriteSize attribute/property. For example, if Logging.FileWriteSize is set to 150 when you use this function/VI, a new TDMS file is created at the next 150 sample interval. If the `logging mode` is set to Log and Read, data is written to disk when you call the DAQmx Read function/VI.

Samples Per File

The Logging.SampsPerFile attribute/property specifies the number of samples to log to a TDMS file before creating a new file. New files are named with the convention of

`<filename>_####.tdms,` where `####` starts at 0001 and increments automatically with each new file.

You can set the Logging.FilePath attribute/property while the task is running to change the file path. The change takes place with the next file created. If you change the log name, the numbering of the files resets to 0001. You can also change the directory to save log files to by specifying a name ending in a back slash (for example, `D:\`). If you specify a directory, the log file name remains the same and file numbering continues off of the old directory. For example, if you change the directory from `C:\` to `D:\` while logging to `testlog_0003.tdms`, `testlog_0003.tdms` is written to `C:\`, and `testlog_0004.tdms` is written to `D:\`.

> **Note** If the Logging.LoggingMode attribute/property is set to Log Only, Logging.SampsPerFile must be divisible by the Logging.FileWriteSize attribute/property, which is based, by default, on the buffer size.

To change a file immediately, you can also use the DAQmx Start New File function/VI while the Logging.SampsPerFile attribute/property is active.

On-Demand Logging

For on-demand logging, NI-DAQmx creates a Time channel in the TDMS file that contains timestamp data. The Time channel can be omitted with the WfmAttr Read attribute/property, which specifies the data to return. To disable the Time channel, set WfmAttr to Samples.

Pausing Logging

You can pause or resume logging by setting the Logging.Pause attribute/property. This can be used if you do not want to initially log data but want to begin when a certain condition is met. You can also use this attribute/property to temporarily disable logging.

If you re-enable logging after disabling logging in an application, a new group is created in the log file. This group is denoted by the name `<group name> #1,` where the number is automatically incremented.

Before you resume logging data in Log and Read mode, you can set the Offset attribute/property to log data that was previously read while logging was paused. For example, if you set Offset to -1000 and resume logging, NI-DAQmx logs the 1000 samples read before Logging.Pause is enabled. After resuming logging, Offset is automatically reset to 0.

Signal Routing

A single routing API now controls all the digital routing for NI measurements devices. Signal routing controls the mapping of digital signals or triggers across hardware such as digital multiplexers or public trigger buses.

Here is a basic list of features in the signal routing API:

- A single unified signal routing API for all devices supported in NI-DAQmx
- Multi-device routing: a single route will be able to span two devices
- Logical inverter support
- Double driving prevention across public trigger buses

Specifying a Route

A route is a connection between a pair of terminals. The source and destination terminals make a terminal pair. Any time the source or destination terminal of a signal is specified, a route is created. Usually, you specify only one terminal for the route. For example, if you export a signal to the I/O connector, you set the destination terminal, but the source terminal is predetermined by the name of the signal. If you import a hardware trigger for a task, you can set the source terminal, but the destination terminal is predetermined by the name of the trigger.

Single-Device Routing Versus Multi-Device Routing

A single-device route is a connection between two terminals on the same physical device. Before NI-DAQmx, all routes were single-device routes. NI-DAQmx introduces multi-device routing. An example is specifying a terminal on a device as the source of a Start Trigger for a second device.

Creating Multi-Device Routes

NI-DAQmx supports multi-device routing. You simply specify the source terminal and destination terminal. If the two terminals are on different devices, NI-DAQmx uses the trigger bus to route the signal from the source device to the destination device. NI-DAQmx also selects and reserves an available trigger line on the trigger bus.

Plugging in and Registering Your RTSI Cable in MAX

To create a multi-device route, the source and destination devices must share a trigger bus both physically and logically in MAX. For PCI devices, you must register your RTSI cable in MAX. For more information on how to register (or add) a RTSI cable in MAX, refer to **_Measurement & Automation Explorer Help for NI-DAQmx_**. If you do not register your RTSI cable, NI-DAQmx fails to create a route. PXI trigger backplanes are automatically registered when you identify your chassis type in MAX.

Dynamically Selecting Trigger Bus Lines

Management of trigger lines is another feature of NI-DAQmx routing. If you hard-code two measurement tasks to the same trigger line for different signals, at least one of the measurement tasks causes a resource conflict. Multi-device routing allows you to dynamically select trigger lines at run time. This means that NI-DAQmx selects any available trigger line. You can still select a specific trigger bus line by splitting your multi-device route into two single-device routes. However, the two static routes lose the ability to dynamically choose an available trigger at run time.

Task-Based Routing

Task-based routing is the most common form of routing. When you create a hardware trigger or export a hardware signal, you create a task-based route. These routes are embedded in a task. You can use Export Signal function/VI to explicitly make a task-based route. When the task is committed, the route is committed. When the task is cleared, the route is unreserved. Clearing the task does not always clear the route. Refer to Lazy Line Transitions for more information.

Related concepts:

- Lazy Line Transitions

Immediate Routing

Immediate routing is not associated with any task. An immediate route is a pair of fully qualified terminal names specifying the source and destination of the route. When an immediate route is created, the route gets committed to hardware immediately. Because an immediate route does not have a task governing its lifetime, you need to actively destroy the route. Create an immediate route with the Connect Terminals function/VI and destroy it with the Disconnect Terminals function/VI. Also, if you make an immediate route multiple times with several calls to Connect Terminals, only one call to Disconnect Terminals releases the route. There are other ways to destroy routes such as resetting the device. Refer to Device Resetting and Interactions with Routing for more information.

Related concepts:

- Syntax for Terminal Names
- Device Resetting and Interactions with Routing

Logical Inversion of Signals

If you route a signal to or from an external device, you might need to invert the polarity of the signal. For example, you may need to change a high gating signal to a low gating one or look at falling edges instead of rising edges. With routing in NI-DAQmx, you can invert a signal during the routing. If there is an inverter available along the route, the inversion of the signal takes place. Inversion could fail if an overlapping route has previously reserved the inverter with an incompatible configuration.

Routing and Hardware Sharing

Two or more routes might overlap in a compatible fashion—especially if these two routes have the same source and destination. When routes overlap in a compatible fashion, the routing software handles this situation.

As an example, assume that two separate tasks make the same route. The resources associated with the routes are not released until both tasks have been unreserved. Mixing task-based and immediate routes is acceptable, too. However, the hardware resources are not released until all task-based routes have been released and the immediate route has been disconnected.

Releasing a task-based route using the Disconnect Terminals function/VI is not possible. You must release task-based routes by unreserving or clearing the task. In LabVIEW, if you explicitly create your task with the Create Task VI, you must clear it with the Clear Task VI. Otherwise, LabVIEW clears your task for you when the top level VI of your program stops executing.

Line Tristating Issues

During device initialization, all terminals on the I/O connector and trigger buses are tristated. Tristated means the terminal is floating or at high impedance. For the terminal to be driven from the device, the tristate buffer associated with the terminal must be enabled.

For instance, assume that you have a device with a single bidirectional terminal on the I/O connector. The terminal on the I/O connector is called the trigger terminal for reference purposes. Also, the trigger terminal of the device is bidirectional because it can accept an external trigger signal or export the internal trigger signal. The exported internal trigger signal could be different from the external trigger signal.

Scenario	Usage and Consequences
The trigger terminal is being driven by an external trigger signal only.	This is a common case for triggering an operation from an external source. As a result of this operation, you must disable the tristate buffer associated with the trigger terminal so that the internal trigger signal does not drive the trigger terminal, too.
The trigger terminal is being driven by the internal device trigger only.	In this case, an internally generated trigger is starting the device. This signal could be useful for other devices, too. To export this trigger signal, you must enable the tristate buffer associated with the trigger terminal, so the

Scenario	Usage and Consequences
	device can drive the pin with the trigger signal. It is important that there is no external signal hooked up to the trigger terminal. If it is inconvenient to unhook the external signal, you must make sure the external signal is at least tristated.
The trigger terminal is being driven by both the internal device trigger AND an external trigger signal.	Driving the trigger terminal both internally and externally is called double driving. If the internal and external sources drive the signal in opposite directions, it signals problems. Usually the driving hardware is damaged, but more extreme consequences can occur as well. Remember to be very careful to avoid double driving any terminals on your I/O connectors.

Lazy Line Transitions

When a task-based route gets created and released, it does not necessarily go away. The hardware resources associated with the route are released, but the configuration might remain so that glitches are minimized.

By default, all tristate buffers associated with I/O connector terminals are disabled. When a task-based route with a destination on the I/O connector is released, the tristate buffer associated with the I/O connector terminal is not disabled. This means that even though the route was released, glitches are minimized on the destination terminal on the I/O connector. If you do not want this behavior, you can disable the tristate buffer associated with the I/O terminal with the Tri-State Output Terminal function/VI. Or, if you initially created the route by exporting a signal with the Export Signal function/VI, you can also disable the tristate buffer by calling the Export Signal function/VI with the same signal name but with an empty string as the output terminal. Putting the terminal back into a tristate mode is necessary if an external signal must be connected to the I/O terminal. If the terminal is not tristated first, double driving the terminal damages the hardware.

All other connectors, such as the RTSI connector, use a different rule. When the task-based route associated with the RTSI connector is released, the tristate buffer associated with the RTSI terminal is disabled. The RTSI bus is a public bus that is

shared by multiple devices. All drivers using the RTSI bus assume that all devices on the bus are tristated. The I/O connector is different because you have full control of it. You must keep track of which terminals are tristated or being driven by internal or external signals.

Device Resetting and Interactions with Routing

When you reset a device in NI-DAQmx, every immediate route and task associated with the device is invalidated. When the task is invalidated, all the routes are invalidated, too. If a task-based route is invalidated using a device reset, its parent task also is invalidated.

For instance, device A is running a task that performs an analog input operation. This same analog input operation receives its Start Trigger from device B. This task spans across device A and B due to the multi-device routing. If device B gets reset, all routes on device B are destroyed. The invalidation of the task-based route on device B causes its parent task on device A to be invalidated, too. You must consider these possible consequences when issuing a device reset. If the route between device A and B is an immediate route, there is not a relationship between the immediate route and the task. This could result in the task not being invalidated. You need to decide if you need to preserve the task.

Device Routing in MAX

To find the device routing table for your device, launch MAX and select **Devices and Interfaces**. Click a device to open a tabbed window in the middle pane. Click the Device Routes tab at the bottom of the pane to display the device routing table.

> ✏️ **Note** MAX does not display the device routing table for SCXI chassis, SCC connector blocks, or RTSI cable devices.

Each cell in the table is an index with the valid source and destination terminal for the device. These are the same terminal names you can find in the Terminal Name I/O control in LabVIEW.

If a route is possible between a source and destination terminal, the intersecting cell is

colored green or yellow. A green cell indicates the route can be made without consuming any important resource of your device. A yellow cell indicates that although the route is possible, something important must be consumed to create the route. Placing the cursor over a yellow square reveals the resource used in the **subsystem used** indicator. Usually, the sacrificed resource is a counter.

When you display the device routing table for a cDAQ chassis or a C Series device, the table contains all of the terminals for the chassis and all devices installed in the chassis.

Counters

This section provides an overview of counters in NI-DAQmx and the two counter measurement method for period and frequency measurements.

Related concepts:

- Counter Parts in NI-DAQmx
- Two Counter Measurement Method

Paired Counters

For more complex and accurate measurements and generations, a counter is paired with another counter with dedicated connections to and from each counter. This pairing allows you to perform such operations as finite pulse-train generations, higher accuracy frequency and period measurements, and cascaded edge counting. Paired counters are generally numbered sequentially. For example, ctr0 and ctr1 are a pair, ctr2 and ctr3 are a pair, and so on.

Two Counter Measurement Method

For period and frequency measurements, you also can use a second counter. For most applications, the low frequency with one counter method is sufficient and desirable because it uses fewer resources. However, if you have a high-frequency or widely varying signal, you can use one of the two counter measurement methods—the high-frequency measurement method or the large-range measurement method. Depending

on the rate of your input signal and measurement method used, your measurement is subject to different amounts of quantization error. In two counter applications, you only need to call the Create Channel function/VI once, specifying only the counter channel to which you want to connect your input signal. NI-DAQmx automatically takes care of making the internal routes necessary to perform the measurement across paired counters.

Related concepts:

- High Frequency Two-Counter Measurement Method
- Large-Range Two Counter Measurement Method
- Quantization Error
- Paired Counters

High Frequency Two-Counter Measurement Method

Use this high-frequency measurement method if you measure a digital frequency or period of a signal with a high frequency component. To perform measurements using this method in NI-DAQmx, a paired counter generates a pulse train with a period specified using the measurement time attribute/property. The measurement time is generally much larger than the period of the input signal being measured to reduce quantization error. However, the measurement time must be small enough to keep the counter from rolling over. The measurement counter counts the number of periods of the input signal that occur during the measurement time, averages the results, and returns the averaged value in the Read function/VI. The value returned is calculated as follows:

Period (in seconds) = Measurement Time / Number of Periods Counted

Frequency (in Hz) = Number of Periods Counted / Measurement Time

To determine if you should use the high-frequency measurement method, refer to the quantization error tables. If the quantization error listed for the one-counter method is too high, use the high-frequency measurement method instead.

Related concepts:

- Paired Counters

- <u>Quantization Error</u>

Large-Range Two Counter Measurement Method

If you measure the digital frequency or the period of a counter signal, you can use this two-counter method to measure signals with large ranges. This method is useful when you have a widely varying signal to measure and would like increased accuracy throughout the entire range. Refer to the quantization error section for more information on increasing measurement accuracy with the large-range measurement method. You can also use this method to measure signal frequencies that are faster than your counter timebase rate as long as the input signal does not exceed the maximum input frequency supported by the counter.

To perform measurements using this method in NI-DAQmx, a paired counter is used to divide the input signal by a value specified using the Divisor attribute/property. However, you need to be careful the Divisor you choose does not cause the counter to roll over. This divisor has the effect of shifting the measurable frequency range upward. The Divisor scales the measured period and returns data according to the following equations:

Period = Measured Period / Divisor

Frequency = Divisor × Measured Period

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

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

$$\text{Frequency} = \left(\frac{100 \text{ kHz}}{2^{24}}\right) \times 1 = 0.006 \text{ Hz and } \left(\frac{100 \text{ kHz}}{2}\right) \times 1 = 50 \text{ kHz}$$

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

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

$$\text{Frequency} = \left(\frac{100 \text{ kHz}}{2^{24}}\right) \times 4 = 0.024 \text{ Hz and } \left(\frac{100 \text{ kHz}}{2}\right) \times 4 = 200 \text{ kHz}$$

To determine if you should use the large-range measurement method, refer to the quantization error tables. If the quantization error listed for the one-counter method in that section is too high, use the large-range measurement method instead.
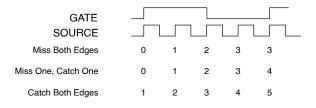
Related concepts:

- Quantization Error
- Paired Counters

Quantization Error

Quantization error is the inherent uncertainty in digitizing an analog value as a result of the finite resolution of the conversion process. Quantization error depends on the number of bits in the converter, along with its errors, noise, and nonlinearities. Quantization error occurs due to phase differences between the input signal and the counter timebase. Depending on how the phase of the input signal and counter timebase align, the count measured has three possibilities:

- **Miss Both Edges**—The counter recognizes neither the first rising edge nor the last rising edge of the counter timebase, giving a count of one less than the expected value.
- **Miss One, Catch One**—The counter only recognizes the first rising edge or the last rising edge of the counter timebase, giving the expected value.
- **Catch Both Edges**—The counter recognizes both the first rising edge and the last rising edge of the counter timebase, giving a count of one more than the expected value.

GATE SOURCE					
Miss Both Edges	0	1	2	3	3
Miss One, Catch One	0	1	2	3	4
Catch Both Edges	1	2	3	4	5

For example, if the counter timebase rate is 20 MHz, and the frequency of the input signal is 5 MHz, the measured value can be 3, 4, or 5 due to quantization error. This corresponds to a measured frequency of 6.67 MHz, 5 MHz, or 4 MHz, resulting in a quantization error of as much as 33%.

Quantization Error with One Counter Time Measurements

For one counter time measurements, the following equation gives the quantization error.

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

You can reduce the quantization error for single counter time measurements by increasing the counter timebase rate. The following table shows the quantization error for various timebase rates with given input signal frequencies:

Actual Frequency of Input Signal	Counter Timebase Rate	Quantization Error
10 Hz	100 kHz	0.01%
100 Hz	100 kHz	0.10%
1 kHz	100 kHz	1.01%
10 kHz	100 kHz	11.11%
10 kHz	20 MHz	0.05%
100 kHz	20 MHz	0.50%
1 MHz	20 MHz	5.26%
2 MHz	20 MHz	11.11%
5 MHz	20 MHz	33.33%

For period and frequency measurements, if the quantization error is too large for your input signal, you might consider using one of the two counter period and frequency measurements.

Quantization Error with High Frequency Two Counter Method

For two counter high-frequency measurements, the following equations give the quantization error.

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

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

Increasing the measurement time reduces the quantization error. The quantization error also decreases with higher frequency input signals. The following table shows the quantization error for various measurement times and input signal frequencies:

Actual Frequency of Input Signal	Measurement Time	Quantization Error
10 kHz	1 ms	10.00%
100 kHz	1 ms	1.00%
1 MHz	1 ms	0.10%
5 MHz	1 ms	0.02%
10 MHz	1 ms	0.01%
10 kHz	10 ms	1.00%
100 kHz	10 ms	0.10%
1 MHz	10 ms	0.01%
5 MHz	10 ms	0.002%
10 MHz	10 ms	0.001%
10 kHz	100 ms	0.10%
100 kHz	100 ms	0.010%
1 MHz	100 ms	0.001%
5 MHz	100 ms	0.0002%
10 MHz	100 ms	0.0001%
10 kHz	1 s	0.010%
100 kHz	1 s	0.0010%
1 MHz	1 s	0.0001%
5 MHz	1 s	0.00002%
10 MHz	1 s	0.00001%

As the table shows, quantization error is reduced at higher frequencies of the input signal. However, the advantage of this measurement method disappears at lower frequency input signals because you need to measure longer to gain accuracy, and you use up more resources.

Quantization Error with Large Range Two-Counter Measurement Method

For two counter large-range measurements, the following equations give the quantization error.

$$Err_{Quantization} = 1 / (Divisor \times Counter\ Timebase\ Rate \times Actual\ Period - 1)$$

$$Err_{Quantization} = Actual\ Frequency / (Divisor \times Counter\ Timebase\ Rate - Actual\ Frequency)$$

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

Actual Frequency of Input Signal	Divisor	Quantization Error
1 kHz	4	0.00125%
100 kHz	4	0.125%
1 MHz	4	1.266%
1 kHz	10	0.0005%
100 kHz	10	0.05%
1 MHz	10	0.5%
1 kHz	100	0.00005%
100 kHz	100	0.005%
1 MHz	100	0.05%

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

measurement.

Quantization Error with Dynamic Averaging Method

For dynamic averaging method, the following equation gives the quantization error.

$Err_{Quantization}$ = Actual Frequency / (Number of Signal Periods x Counter Timebase Rate - Actual Frequency)

To calculate the quantization error, this equation uses the Number of Signal Periods of the input signal that have been measured and averaged. The number of periods is dynamically adjusted based on a combination of the measurement time and divisor settings, plus the period of the input signal being measured as shown in the following equation.

Number of Signal Periods = Max(1, Min(Divisor, Floor(Measurement Time / Signal Period)))

Increasing the divisor or measurement time results in a larger number of signal periods being averaged, and more signal periods in turn reduces the quantization error.

The following table shows examples of quantization error for various divisor and measurement time settings for different input signal frequencies. The counter timebase rate is 100 MHz.

Actual Frequency of Input Signal	Divisor	Measurement Time	Property Used for Frequency Measurement	Number of Signal Periods	Quantization Error
100 Hz	1	0 s	Divisor	1	0.0001%
	0	200 ms	Measurement Time	20	0.000005%
	10	200 ms	Divisor	10	0.00001%
	10	50 ms	Measurement Time	5	0.00002%
	100	50 ms	Measurement Time	5	0.00002%

Actual Frequency of Input Signal	Divisor	Measurement Time	Property Used for Frequency Measurement	Number of Signal Periods	Quantization Error
1 kHz	1	0 s	Divisor	1	0.001%
	0	20 ms	Measurement Time	20	0.00005%
	10	20 ms	Divisor	10	0.0001%
	10	5 ms	Measurement Time	5	0.0002%
	100	50 ms	Measurement Time	50	0.0002%
100 kHz	1	0 s	Divisor	1	0.1%
	0	200 µs	Measurement Time	20	0.005%
	10	200 µs	Divisor	10	0.01%
	10	50 µs	Measurement Time	5	0.02%
	100	50 ms	Divisor	100	0.001%
1 MHz	1	0 s	Divisor	1	1%
	0	20 µs	Measurement Time	20	0.05%
	10	20 µs	Divisor	10	0.1%
	10	5 µs	Measurement Time	5	0.2%
	100	50 ms	Divisor	100	0.01%

Related concepts:

- Dynamic Averaging Method

Dynamic Averaging Method

The dynamic averaging method for frequency and period measurements provides

configuration options that can be set before the start of an acquisition. These options affect the amount of averaging or filtering applied to the counter measurements allowing for tradeoffs in measurement accuracy and noise versus latency. During the acquisition, the counter continuously measures and filters the input signal periods to produce frequency or period measurements. The NI 9326 and 9361 use this method.

Related reference:

- Quantization Error with Dynamic Averaging Method

Related information:

- ni.com/docs

With Counter Filtering Enabled

The NI 9326 can enable low pass filtering of counter signals to give accurate measurement of in-band frequencies while rejecting higher frequency noise. Set the CI.Filter.Enable attribute/property to true to enable filtering. Filtering is enabled by default and is recommended as the easiest way to achieve accurate, low-noise counter measurements.

With filtering enabled, you can then specify the filter cutoff frequency using the CI.Filter.Freq attribute/property to control amount of filtering applied. Lower cutoffs provide lower noise but longer latency. The latency of the filter can be queried using the CI.FilterDelay attribute/property. Each counter can be configured with independent settings.

> **Note** The measurement time, divisor, and maximum measurable period parameters are ignored when filtering is enabled.

With Counter Filtering Disabled

If CI.Filter.Enable is set to false or if counter filtering is not supported on your device, then the dynamic averaging method produces measurements by averaging one or more periods of the input signal. The amount of averaging performed is controlled by the following:

- **Divisor**—Specifies the number of periods of the input signals to measure and average. Use a larger divisor to get more averaging and lower noise measurements.
- **Measurement Time**—Specifies the amount of time over which to measure periods of the input signal to get the average value of the input signal period. Use a longer measurement time to get more averaging and lower noise measurements.
- **Maximum Measurable Period**—Specifies the duration to wait for the input signal transitions before a no valid measurement found state is indicated. Set this to be longer than the slowest frequency you wish to measure.

> **Note** The input divisor and measurement time settings wired into DAQmx Create Virtual Channel VI are ignored. Instead, DAQmx calculates default values for these settings based on the minimum and maximum values wired to the DAQmx Create Virtual Channel VI. You can use a channel property node to override these defaults allows for better control over the amount of averaging applied to your application.

During an acquisition, the dynamic averaging method will average between 1 period and divisor periods of the input signal to determine the input signal's frequency. The measurement time property sets an upper limit to the amount of time that is used for this averaging. As a result, the dynamic averaging method balances measurement accuracy vs. measurement latency throughout the input signal range as shown in the following table.

Signal Frequency	Counter Behavior
High (Divisor*Input Signal Period < Measurement Time)	Measures divisor periods of signal.
Medium (Input Signal Period < Measurement Time < Divisor*Input Signal Period)	Measures between 1 and Divisor periods of the signal, depending on the number of periods that fit within the Measurement Time.
Low	Measure 1 period of signal.

Signal Frequency	Counter Behavior
(Input Signal Period >= Measurement Time)	
Very Low (Input Signal Period > Maximum Measurable Period)	Input signal is slower than the maximum measurable period. 0 Hz returned.

The measurement time and divisor can also be set manually using the property node before the acquisition starts if the auto measurement settings are not desired. The measurement time or divisor can be disabled by setting it to zero.

Measurement Time	Divisor	Notes
0	1	Measure 1 period of the input signal, similar to the 1 Counter (Low Frequency) method.
Input measurement time	0	Counts the number of periods of the input signal that occur during input measurement time, similar to the 2 Counters (High Frequency) method.
0	Input divisor	Counts how long it takes for the input divisor periods of the signal to elapse, similar to the 2 Counters (Large Range) method.

You can view examples of the dynamic averaging method in the **_NI 9361 datasheet_**, which you can find at ni.com/manuals.

Counter Parts in NI-DAQmx

A counter contains several advanced terminals that you can use to perform time measurements and generate pulses. For most applications, NI-DAQmx automatically makes the proper routes from the terminal connector block to the correct advanced terminal with no additional routing required. For advanced applications, you might need to make explicit routes to the internal counter terminals

Related concepts:

- Line Tristating Issues

Advanced Terminals and Common Counter Applications

The following list identifies the names of the advanced terminals as well as their common uses:

- **CtrnGate**—The signal at this advanced terminal is used as the Start Trigger, pause trigger, sample clock, or the input signal being measured. The following table lists how this terminal is used in various applications:

Application	Purpose of Gate Terminal
Pulse Generation	Pause or Start Trigger
One Counter Time Measurements	Input Signal
Two Counter Time Measurements	Unused
Nonbuffered Edge Counting	Pause Trigger
Buffered Edge Counting	Sample Clock
Two-Edge Separation	Second Input Terminal
Position	Z Input Terminal

- **CtrnSource**—The signal at this advanced terminal is either the input terminal for the measurement or the counter timebase. The following table lists how this terminal is used in various applications:

Application	Purpose of Source Terminal
Pulse Generation	Counter Timebase
One Counter Time Measurements	Counter Timebase
Two Counter Time Measurements	Input Terminal
Nonbuffered Edge Counting	Input Terminal
Buffered Edge Counting	Input Terminal
Two-Edge Separation	Counter Timebase
Position	A Input Terminal

- **CtrnInternalOutput**—The signal at this advanced terminal is where the pulsed or toggled output of the counter appears. The output of a counter pulses or toggles when the counter reaches terminal count. When counting down, the counter reaches terminal count when the count reaches zero. When counting up, the counter reaches terminal count when the counter rolls over. To configure the counter to toggle or generate pulses, use the Export Signal function/VI with Counter Output Event as the signal name.

  For the output of a counter to appear at the I/O connector (Ctr0Out, for example), the signal on the internal output terminal must be routed to a terminal on the I/O connector. For pulse generations, this route is automatically made to the dedicated counter output terminal on the I/O connector. For measurements, if you are interested in observing this signal, you need to manually make this route to the appropriate pin on the I/O connector using the Export Signal function/VI with Counter Output Event as the signal name. After you route the internal output of a counter to the I/O connector, the signal remains on the I/O connector until the device is reset or you explicitly tristate the terminal.

- **CtrnAux**—The following table lists how this terminal is used in various applications:

Application	Purpose of Aux Terminal
Pulse Generation	Unused
One Counter Time Measurements	Unused
Two Counter Time Measurements	Unused
Nonbuffered Edge Counting	Optional Count Direction Terminal
Buffered Edge Counting	Optional Count Direction Terminal
Two-Edge Separation	First Input Terminal
Position	B Input Terminal

- **CtrnSampleClock**—You can connect a sample clock in or export it through this terminal.

Counter Parts
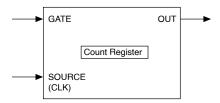
The main parts of a counter include the following:

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

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

A count register increments or decrements the number of edges to count. If the count register decrements, it counts down to zero. The count register size is the number of bits in the counter, and you calculate it as Count Register $= 2^{\text{no. of bits.}}$

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

```
        GATE          OUT

           Count Register

        SOURCE
        (CLK)
```

Configuring a Time-Based Measurement in NI-DAQmx

To configure a measurement, you specify the expected range of the input signal. Based on this range, NI-DAQmx automatically picks the internal timebase that provides the highest resolution for your measurement and uses it as the counter timebase. You also can explicitly specify the source of the counter timebase by setting the Counter Timebase Source attribute/property and the rate of the timebase by setting the Counter Timebase Rate attribute/property. For more information on where to connect input signals, refer to Connecting Counter Signals.

For non-buffered time-based measurements, calling the Read function/VI initiates the measurement and returns the next valid sample. Calling the Read function/VI repeatedly does not return consecutive measurements of the input signal.

To perform buffered time-based measurements, you must use Implicit or Sample

Clock timing, configured using the DAQmx Timing function/VI.

Related concepts:

- Connecting Counter Signals

Implicit Timing

After the acquisition begins, NI-DAQmx measures each consecutive sample of the input signal and stores the measurement in the input buffer. Due to this consecutive measurement, the rate of the input signal implicitly determines the rate of the acquisition.

Sample Clock Timing

Some devices support Sample Clock timing for buffered time-based measurements. After the acquisition begins, your device measures each consecutive sample of the input signal, but does not store it to the input buffer until an active edge of the Sample Clock occurs. Using this timing type, the Sample Clock rate determines the acquisition rate rather than the input signal. When using Sample Clock timing, all measurements returned are a valid, complete cycle of your input signal. Using this method, you can measure signals that are much faster than your sample rate, which minimizes the amount of data transferred from your device to NI-DAQmx.

Related concepts:

- Sample Clock Timing Support for Time-Based Measurements

Averaging

For frequency and period measurements using Sample Clock timing, some devices can return an averaged measurement of all periods since the previous Sample Clock pulse, instead of measuring only the period immediately preceding the current Sample Clock pulse. Use the EnableAveraging DAQmx Channel property/attribute, associated with each of those measurement types, to enable averaging.

Related concepts:

- [Averaging Support](#)

Invalid Initial Samples

Depending on the phase of the input signal in relation to the start of the measurement, the first sample of a buffered measurement is often invalid. For instance, if you are performing a buffered period measurement, and you start the measurement when the input signal is halfway through its current cycle, the measured period for the first sample is half its expected value. Subsequent samples indicate the correct values because they are guaranteed to be taken after a full period of the input signal. For this reason, the first sample of buffered period, pulse width, and semi-period measurements often indicates a smaller value than the actual value. For buffered frequency measurements, the first sample often indicates a higher frequency than the actual frequency. Some devices detect these incomplete samples and discard them.

Related concepts:

- [Incomplete Sample Detection](#)

Pulse Measurement

Some devices support measuring individual pulses, returning each sample as a tuple of frequency/duty cycle, high/low time, or high/low ticks.

Related concepts:

- [Pulse Measurement Support](#)

Configuring a Displacement Measurement with NI-DAQmx
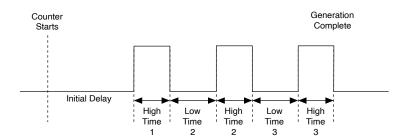
To configure a measurement, specify the initial sensor position through the Initial Angle attribute/property. You also can specify if the Z Input Terminal is used with the Z Index Enable attribute/property. You can configure the reload position on a Z index, and when a Z index position should cause a reload to occur in relation to the A and B signals, by using the Z Index Phase and Z Index Value attributes/properties, respectively.

When performing a single point, or on-demand, displacement measurement, you first arm the counter by calling the Start Task function/VI. Each subsequent read returns the current position of the encoder. If you perform multiple reads without first starting the counter, the counter implicitly starts and stops with each Read function/VI call, and the position is not recorded properly between read calls.

With a buffered displacement measurement, the device latches the current position onto each active edge of the sample clock and stores the position in the buffer. There is no onboard clock for buffered displacement measurement, so you must supply an external sample clock.

Buffered Pulse Generation

You can specify the size of the buffer by calling the DAQmx Configure Buffer function/VI, by specifying the buffer size attribute/property in the buffer property node, or by writing a number of pulse specifications using the DAQmx Write Counter MultiPoint function/VI before starting the task. This is ideal for applications that require pulse-width modulation, such as proportional integral derivative (PID) loop control applications. An example of an Implicit buffered generation would look like the following:
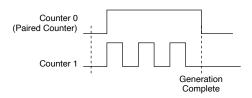
The high and low times provided in the Create Channel function/VI are ignored in this case.

If you do not use a software buffer, all pulses generated will be the same, unless you update the high time and low time while the application is running. This will cause the pulse specifications to be software timed and change on-demand.

You can use the same attributes/properties that create the channel to update the pulse specifications of the pulse train generation. Because you need two attributes/

properties to specify the pulse specifications of the pulse train, the specifications only update when you set one of the two. For example, if you specify the pulse generation in terms of frequency, the `frequency` and `duty cycle` control the specifications of the generation. However, the pulse specifications only update when you set the frequency attribute/property. The same is true when you specify pulse generation in terms of time or ticks; the `low time` and `low ticks` control when the pulse specifications update. When updating the pulse specifications of the pulse generation, a complete period of the current specification generates before the new pulse specification takes effect. Updating the pulse specifications while running is not supported on buffered pulse train generation.

In some devices, such as M Series, E Series, and S Series devices, generating finite pulse trains requires the use of paired counters. In devices that require paired counters on a finite pulse train generation, the first counter (for example, Counter 0) generates a pulse of desired width. The second counter (Counter 1) generates the pulse train, which is gated by the pulse of the first counter. The routing is done internally. The following illustration shows a two counter finite pulse train timing diagram.

Note STC3-based devices, such as X Series devices, do not require paired counters.
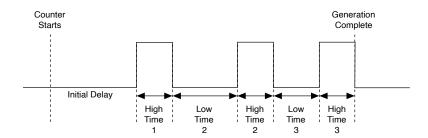
Configuring Triggers for Pulse Generation

You can configure a variety of triggers with pulse generations. All pulse generations support Start Triggers. Single pulse generation and finite pulse train generation also support the Retriggerable attribute/property, or attribute, for Start Triggers. To determine if a pulse is complete and the hardware is ready for another Start Trigger, query the Pulse Done attribute/property. Continuous pulse train generations also support pause triggers. However, you cannot use both the start and the pause trigger at the same time.
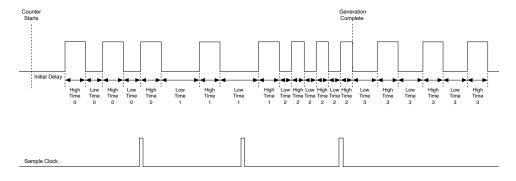
Generating Single Pulses, Finite Pulse Trains, and Continuous Pulse Trains

When generating pulses, you can generate either a single pulse, a finite pulse train, or a continuous pulse train. By default, single pulses are generated unless you use the Timing function/VI with the Implicit or Sample Clock timing types.

In Implicit mode, the `Samples per Channel` input to the Timing function/VI determines the number of pulses to generate for finite pulse trains. In the following illustration, three pulses are generated:

In Sample Clock mode, which uses a software buffer, the `Samples per Channel` input determines the number of distinct pulse specification (High Time/Low Time) transitions to generate. In the following example, `Samples per Channel` is set to three. Notice that the counter increments after the sample clock pulses and the counter signal transitions from high to low. The device generates the high and low values specified in the Create Channel function/VI until the first sample clock arrives.
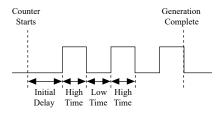
As illustrated previously, to output pulse trains in which pulse specifications are hardware timed and change deterministically, you must use a software buffer, if supported by your device.
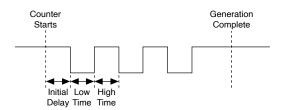
Setting Pulse Train Polarity and the Initial Delay State

The idle state, which controls the pulse train polarity, is applied to the signal when a task transitions to the Committed state (which happens automatically when the task starts). The idle state polarity also determines the state of the initial delay.

When you set the idle state to low, the generation starts low for the initial delay, then transitions to high for the high time. The low time is ignored for the first pulse, but will be repeated before the high time for each subsequent pulse, as shown in the following illustration.

When you set the idle state to high, the generation starts high for the initial delay, then transitions to low for the low time. The high time is ignored for the first pulse, but is repeated after the low time for each subsequent pulse. In both cases, the output rests at the idle state after the pulse generation completes.

Counter Frequency Coercion

The frequency of the counter output must be evenly divisible into the frequency of its timebase. The resulting period is further divided by the duty cycle, which defines how long the signal stays high and how long it stays low for each period. If the requested frequency and duty cycle combination can't be produced exactly, then DAQmx will coerce the signal to the closest frequency and duty cycle that is possible. If you want to define exactly how long the signal stays high and low, use ticks instead.

Related concepts:

- Clock Frequency Coercion

Terminals

A ***terminal*** is a named location where a signal is either generated (output or produced) or acquired (input or consumed). A terminal that can output only one signal is often named after that signal. A terminal with an input that can be used only for one signal is often named after the clock or trigger that the signal is used for. Terminals that are used for many signals have generic names such as RTSI, PXITrig, or PFI.

Related concepts:

- Signal Versus Terminal
- Terminal Names
- Specifying a Route

Signal Versus Terminal

A signal is a means of conveying information. An analog waveform and a digital edge are both examples of signals. The word signal, in this section, refers to the digital edge variety, also known as hardware signals. A terminal, on the other hand, is a named location where a signal is either generated (output or produced) or acquired (input or consumed).

When a terminal shares a name with a signal, it is not always clear which is being referred to—the terminal or the signal. The sample clock provides a good example.

Within most devices, there is a terminal such that the signal at that terminal is always used as the sample clock. So when you refer to the sample clock signal, you refer to this terminal. For instance, for M Series analog input tasks, this terminal is named the ai/SampleClock terminal. For analog output tasks, this terminal is named the ao/SampleClock terminal.

When you use the Timing function/VI to select the source of the sample clock signal for

your analog input task on an M Series device, you choose a signal at some other terminal to act as the source for the ai/SampleClock terminal. In other words, NI-DAQmx connects your chosen terminal (a PFI terminal pin, for instance) to the ai/SampleClock terminal. Selecting the ai/SampleClock terminal as the sample clock source returns an error because a terminal cannot be connected to itself.

Terminal Names

Terminal Names	Explanation
OnboardClock	An alias for the terminal within a device where the default source for a clock can be found. If your application does not set the source of a clock (or uses an empty string as the source), the clock's particular onboard clock is used. For example, the onboard clock for the ai sample clock is the ai Sample Clock Timebase.
PFIn	`Programmable Function Interface`—general-purpose input terminals, fixed-purpose output terminals. The name of the fixed output signal is often placed on the I/O connector next to the terminal as a hint.
PXI_Trign	`PXI Trigger bus`—general-purpose input/output lines.
RTSIn	`Real Time System Integration bus`—general-purpose input/output lines. RTSI7 is the exception. It is the only line to use for the 20 MHz Timebase signal.
ai/SampleClock	A terminal within a device where the analog input sample clock can be found.
ai/StartTrigger	A terminal within a device where the analog input Start Trigger can be found.
ai/ReferenceTrigger	A terminal within a device where the analog input Reference Trigger can be found.
ao/SampleClock	A terminal within a device where the analog output sample clock can be found.
ao/StartTrigger	A terminal within a device where the analog output Start Trigger can be found.
20MHzTimebase	A terminal within a device where the onboard clock source for the master timebase can be found.
di/SampleClock	A terminal within a device where the digital input sample clock can be found.

Terminal Names	Explanation
do/SampleClock	A terminal within a device where the digital output sample clock can be found.
di/ReferenceTrigger	A terminal within a device where the digital input Reference Trigger can be found.
80MHzTimebase	A terminal within a device where the onboard clock source for the master timebase can be found.
100MHzTimebase	A terminal within a device where the onboard clock source for the master timebase can be found.
MasterTimebase	A terminal within a device where the master timebase signal can be found. This signal originates either from the 20MHzTimebase terminal or the RTSI7 terminal. This signal is the onboard source for the Sample Clock Timebases and is one of the possible sources for the AI convert clock timebase.
100kHzTimebase	A terminal within a device where the 100 kHz Timebase signal can be found. This signal is created by dividing the signal at the 20MHzTimebase terminal by 200 and is one of the possible sources for the Sample Clock Timebases.
ai/ConvertClock	A terminal within a device where the AI Convert Clock can be found.
ai/ConvertClockTimebase	A terminal within a device where the AI Convert Clock Timebase can be found. This is the onboard clock source for the AI convert clock.
ai/HoldCompleteEvent	A terminal within a device where the AI Hold Complete Event signal can be found.
AIHoldComplete	The terminal at the I/O connector (external to the device) where the AI Hold Complete Event signal can be emitted.
ai/PauseTrigger	A terminal within a device where the analog input pause trigger can be found.
ai/SampleClockTimebase	A terminal within a device where the AI Sample Clock Timebase can be found. This is the onboard clock source for the AI sample clock.
AnalogComparisonEvent	A terminal within a device where the output of the analog comparison circuit, the Analog Comparison Event signal, can be found. This circuit is active whenever an analog edge or window trigger is configured.
ao/PauseTrigger	A terminal within a device where the analog output pause trigger can be found.
ao/	A terminal within a device where the AO Sample Clock Timebase can be

Terminal Names	Explanation
SampleClockTimebase	found. This is the onboard clock source for the AO sample clock.
di/ SampleClockTimebase	A terminal within a device where the DI Sample Clock Timebase can be found. This is the onboard clock source for the DI sample clock.
do/ SampleClockTimebase	A terminal within a device where the DO Sample Clock Timebase can be found. This is the onboard clock source for the DO sample clock.
Ctr0Out, Ctr1Out, Ctr2Out, Ctr3Out	Terminals at the I/O connector where the output of counter 0, counter 1, counter 2, or counter 3 can be emitted. You also can use Ctr0Out as a terminal for driving an external signal onto the RTSI bus.
Ctr0Gate, Ctr1Gate, Ctr2Gate, Ctr3Gate	Terminals within a device whose purpose depends on the application. Refer to Counter Parts in NI-DAQmx for more information on how the gate terminal is used in various applications.
Ctr0Source, Ctr1Source, Ctr2Source, Ctr3Source	Terminals within a device whose purpose depends on the application. Refer to Counter Parts in NI-DAQmx for more information on how the source terminal is used in various applications.
Ctr0InternalOutput, Ctr1InternalOutput, Ctr2InternalOutput, Ctr3InternalOutput	Terminals within a device where you can choose the pulsed or toggled output of the counters. Refer to Counter Parts in NI-DAQmx for more information on internal output terminals.
PairedCtrInternalOutput	A terminal within a device that chains counters together, creating a paired counter without using any external connections. If your application uses counter 0, PairedCtrInternalOutput refers to the output of counter 1. If your application uses counter 1, PairedCtrInternalOutput refers to the output of counter 0.
PairedCtrOutputPulse	A terminal within a device that chains counters together without using any external connections. If you configure counter 0, PairedCtrOutputPulse refers to the pulsed output of counter 1. If you configure counter 1, PairedCtrOutputPulse refers to the pulsed output of counter 0. Refer to Paired Counters for more information. When the counter reaches terminal count (zero when counting down, its maximum count when counting up), the output of the PairedCtrOutputPulse pulses. By using this terminal, you can chain counters together to create a wider counter, perform buffered edge counting using the other counter as your clock source, perform finite pulse-train generation, and create other custom applications.
di/ ChangeDetectionEvent	A terminal within a device where the change detection event occurs.

Terminal Names	Explanation
FrequencyOutput	A terminal within a device where the Frequency Output signal, which is used to generate a pulse train, is found.
SyncPulse	A terminal within a device where the Sync Pulse signal can be found. This signal is used to synchronize DSA devices.
Ctr0StartArmTrigger	A terminal within a device where the Arm Start Trigger can be found.
Ctr0Aux, Ctr1Aux	Terminals within a device whose purpose depends on the application. Refer to Counter Parts in NI-DAQmx for more information on how the aux terminal is used in various applications.
Ctr0A, Ctr1A, Ctr2A, Ctr3A	Terminals within a device that are used for position measurements.
Ctr0B, Ctr1B, Ctr2B, Ctr3B	Terminals within a device that are used for position measurements.
Ctr0Z, Ctr1Z, Ctr2Z, Ctr3Z	Terminals within a device that are used for position measurements.
port[0..2]line[0..3]	Terminals within a C Series device used for triggers, clocks, and timebase routing. These terminals are not available for digital I/O.
te0/StartTrigger, te1/StartTrigger	A terminal within a device where the timing engine Start Trigger can be found.
te0/ReferenceTrigger, te1/ReferenceTrigger	A terminal within a device where the timing engine Reference Trigger can be found.
te0/SyncPulse, te1/SyncPulse	A terminal within a device where the timing engine Sync Pulse signal can be found. This signal is used to synchronize DSA devices.
te0/SampleClock, te1/SampleClock	A terminal within a device where the timing engine Sample Clock can be found.
it0/SampleClock, it1/SampleClock, it2/SampleClock, it3/SampleClock, it4/SampleClock, it5/SampleClock, it6/SampleClock, it7/SampleClock	A terminal within a device where the input timing engine Sample Clock can be found.
it0/StartTrigger, it1/StartTrigger, it2/	A terminal within a device where the input timing engine Start Trigger can be found.

Terminal Names	Explanation
StartTrigger, it3/ StartTrigger, it4/ StartTrigger, it5/ StartTrigger, it6/ StartTrigger, it7/ StartTrigger	
it0/ReferenceTrigger, it1/ ReferenceTrigger, it2/ ReferenceTrigger, it3/ ReferenceTrigger, it4/ ReferenceTrigger, it5/ ReferenceTrigger, it6/ ReferenceTrigger, it7/ ReferenceTrigger	A terminal within a device where the input timing engine Reference Trigger can be found.
it0/PauseTrigger, it1/ PauseTrigger, it2/ PauseTrigger, it3/ PauseTrigger, it4/ PauseTrigger, it5/ PauseTrigger, it6/ PauseTrigger, it7/ PauseTrigger	A terminal within a device where the input Pause Trigger can be found.
it0/ SampleClockTimebase, it1/ SampleClockTimebase, it2/ SampleClockTimebase, it3/ SampleClockTimebase, it4/ SampleClockTimebase, it5/ SampleClockTimebase, it6/ SampleClockTimebase, it7/ SampleClockTimebase	A terminal within a device where the input Sample Clock Timebase can be found. This is the onboard clock source for the input sample clock.
ot0/SampleClock, ot1/	A terminal within a device where the output timing engine Sample Clock

Terminal Names	Explanation
SampleClock, ot2/ SampleClock, ot3/ SampleClock, ot4/ SampleClock, ot5/ SampleClock, ot6/ SampleClock, ot7/ SampleClock	can be found.
ot0/StartTrigger, ot1/ StartTrigger, ot2/ StartTrigger, ot3/ StartTrigger, ot4/ StartTrigger, ot5/ StartTrigger, ot6/ StartTrigger, ot7/ StartTrigger	A terminal within a device where the output timing engine Start Trigger can be found.
ot0/PauseTrigger, ot1/ PauseTrigger, ot2/ PauseTrigger, ot3/ PauseTrigger, ot4/ PauseTrigger, ot5/ PauseTrigger, ot6/ PauseTrigger, ot7/ PauseTrigger	A terminal within a device where the output Pause Trigger can be found.
ot0/ SampleClockTimebase, ot1/ SampleClockTimebase, ot2/ SampleClockTimebase, ot3/ SampleClockTimebase, ot4/ SampleClockTimebase, ot5/ SampleClockTimebase, ot6/ SampleClockTimebase, ot7/ SampleClockTimebase	A terminal within a device where the output Sample Clock Timebase can be found. This is the onboard clock source for the input sample clock.

Terminal Names	Explanation
cRIO_Trig0, cRIO_Trig1, cRIO_Trig2, cRIO_Trig3, cRIO_Trig4, cRIO_Trig5, cRIO_Trig6, cRIO_Trig7	`cRIO Trigger bus`—general-purpose input/output lines for sharing signals between NI-DAQmx tasks and FPGA IO on a cRIO chassis.

Note M Series, C Series, and X Series devices do not have a master timebase of an arbitrary frequency. These devices use the 20 MHz/80 MHz/100 kHz timebase directly.

Related concepts:

- Counter Parts in NI-DAQmx
- Paired Counters
- Analog Input Accessory Terminal Names
- Analog Output Accessory Terminal Names
- Counter Accessory Terminal Names
- Digital Accessory Terminal Names
- Syntax for Terminal Names

Analog Input Accessory Terminal Names

The following table lists the revised names for analog input terminal names.

Original Terminal Names	Revised Terminal Names	Explanation
AIGND, ACHGND	AIGND	The reference point for referenced single-ended measurements and the bias current return point for differential measurements
ACH#	AI#	AI0, AI1, and so on; the analog input channels
AISENSE	AISENSE	The reference point for NRSE measurements using channels 0-15
AISENSE2	AISENSE2	The reference point for NRSE measurements using channels 16-79
AISENSE3	AISENSE3	The reference point for NRSE measurements using channels 80-143
AISENSE4	AISENSE4	The reference point for NRSE measurements using channels 144-207

Original Terminal Names	Revised Terminal Names	Explanation
SCANCLK	AI HOLD COMP	The terminal where the AI Hold Complete Event signal appears
TRIG1	AI START TRIG	Placed as a hint next to the PFI terminal where the AI Start Trigger can be emitted
TRIG2	AI REF TRIG	Placed as a hint next to the PFI terminal where the AI Reference Trigger can be emitted
CONVERT*	AI CONV CLK	Placed as a hint next to the PFI terminal where the AI Convert Clock can be emitted
STARTSCAN	AI SAMP CLK	Placed as a hint next to the PFI terminal where the AI Sample Clock can be emitted

Related concepts:

- Change Detection Considerations for NI 6527 Devices

Analog Output Accessory Terminal Names

The following table lists the revised names for analog output terminal names.

Original Terminal Names	Revised Terminal Names	Explanation
DAC0OUT	AO0	An analog output channel
DAC1OUT	AO1	An analog output channel
EXTREF	AO EXT REF	AO external reference
AOGND	AO GND	The analog output ground
UPDATE*	AO SAMP CLK	Placed as a hint next to the PFI terminal where the AO Sample Clock can be emitted
WFTRIG	AO START TRIG	Placed as a hint next to the PFI terminal where the AO Start Trigger can be emitted

Counter Accessory Terminal Names

The following table lists the revised names for counter terminal names.

Original Terminal Names	Revised Terminal Names	Explanation
GPCTR1_SOURCE	CTR1SOURCE	Placed as a hint next to the PFI terminal where the Ctr1Source signal can be emitted
GPCTR1_GATE	CTR1GATE	Placed as a hint next to the PFI terminal where the Ctr1Gate signal can be emitted
GPCTR1_OUT	CTR1OUT	The name of the terminal where the Ctr1Out signal appears
GPCTR0_SOURCE	CTR0SOURCE	Placed as a hint next to the PFI terminal where the Ctr0Source signal can be emitted
GPCTR0_GATE	CTR0GATE	Placed as a hint next to the PFI terminal where the Ctr0Gate signal can be emitted
GPCTR0_OUT	CTR0OUT	The name of the terminal where the Ctr0Out signal appears
FREQ_OUT	FREQ OUT	The name of the terminal where the output of the 4-bit clock divider signal appears

Digital Accessory Terminal Names

The following table lists the revised names for digital terminal names.

Original Terminal Names	Revised Terminal Names	Explanation
DIO#	P0.#	Ports on devices are referred to by a number. Port A is called port 0, for instance. The # symbol refers to a single digital line
PA#, PB#, and so on	P0.#, P1.#, and so on	Ports on devices are referred to by a number. Port A is called port 0, for instance. The # symbol refers to a single digital line
DIOA#, DIOB#	P0.#, P1.#, and so on	The # symbol refers to a single digital line

Syntax for Terminal Names

The syntax for terminal names is a unique identifier that refers to a physical terminal in

your system. To guarantee the uniqueness of a terminal name across multiple devices, terminal names begin with a forward slash, followed by the name of the device as configured in MAX, such as `Dev1`. A forward slash and the name of the terminal follow the device identifier, such as `PFI3`. For example, the fully qualified terminal name for `PFI3` on `Dev1` is `/Dev1/PFI3`.

For terminals that exist on multiple subsystems or timing engines, the name of the subsystem or timing engine precedes the terminal name. For example, the output terminal of the Start Trigger for the analog input subsystem on `Dev1` is `/Dev1/ai/StartTrigger`.

Related concepts:

- Timing Engines

Coercion

When a value you set cannot be met exactly, NI-DAQmx sometimes adjusts—or coerces—that value to a valid one. Coercion often occurs when an attribute/property supports a set of discrete ranges.

After you set an attribute/property, you can query that attribute/property to determine its actual value after coercion.

Related concepts:

- Input Limit Coercion
- Clock Frequency Coercion
- Counter Frequency Coercion

Input Limit Coercion

Some devices support only a discrete set of device ranges. When you specify input limits, NI-DAQmx coerces those values to fit within one of the supported device ranges.

For instance, suppose your device only supports ranges of 0 to 10 V, -5 to 5 V, and -10 to

0 V. If you set a maximum value of 8 V, NI-DAQmx coerces the maximum value to 10 V. NI-DAQmx also coerces scaled values, including custom scaling. If you have a temperature sensor that outputs 100 mV for every 1 °C, NI-DAQmx coerces a maximum value of 80 °C to 100 °C.

Because NI-DAQmx coerces input limits, code width is calculated based on the coerced values, which can be outside the minimum and maximum values you expect to measure.

Related concepts:

- Device Range
- Input Limits (Maximum and Minimum Values)
- Coercion
- Custom Scales
- Calculating the Smallest Detectable Change—Code Width

Clock Frequency Coercion

Frequencies of clocks must be evenly divisible into the frequency of their timebase. For example, the rate of the Sample Clock must be evenly divisible into the frequency of the Sample Clock Timebase. If you specify a Sample Clock rate that is not evenly divisible into the frequency of the Sample Clock Timebase, NI-DAQmx coerces the Sample Clock rate to one that is valid.

Related concepts:

- Coercion
- Counter Frequency Coercion

Calibration

There are two types of calibration, channel calibration and device calibration.

Related concepts:

- Channel Calibration

- Device Calibration

Device Calibration

What Is Device Calibration?

Device calibration consists of verifying the measurement accuracy of a device and adjusting for any measurement error. Verification consists of measuring the performance of the device and comparing these measurements to the published specifications. During calibration, you supply and read voltage levels or other signals using external standards, then you adjust the device calibration constants. The new calibration constants are stored in the EEPROM. These calibration constants are loaded from memory as needed to adjust for the error in the measurements taken by the device. There are two kinds of calibration, external and self. For more information on calibrating your device with NI-DAQmx, refer to Device Calibration Considerations. For detailed external calibration procedures, refer to ni.com/calibration.

Related concepts:

- Device Calibration Considerations

Related information:

- Calibration Services

External Calibration

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

Self-Calibration (Internal Calibration)

Self-calibration adjusts the calibration constants with respect to an onboard reference stored on the device. The new calibration constants are defined with respect to the calibration constants created during an external calibration to ensure that the measurements are traceable to these external standards. The new calibration constants do not affect the constants created during an external calibration because they are stored in a different area of the device memory. You can perform a self-calibration at any time to adjust the device for use in environments other than those in which the device was externally calibrated. You use the DAQmx Self Calibrate (or DAQmxSelfCal) function/VI to perform a self-calibration.

> **Note** Self-calibration does not require any external connections.

Channel Calibration

Channel calibration is a technique used to achieve higher measurement accuracy. In most applications, device calibration provides sufficient accuracy. However, in applications where the highest degree of accuracy is critical, channel calibration is necessary, but it does not replace device calibration. Channel calibration compensates for various errors, including those introduced by cabling, wiring, and sensors.

Related concepts:

- [Device Calibration](#)

Control in NI-DAQmx

This section explains control concepts as implemented in NI-DAQmx. Timing control loops, synchronizing analog input and output, using control algorithms, single-point real-time applications, and setting priorities for control applications in LabVIEW are described.

For a general introduction to control, independent of the software you are using, refer to the control overview.

Related concepts:

- Timing Control Loops
- Synchronizing Analog Input and Output
- Control Algorithms
- NI-DAQmx Single-Point Real-Time Applications
- Setting Priorities for Control Applications in LabVIEW
- Control Overview

NI-DAQmx Single-Point Real-Time Applications

This section describes sample applications that demonstrate the functionality for hardware-timed single-point operations on real-time platforms.

Most of these applications use the Wait For Next Sample Clock function/VI, which guarantees tight synchronization between the hardware layer and the software layers for hardware-timed single-point tasks. Wait For Next Sample Clock provides an accurate way to correlate application execution to hardware signals, such as the sample clock for the given task, while providing feedback on the overall real-time execution of the control loop.

The following sections present common control applications:

- Hardware-Timed Simultaneously Updated I/O
- Hardware-Timed Simultaneously Updated I/O with Data Exchanges between Time-Critical and Non-Time-Critical Loops
- Hardware-Timed Input, Software-Timed Output
- Hardware-Timed Counter Tasks
- Software-Timed I/O
- Hardware-Timed Simultaneously Updated I/O Using the Timed Loop (LabVIEW Only)

Related concepts:

- Hardware-Timed Simultaneously Updated I/O
- Hardware-Timed Simultaneously Updated I/O with Data Exchanges between Time-Critical and Non-Time-Critical Loops

- Hardware-Timed Input, Software-Timed Output
- Hardware-Timed Counter Tasks
- Software-Timed I/O
- Hardware-Timed Simultaneously Updated I/O Using the Timed Loop (LabVIEW Only)

Hardware-Timed Simultaneously Updated I/O

- **Requirement**—The I/O must be hardware-timed. All output values need to simultaneously update at the arrival of the sample clock edge.
- **Solution**—Use the DAQmx Wait For Next Sample Clock function/VI to verify that a new sample clock edge has not yet occurred.

Advantages
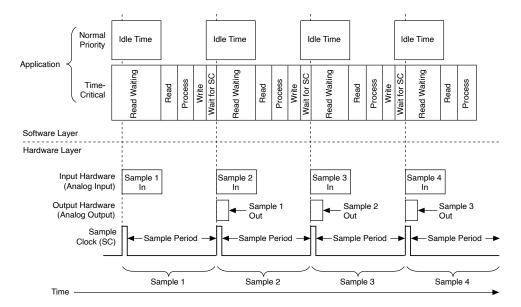
- The current iteration's output samples are guaranteed to be aligned with the next iteration's input samples.
- NI-DAQmx returns an error if the DAQmx Wait For Next Sample Clock function/VI does not start before the next sample clock edge occurs.
- I/O jitter is confined to the jitter of the hardware clock, which is on the order of a few nanoseconds.

Restrictions

Read, process, and write operations are confined to the time available between the moment the device starts acquiring data and the moment the next sample clock edge arrives.

Sample Application—Hardware-Timed Simultaneously Updated I/O

An example of this kind of application is an analog control loop that reads samples from a specific number of analog input channels, processes the data using a control algorithm (such as PID), and writes new control values to the analog output channels.

LabVIEW Example—Hardware-Timed Simultaneously Updated I/O
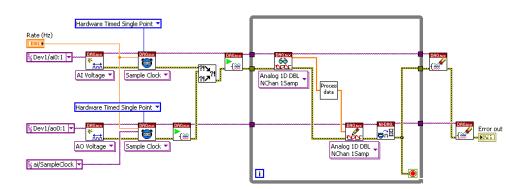
> ✏️ **Note** Although this example is written for LabVIEW users, the principles apply if you are using another ADE, such as LabWindows/CVI.

LabVIEW Example

- Wire the DAQmx Wait For Next Sample Clock VI to one of the hardware-timed tasks. Use dataflow wiring to guarantee that the DAQmx Wait For Next Sample Clock VI executes after the AO Write call.
- If the DAQmx Wait For Next Sample Clock VI does not start before the arrival of the next sample clock edge, it returns an error.

Sample Block Diagram

> ✎ **Note**
>
> - Use only one DAQmx Wait For Next Sample Clock VI within a LabVIEW loop. If you have multiple hardware-timed single-point I/O tasks within the same LabVIEW loop, you can connect the DAQmx Wait For Next Sample Clock VI to any one hardware-timed single-point task within that loop.
> - If, when a cycle overflow occurs, you want to receive a warning rather than an error, set the **DAQmx Real-Time»Convert Late Errors to Warnings** property to True.
> - The DAQmx Wait For Next Sample Clock VI has two modes of operation: Polling and Wait For Interrupt. To change these values, use the **DAQmx Real-Time»Wait For Next Sample Clock Wait Mode** property. Wait For Interrupt mode, which is the default, allows lower priority processes to execute while the time-critical loop waits for the next sample clock. Polling mode allows for higher sampling rates, but it prevents lower priority processes in the system from executing while the time-critical loop waits for the next sample clock.
> - The analog instance of DAQmx Read calls have two modes of operation: Polling and Wait For Interrupt. To change these values, use the **DAQmx Read»Advanced»Wait Mode** property. Wait For Interrupt mode allows lower priority processes to execute while the time-critical loop waits for all the requested samples to be converted. Polling mode allows for higher sampling rates, but it prevents lower priority processes in the system from executing while the time-critical loop waits for the converted analog samples.
> - The specific application shown in this section assumes the use of Wait For Interrupt mode for both the DAQmx Wait For Next Sample Clock VI and the analog instance of the DAQmx Read VI. To change these values, use the **DAQmx Read»Advanced»Wait Mode** and/or **DAQmx Real-Time»Wait For Next Sample Clock Wait Mode** properties.

Hardware-Timed Simultaneously Updated I/O with Data Exchanges between Time-Critical and Non-Time-Critical Loops

- **Requirement**—The I/O needs to be hardware-timed. All output values need to simultaneously update at the arrival of the sample clock edge. Data needs to be exchanged between a time-critical loop and lower-priority processes.
- **Solution**— Use the Wait For Next Sample Clock function/VI to verify that a new sample clock edge has not yet occurred. Place the communication code (usually real-time FIFOs in LabVIEW or a thread-safe queue in LabWindows/CVI) after the Wait For Next Sample Clock function/VI.

Advantages

- The current iteration's output samples are guaranteed to be aligned with the next iteration's input samples.
- NI-DAQmx returns an error if the Wait For Next Sample Clock function/VI does not start before the next sample clock edge occurs.
- I/O jitter is confined to the jitter of the hardware clock, which is on the order of a few nanoseconds.
- Hardware-timed counter input operations have no conversion period similar to that of multiplexed analog input. Therefore, you can place the real-time FIFO, or the thread-safe queue, anywhere within the loop.

Restrictions
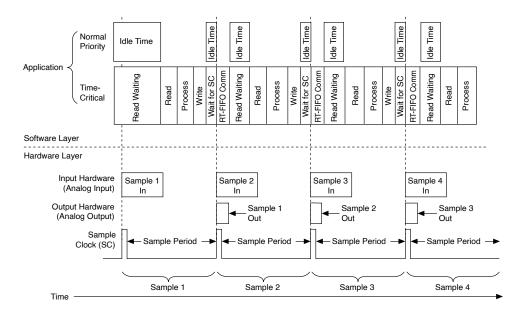
Read, process, and write operations are confined to the amount of time available between the moment the device starts acquiring data and the moment the next sample clock edge arrives.

Sample Application—Hardware-Timed Simultaneously Updated I/O with Data Exchanges between Time-Critical and Non-Time-Critical Loops

An example of this kind of application is an analog control loop that reads samples from a specific number of analog input channels, processes the data using a control algorithm (such as PID), and writes the new control values to the analog output channels. The application uses a real-time FIFO to stop the control loop based on a Boolean value provided by a lower-priority process. A similar approach can employ the use of real-time FIFOs to vary the PID parameters on the fly, or to transfer acquired and control output values to lower-priority processes for data logging and remote monitoring.

Timing Diagram
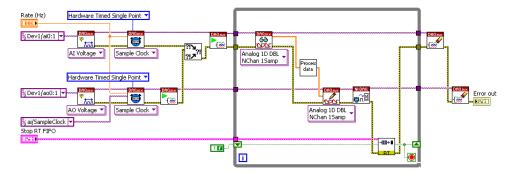
LabVIEW Example—Hardware-Timed Simultaneously Updated I/O with Data Exchanges between Time-Critical and Non-Time-Critical Loops

> **Note** Although this example is written for LabVIEW users, the principles apply if you are using another ADE, such as LabWindows/CVI.

- Wire the Wait For Next Sample Clock VI to one of the hardware-timed tasks. Use dataflow wiring to guarantee that the Wait For Next Sample Clock VI executes after the AO Write call.
- Use dataflow wiring to guarantee that the real-time FIFO operations execute after the Wait For Next Sample Clock VI executes.
- If the Wait For Next Sample Clock VI does not execute before the arrival of the next sample clock edge, it returns an error.

Sample Block Diagram

> **✎ Note**
>
> - Use only one Wait For Next Sample Clock VI within a LabVIEW loop. If you have multiple hardware-timed I/O tasks within the same LabVIEW loop, you can connect the Wait For Next Sample Clock VI to any one hardware-timed single-point task within that loop.
> - If, when a cycle overflow occurs, you want to receive a warning rather than an error, set the `DAQmx Real-Time»Convert Late Errors to Warnings` property to True.
> - Although you do not have to place the real-time FIFO code after the Wait For Next Sample Clock VI, it is highly recommended that you do so when dealing with multiple-channel analog input operations on multiplexed devices (such as E and M Series). Because the device can spend up to 50% of the sample period converting samples on the analog input channels, executing the FIFO code during this conversion period has the advantage of using up otherwise idle time.
> - You can increase the Analog Input Conversion Rate manually through the DAQmx Timing Property Node. This reduces the total amount of time spent converting the requested number of samples. It is important to consider the minimum settling time specifications for the complete data acquisition system to avoid signal degradation and interference.

Hardware-Timed Input, Software-Timed Output

- **Requirement**—An analog input task must be hardware-timed. The output task does not need hardware synchronization with the sample clock edge.
- **Solution**—Use the DAQmx real-time Report Missed Samples attribute/property, which returns an error if new samples are available before the read operation

finishes converting samples from the previous iteration.

Advantages

- Input samples are hardware-timed.
- Read, process, and write operations can overflow into the next sample period, as long as enough time remains for the subsequent read operation to complete on the next set of input samples. An application that acquires data from multiple channels on multiplexed devices (such as E Series and M Series) has to wait for the device to convert input samples before the read operation can return. By allowing process and write operations to overflow into the next sample period, the application takes advantage of otherwise idle time. This enables the application to achieve higher control-loop rates.
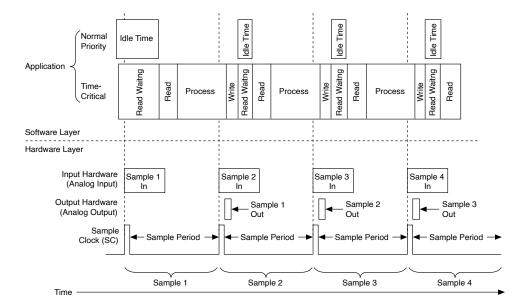
Restrictions

Output updates suffer from software jitter because they are not hardware-timed.

Sample Application—Hardware-Timed Input, Software-Timed Output

An example of this kind of application is an analog control loop that reads samples from a specific number of multiplexed analog input channels, processes the data using a control algorithm (such as PID), and writes the new control values to the analog output channels using a software-timed task.

Timing Diagram
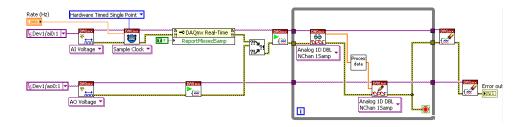
LabVIEW Example—Hardware-Timed Input, Software-Timed Output

> **Note** Although this example is written for LabVIEW users, the principles apply if you are using another ADE, such as LabWindows/CVI.

- Set the Report Missed Samples property for the analog input operation to True.
- The analog input operation returns an error if new samples are available before the read operation finishes converting samples from the previous iteration.

Sample Block Diagram

> **Note**
>
> - If, when an Analog Input Read overflow error occurs, you prefer to receive a warning rather than an error, set the Convert Late Errors to

> Warnings property to True.
> - Do not use the Wait For Next Sample Clock VI and the Report Missed Samples property within the same LabVIEW loop.
> - Only hardware-timed single-point analog input tasks support the Report Missed Samples property.
> - Because the analog output task is software timed, the value is written out as soon as the write call is initiated. It does not wait for a hardware clock to output the data.

Hardware-Timed Counter Tasks

- **Requirement**—Use hardware-timed counter input operations to drive a control loop.
- **Solution**—Use the Wait For Next Sample Clock function/VI to synchronize the counter operations with the counter's sample clock.

Advantages

- Counter tasks allow for flexible timing and event detection operations that can drive the software processing of the control loop. In other words, the control loop can have a dynamic clock rate.
- NI-DAQmx returns an error if the Wait For Next Sample Clock function/VI does not start before the next sample clock edge arrives.

Restrictions

Read, process, and write operations are confined to the time available between the moment the device starts acquiring data and the moment the next sample clock edge arrives.
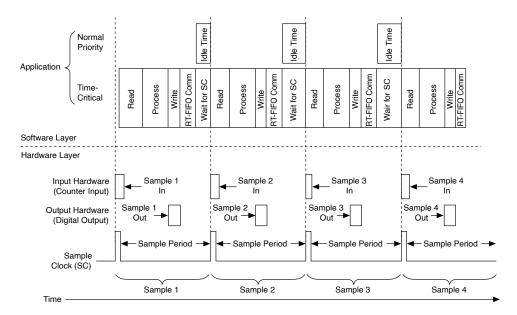
Related concepts:

- [LabVIEW Example—Hardware-Timed Counter Tasks](#)

Sample Application—Hardware-Timed Counter Tasks

An example of this kind of application is a control loop that uses a counter input task, such as count edges, while controlling digital lines based on some predefined control logic. This sample application performs communication through the use of real-time FIFOs. In LabWindows/CVI, you can use a thread-safe queue instead of real-time FIFOs.
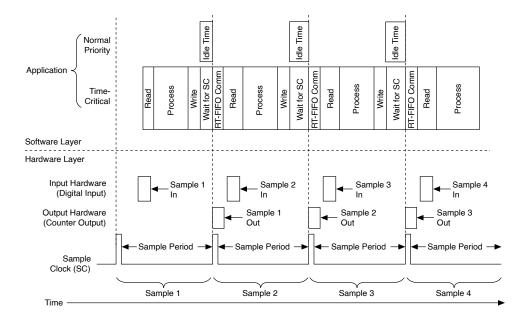
Timing Diagram

Sample Application 2—Hardware-Timed Counter Tasks

Another example application is a control loop that monitors discrete inputs and uses the values to update a counter output task, using pulse frequency mode to generate pulse-width modulation control signals. This example application performs communication through the use of real-time FIFOs. In LabWindows/CVI, you can use a thread-safe queue instead of real-time FIFOs.

Timing Diagram
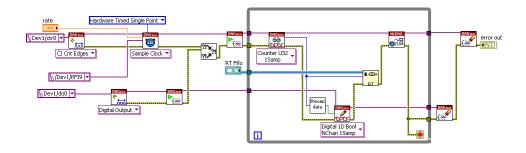
LabVIEW Example—Hardware-Timed Counter Tasks

> **Note** Although this example is written for LabVIEW users, the principles apply if you are using another ADE, such as LabWindows/CVI.

- Wire the Wait For Next Sample Clock VI to the counter input task.
- If the Wait For Next Sample Clock VI does execute before the arrival of the next sample clock edge, it returns an error.
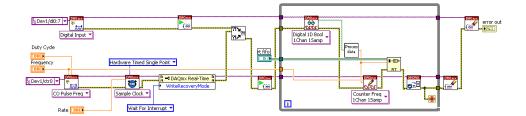
Sample Block Diagram

Example 2

- Wire the Wait For Next Sample Clock VI to the counter output task.
- If the Wait For Next Sample Clock VI does not execute before the arrival of the next sample clock edge, it returns an error.

Sample Block Diagram

> ✏️ **Note**
>
> - Use only one Wait For Next Sample Clock VI within a LabVIEW loop. If you have multiple hardware-timed I/O tasks within the same LabVIEW loop, you can connect the Wait For Next Sample Clock VI to any one hardware-timed single-point task within that loop.
> - If, when a cycle overflow occurs, you want to receive a warning rather than an error, set the **DAQmx Real-Time»Convert Late Errors to Warnings** property to True.
> - Hardware-timed counter operations have no conversion period similar to that of multiplexed analog input. Therefore, the real-time FIFO can be placed anywhere within the LabVIEW loop.
> - NI-DAQmx provides a mechanism to recover after missing a sample clock edge when performing counter writes. If this write recovery mechanism is not successful, NI-DAQmx returns an error, and subsequent operations on that task are no longer hardware timed.
> - The **DAQmx Real-Time»Write Recovery Mode** property allows you to choose between Wait For Interrupt or Polling mode for the recovery mechanism. Wait For Interrupt, which is the default, allows lower priority processes to execute while NI-DAQmx attempts to recover. Polling mode, on the other hand, allows for higher sampling rates.

Hardware-Timed Simultaneously Updated I/O Using the Timed Loop (LabVIEW Only)

- **Requirement**—I/O needs to be hardware-timed. All output values need to simultaneously update at the arrival of the sample clock edge. The application

uses the Timed Loop.

- **Solution**—Use the DAQmx Create Timing Source function/VI to create a timing source that drives a Timed Loop that contains the I/O operations and the control algorithm.

Advantages

- Using a timing source allows you to specify an I/O signal (for example, the sample clock signal) to trigger the execution of Timed Loop iterations.
- Timing sources such as the Control Loop From Task provide strict lateness checking and allow other threads to execute while several analog channels are being multiplexed and sampled.
- The Timed Loop provides feedback as to whether the iterations complete in time.
- Multi-rate applications, using distinct I/O hardware subsystems, are possible by extending this approach to multiple Timed Loops.

Restrictions

- Minor increase in overhead when compared to a regular LabVIEW While Loop
- Requires additional code to handle warm-up iterations

Sample Application—Hardware-Timed Simultaneously Updated I/O Using the Timed Loop (LabVIEW Only)
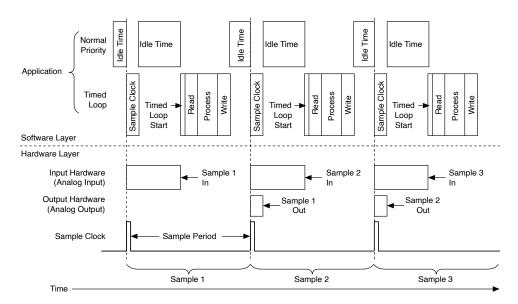
An example of this kind of application is an analog control loop that reads samples from a specific number of analog input channels, processes the data using a control algorithm (such as PID), and writes the new control values to the analog output channels.

You can create such an application with the Control Loop From Task timing source. The Control Loop From Task timing source uses the sample clock signal from the analog input task, which allows strict lateness-checking of all tasks associated with that sample clock.

The Control Loop From Task timing source also allows you to specify a delay between the time the sample clock event is handled and the time the Timed Loop starts executing. This delay, or sleep time, keeps the DAQmx Read function/VI inside the

Timed Loop from using 100% of the CPU time available while waiting for the analog input samples to be multiplexed and digitized.
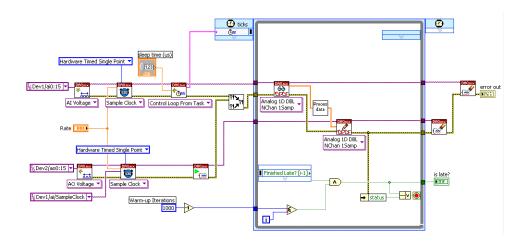
Timing Diagram

LabVIEW Example—Hardware-Timed Simultaneously Updated I/O Using the Timed Loop

- Create a Control Loop From Task timing source for the Timed Loop. This signal serves as the timebase that drives the execution of the Timed Loop.
- The Timed Loop provides feedback to the application as to whether the previous iteration completed in time. It does this through the "Finished Late [i-1]" node.
- Allow a few warm-up iterations to account for the effects of processor-caching and other events that may occur during the first iterations of the loop.
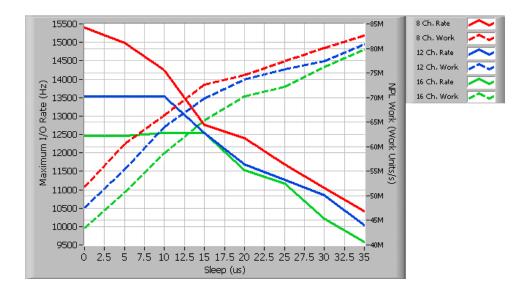
Sample Block Diagram

> ## ✎ Note
>
> - The DAQmx Read VI is implicitly configured to polling mode when using the Control Loop From Task timing source. Polling mode avoids the additional scheduling overhead associated with interrupts inside the Timed Loop.
> - You can increase the Analog Input Conversion Rate manually through DAQmx Timing properties. This reduces the total amount of time spent converting the requested number of samples. It is important to consider the minimum settling time specifications for the complete data acquisition system to avoid signal degradation and interference.
> - Do not use the Wait For Next Sample Clock VI for any of these tasks.
> - Lower-priority processes, including other Timed Loops with lower priorities, can execute while the Timed Loop waits until its next iteration.
> - To optimize multi-channel control applications in which lower-priority threads might require additional processing time, provide a non-zero value for the sleep time (us) parameter of the DAQmx Create Timing Source VI. This non-zero value allows other threads to use the time spent converting analog input samples to perform other tasks such as communication or logging to disk.
> - The maximum amount of sleep time you can set without impacting the overall rate of the application depends on several factors, including the number of analog channels being acquired, the sample conversion rate, and the system's specifications.

The following diagram shows, for multiple channel configurations, the effect of the amount of sleep over the maximum achievable rate and the amount of work lower-priority threads can execute at such rates.

*See benchmark configuration below.

The graph shows that, when acquiring 8 channels using a specific hardware and software configuration, the maximum achievable rate decreases as soon as the amount of sleep time increases from 0 to 5 μs. This is not, however, the case for the 12- and 16-channel configurations, for which increasing the amount of sleep up to 10 and 15 μs respectively has no visible effect on the maximum achievable I/O rates. In the 12- and 16-channel case, the additional sleep interval allows other threads to execute more work (refer to definition below) without affecting the overall I/O rate of the application.

Benchmark Configuration

- Hardware Configuration:
  - NI PXI-8196 RT Controller
  - NI PXI-6070 E Series MIO device
  - NI PXI-6723 Analog Output device
- Software Configuration:
  - LabVIEW Real-Time 8.0
  - NI-DAQmx 8.0
  - Ethernet driver set to polling mode

- Benchmark details:
  - A work unit is defined as the number of times a normal-priority loop can increment an unsigned 64-bit number while the I/O Timed Loop, depicted in the sample block diagram above, runs in parallel with it.
  - The analog input conversion is not explicitly configured. This means that the DAQmx driver auto-calculates it based on the number of channels and desired sample clock rate.

Software-Timed I/O

- **Requirement**—The I/O tasks do not support hardware-timed operations.
- **Solution**—Apply software timing to your time-critical loop by using the Timed Loop in LabVIEW or asynchronous timers in LabWindows/CVI. Configure your NI-DAQmx tasks to use on-demand timing.

Advantages

- You can perform I/O control loops with operations that are not hardware-timed.
- Read, process, and write operations are confined to the software timing period that you define with the Timed Loop or asynchronous timers.
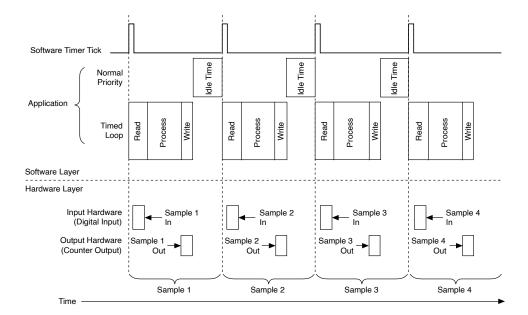
Restrictions

I/O samples suffer from software jitter.

Sample Application—Software-Timed I/O

An example of this kind of application is a digital I/O control loop. The application monitors the state of several discrete inputs and toggles the corresponding output based on the control algorithm. Hardware timing is not available for single-point digital I/O tasks in NI-DAQmx.

Timing Diagram
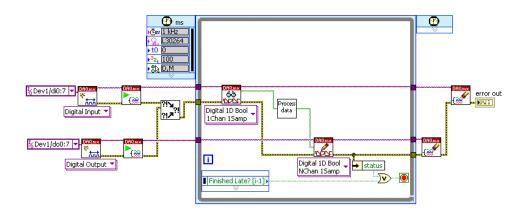
LabVIEW Example—Software-Timed I/O

> **Note** Although this example is written for LabVIEW users, the principles apply if you are using another ADE, such as LabWindows/CVI.

- A Timed Loop running off the system's time sources (millisecond or microsecond resolution) accomplishes the task. Configure the Timed Loop to run at the desired rate.
- Configure all tasks to be software-timed (on demand).
- The Timed Loop provides feedback to the application as to whether the previous iteration completed in time. It does this through the "Finished Late [i-1]" node.

Sample Block Diagram

> **✎ Note**
>
> - The Timed Loop allows the application to adjust its period from within the loop, allowing the implementation of dynamic timing algorithms for control.
> - Lower-priority processes can execute while the Timed Loop waits until its next iteration.
> - Other software timing methods include the use of the Wait and Wait Until next multiple VIs (with microsecond or milliseconds resolution). These methods provide no feedback when the application falls behind.

Timing Control Loops

You can time control loops using software timing or hardware timing. You can also use the Timed Loop structure.

For software timing, the software and operating system determines the rate at which the loop executes. Software timing is not deterministic. Controlling a while loop and using the Wait Until Next ms Multiple VI to handle timing is an example of a software-timed loop. Hardware timing uses the DAQ device internal clock or an external clock to control when a read executes within a loop. The example block diagram shown in **Control Loops** in the Common Applications section uses hardware timing.

The Timed Loop structure is hardware timed. It is ideal for multirate applications. By default, the Timed Loop uses the 1 kHz clock of the Windows operating system as its timing source. Refer to your **LabVIEW Help** for more information about the Timed Loop structure.

Related concepts:
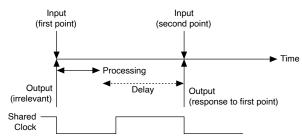
- [Real Time](#)

Control Algorithms

There are many data processing algorithms to consider when creating a control application. You can create custom control algorithms using LabVIEW. You also can use VIs, such as the LabVIEW Control Design and Simulation Module VIs and Functions, to process control application data. This module offers several libraries that you can use to design, analyze, simulate, and deploy dynamic system models, including controllers.

Synchronizing Analog Input and Output

You can time analog output events and synchronize the events to the analog input clock. You can share a common clock for A/D and D/A conversions to ensure that both occur simultaneously. The advantage of this method is that, so long as your software completes in time, you mask out jitter. One caveat is that outputs are always one loop cycle behind the inputs, as shown in the following figure.

Related concepts:

- [Synchronization](#)

Setting Priorities for Control Applications in LabVIEW

The relative priority of parallel processes is important in a control application. Because LabVIEW is multi-threaded, you can separate your application into individual tasks, each with its own priority. By setting priorities, time-critical tasks can take precedence over non-time-critical tasks. The time-critical task must periodically yield processor resources to the lower-priority tasks so they can execute. By properly separating the time-critical task from lower priority tasks, you can reduce application jitter. Refer to the *LabVIEW Real-Time Module Help* for details on assigning priorities to tasks.

Related concepts:

- Jitter Overview for Control Applications

I/O Cycles

The input and output operations along with any processing performed during a single sample clock period are called *I/O cycles*. I/O cycles can consist of reads or writes only, but an I/O cycle in a typical control application consists of reading data, processing that data, then writing a result.

NI-DAQmx Simulated Devices

This section includes information about timing and triggering, task behavior, and reading and writing data with NI-DAQmx simulated devices.

Refer to the *Measurement & Automation Explorer Help for NI-DAQmx* for detailed instructions on creating NI-DAQmx simulated devices and importing NI-DAQmx simulated device configurations onto physical devices.

Related concepts:

- Timing and Triggering with NI-DAQmx Simulated Devices
- Task Behavior of NI-DAQmx Simulated Devices
- Reading and Writing Data with NI-DAQmx Simulated Devices

Timing and Triggering with NI-DAQmx Simulated Devices

With NI-DAQmx simulated devices, the following timing and triggering considerations exist:

- NI-DAQmx simulated devices simulate timing for continuous analog input, digital input, and all output tasks. Timing is not simulated for counter tasks.
- NI-DAQmx simulated devices do not cause a timed loop to execute.
- NI-DAQmx simulated devices support software events. However, events that rely on the hardware, such as a sample clock event, are not supported.
- Triggers always occur immediately.

- Watchdog timers do not expire.

Task Behavior of NI-DAQmx Simulated Devices

NI-DAQmx tasks using NI-DAQmx simulated devices are verified just as tasks are on physical devices. If a property is set to an invalid value, the error returned for an NI-DAQmx simulated device is identical to the error returned for a real device. All resources necessary for the task are reserved for NI-DAQmx simulated devices. RTSI lines, PXI Trigger lines, DMA channels, counters, and so on are counted and reserved just as on physical devices.

> **Note** NI-DAQmx simulated devices cannot be included in the same task with physical devices.

Reading and Writing Data with NI-DAQmx Simulated Devices

NI-DAQmx simulated devices will show up in NI MAX as the appropriate icon (For example, USB, PXI, PCIe, cDAQ Chassis, C Series Module), but will be colored yellow to indicate that the device is simulated. The naming convention for these devices is the same as for real hardware. Data written to an NI-DAQmx simulated device is scaled as if the device were real.

Reading Data

All NI-DAQmx simulated devices return analog input data in the form of a full-scale sine wave with 3% of full-scale noise.

When multiple channels are in the task, the data for each channel is offset 5° in time.

Setting the minimum and maximum signal input range determines the amplitude of the simulated signal. If you decrease the minimum and maximum values the amplitude of the simulated sine wave decreases. However, on real hardware , your analog input signal clips at the maximum and minimum signal input range that is set.

For simulated AO data, set the minimum and maximum values for Signal Output Range. The range cannot exceed the specifications for the real hardware. Set the Test

Signal Type to Sine Wave, Square Wave, or Triangle Wave. If you are using a Test Panel, there are options for DC and Sinewave generation.

Digital data is always returned as if each eight-bit port were a binary counter for both digital input and output.

Counter data is always returned as 0.

Writing Data

Data written to an NI-DAQmx simulated device is scaled as if the device were real. So, any custom scales implemented in a task would affect the simulated data appropriately.

Distributed Applications

This section includes information about deploying applications and using the DAQmx I/O Server for distributed applications.

Related concepts:

- Deployment
- DAQmx I/O Server and Virtual Channels

Deployment

Deployment refers to developing an application so that it can be distributed, or deployed, on a different computer than the one on which the application was developed. To deploy an application, you need the saved application and any configuration information the application and system requires.

When deploying an application, you must coordinate the use of configuration items that can be shared among multiple tasks. This includes devices, scales, and global virtual channels.

Developing Applications for Deployment

You can deploy NI-DAQmx applications in several ways:

- You can use the MAX Export Wizard to deploy an entire setup to another computer, including tasks, channels, scales, and devices.
- You can use the MAX Export Wizard to deploy an entire setup, except the device configurations, to another computer. You might choose to do this if the target computer already has tasks that rely on existing device configurations. In this case, you might have to make modifications after deployment so that your tasks and channels refer to the device configurations on the target computer.
- You can use the MAX Export Wizard initially to deploy a fixed set of device, scale, and global virtual channel configurations among a group of users. Each member of the group can create tasks that rely on the shared configurations, then create applications that use these tasks, and finally share the applications within the group. In some cases, the tasks deploy with the applications automatically. In other cases, you must deploy the tasks separately from the applications. Refer to the Deploying Tasks and Channels section for more information.

Deploying Tasks and Channels

Your tasks and channels deploy automatically with your application under the following circumstances:

- You use LabVIEW Express VIs for your DAQ applications, and the tasks use local channels only.
- You save your tasks within a LabWindows/CVI or Measurement Studio project, and the tasks use local channels only.
- You create your tasks and channels programmatically using the NI-DAQmx API.

You must deploy your tasks and channels using the MAX Export Wizard if you create your tasks and channels in the following ways:

- You create your tasks and channels directly in MAX.
- You create your tasks and channels in LabVIEW from the Task Name and Channel Name controls and do not generate configuration code.
- You create your tasks in LabWindows/CVI and neither generate configuration code nor copy the tasks to your project.

- You create a SignalExpress project that includes a global virtual channel.
- You use LabVIEW Express VIs for your DAQ applications with tasks that use global virtual channels, but do not use code generation.

> ✎ **Note** If your project uses global virtual channels that were added, not copied, generating a LabVIEW diagram from SignalExpress still requires deployment of channels using the MAX Export Wizard.

DAQmx I/O Server and Virtual Channels

In NI-DAQmx 8.0 or later, you can set up distributed applications to combine remote data applications using the NI-DAQmx I/O Server. For instance, you can configure a central computer to monitor other computers that control hardware sensors by using shared variables. You can connect to the NI-DAQmx I/O Server through LabVIEW 8.0 or later, but LabVIEW is not required. If you use a third-party OPC client, you also can access DAQ channels.

> ✎ **Note** You must have at least one global virtual channel defined either in the project or in Measurement & Automation Explorer (MAX) to use the NI-DAQmx I/O Server. Global virtual channels of any I/O type can be bound to shared variables, but tasks cannot.

When using a third-party OPC client, connect to the variable engine server to access DAQ channels on the network.

Refer to the ***LabVIEW Help*** for information about binding to a DAQ channel using a shared variable in a LabVIEW Project.

Functions, VIs, Properties, and Attributes

You program your device primarily with the functions (or VIs in LabVIEW) that make up the NI-DAQmx API. The functions contain the core functionality of the API, but for advanced or uncommon functionality, you can use the following:

- Properties for LabVIEW, Visual C++, Visual Basic .NET, and Visual C#

- Get and Set Attribute functions for ANSI C and LabWindows/CVI

Refer to your function or VI reference help for detailed information on available attributes and properties.

External Reference Sources for Generating Voltage

Devices that support an external voltage reference enable you to maximize the resolution of your device. If the voltages you want to generate do not exceed a certain level and you can supply an external reference voltage at that level, you achieve your device's maximum resolution. The external reference voltage settings are available as a Channel property in the **Analog Output»General Properties»DAC»Reference Voltage**.

You also can use external reference voltages to apply a gain to a DC voltage or to a time-varying waveform. For example, set your external reference voltage level to 1.0, and write a sine wave buffer with values from -1.0 to +1.0 V. When you apply an actual reference voltage of 2.0 V, your signal jumps to ±2 V in amplitude. Increasing the reference voltage level to 3.0 again jumps the signal to ±3 V. Applying a reference voltage level of 0.0 V immediately flat-lines your time-varying voltage signal at 0.0 V.

The terminal you use for external reference sources varies depending on your device.

Related concepts:

- External Reference Sources

Custom Scales

You can create scales to specify a conversion from the prescaled units measured by a channel to the scaled units associated with your transducer or actuator. For input channels, the scale converts samples read to the final scaled units. For example, a scale could convert a voltage to a linear position. For output channels, the scale converts samples written to the prescaled units of the channel. For example, a scale could convert a linear position to a voltage. You also can use scales to calibrate samples read or written so that the final scaled units are identical to the prescaled

units of the channel.

Often, you do not need to create a scale because NI-DAQmx has explicit support for many of the most common transducers, sensors, and actuators. For example, when creating an analog input temperature channel, you can specify the type of transducer (for example, thermistor, RTD, or thermocouple) used to make the measurement when creating the channel. However, if NI-DAQmx does not explicitly support your transducer or actuator, you can create a scale that specifies how to convert from the prescaled units to the scaled units. You can associate the same scale with multiple channels. You do not need to create a scale for each channel if the scale is the same. After a scale is assigned to a channel, the scale applies to all attributes normally expressed in the prescaled units of the channel. For example, if a custom scale, which converts volts to meters, is assigned to a voltage channel, the channels minimum and maximum attributes are expressed in meters.

Prescaled Versus Scaled Units

Prescaled refers to values expressed in the unit of the channel prior to the custom scale being applied. Usually, these prescaled units are volts or amps since scales are most often associated with channels that natively measure or generate signals using these units. However, it is possible to associate a scale with a channel that contains a transducer explicitly supported by NI-DAQmx. In this case, the prescaled units are the units of the channel including the explicitly supported transducer. For example, you can create a analog input resistance channel and associate a scale with this channel. In this example, the prescaled units would be ohms and the scale would specify how to convert from ohms to the desired scaled units.

Scaled refers to values expressed in the final unit after NI-DAQmx applies the custom scale. For example, a linear-position-to-voltage scale is assigned to a voltage output channel. In this case, the prescaled samples are in volts while the scaled samples could be specified in meters. Scaled units are the units that are most convenient for your application. You have complete control over the scaled units when specifying your scale. The scale specifies the conversion from the prescaled units of the signal measured or generated by the channel to your specified scaled units. When you read samples from a channel associated with a scale, the samples are in scaled units. Likewise, when you write samples to a channel associated with a scale, the samples are in scaled units.

You can create scales in the DAQ Assistant or programmatically. When you programmatically associate a scale with a channel, you must set the custom scale name attribute/property to the name of the scale and set the units attribute/property to From Custom Scale.

> ✎ **Note** Unscaled data is not synonymous with prescaled units. Unscaled data refers to an 8-, 16-, or 32-bit integer in the native format of the device. Prescaled refers to the units of measurement, such as volts or amps, before a custom scale is applied.

Related concepts:

- [Unscaled Data](#)

Example—Converting Volts to Revolutions/Minute

Imagine that you have connected an analog output voltage channel to a motor whose speed is proportional to the generated voltage, and you want to create a scale that specifies this conversion. The prescaled units in this case would be volts and the scaled units could be revolutions/minute. You would then specify the equation, table, or map that converts from volts to revolution/minute. After you have created this scale, you would associate the scale with an analog output voltage channel. Now, rather than having to convert between volts and revolutions/minute when operating your application or having to develop additional code in your application to perform this conversion, you can simply write samples in units of revolutions/minute directly to the channel associated with the scale and NI-DAQmx automatically performs the specified conversions. Scales can simplify your code and improve the usability of your application.

Applying Scales That Do Not Monotonically Increase or Decrease

Some scale types allow scales that do not monotonically increase or decrease. This is problematic because application of the scale may not produce the desired results. For example, if multiple prescaled values map to the same scaled value, the conversion from the scaled value to the prescaled values is ambiguous. The conversion is well defined and predictable even in these cases. While not disallowed, non-monotonically increasing scales should be avoided or used with caution.

NI-DAQmx Versus Traditional NI-DAQ (Legacy)

NI-DAQmx replaced Traditional NI-DAQ (Legacy) in 2003. NI-DAQmx and Traditional NI-DAQ (Legacy) have their own APIs, hardware configurations, and software configurations.

NI DAQmx

NI-DAQmx has the following advantages over Traditional NI-DAQ (Legacy):

- DAQ Assistant—a graphical way to configure virtual channels and measurement tasks for your device, and to generate NI-DAQmx code based on your virtual channels and tasks, for use in LabVIEW, SignalExpress, LabWindows/CVI, and Measurement Studio.
- Increased performance, including faster single-point analog I/O and multithreading.
- NI-DAQmx simulated devices for testing and modifying applications without plugging in hardware.
- Simpler, more intuitive APIs for creating DAQ applications using fewer functions and VIs than earlier versions of NI-DAQ.
- Expanded functionality for LabVIEW, including property nodes and waveform data type support.
- Similar APIs and functionality for ANSI C, LabWindows/CVI, and Measurement Studio, including native .NET and C++ interfaces.
- Improved support and performance for the LabVIEW Real-Time Module.

Who Can Use NI-DAQmx

Install and use NI-DAQmx if the following situations apply:

- You are a new user.
- You are using devices supported by NI-DAQmx; refer to the NI-DAQmx Readme for a list of supported devices.
- You are using an OS supported by NI-DAQmx; refer to the NI-DAQmx Readme for a list of supported OSes.

If you are using NI application software with NI-DAQmx, you must use supported

versions of LabVIEW, LabWindows/CVI, Measurement Studio, NI SignalExpress, or the LabVIEW Real-Time Module. Refer to the NI-DAQmx Readme for specific versions of NI application software supported in your version of NI-DAQmx.

If you use one of the Microsoft .NET languages, Visual C# and/or Visual Basic .NET, or a device supported only by NI-DAQmx, such as an M Series device, you must use NI-DAQmx.

You also can use NI-DAQmx with a supported compiler, such as an ANSI C compiler.

Who Must Use Traditional NI-DAQ (Legacy)

Install and use Traditional NI-DAQ (Legacy) if one of the following situations apply:

- You have a device that is not supported by NI-DAQmx, such as the AT E Series multifunction DAQ devices.
- You are using an unsupported version of LabVIEW, LabWindows/CVI, or Measurement Studio.
- You are upgrading from NI-DAQ 6.9.x and have existing applications that you do not want to port to NI-DAQmx now.

NI-DAQmx Device Considerations

This help file contains information specific to analog output (AO) Series devices, B Series devices, C Series devices, digital I/O (DIO) devices, DSA devices, E Series devices, FieldDAQ Devices, M Series devices, S Series devices, SCC devices, SC Express, SCXI devices, switches, timing I/O (TIO) devices, USB DAQ devices, and X Series devices that might help you as you create applications with NI-DAQmx.

Device Groups in NI-DAQmx

Refer to the following list to see how devices are grouped in NI-DAQmx.

FieldDAQ

- FD-11601
- FD-11603
- FD-11605
- FD-11613
- FD-11614
- FD-11634
- FD-11637

CompactRIO Single-Board Controllers

- sbRIO-9603
- sbRIO-9608
- sbRIO-9609
- sbRIO-9628
- sbRIO-9629
- sbRIO-9638

CompactRIO Controllers

- cRIO-9040

- cRIO-9041
- cRIO-9042
- cRIO-9043
- cRIO-9045
- cRIO-9046
- cRIO-9047
- cRIO-9048
- cRIO-9049
- cRIO-9053
- cRIO-9054
- cRIO-9055
- cRIO-9056
- cRIO-9057
- cRIO-9058

X Series DAQ (Multifunction I/O)

- NI 6320
- NI 6321
- NI 6323
- NI 6341
- NI 6343
- NI 6345
- NI 6346
- NI 6349
- NI 6351
- NI 6353
- NI 6355
- NI 6356
- NI 6358
- NI 6361
- NI 6363
- NI 6365
- NI 6366
- NI 6368
- NI 6374
- NI 6375

- NI 6376
- NI 6378
- NI 6386
- NI 6396

M Series DAQ (Multifunction I/O)

- NI 6210/11/12/15/16/18
- NI 6220
- NI 6221
- NI 6224
- NI 6225
- NI 6229
- NI 6230/32/33/36/38/39
- NI 6250
- NI 6251
- NI 6254
- NI 6255
- NI 6259
- NI 6280
- NI 6281
- NI 6284
- NI 6289

S Series DAQ

- NI 6110
- NI 6111
- NI 6115
- NI 6120
- NI 6122
- NI 6123
- NI 6124
- NI 6132
- NI 6133
- NI 6143
- NI 6154

SC Express

- NI 4300
- NI 4302
- NI 4303
- NI 4304
- NI 4305
- NI 4309
- NI 4310
- NI 4322
- NI 4330
- NI 4331
- NI 4339
- NI 4340
- NI 4353
- NI 4357

C Series, Network DAQ, and USB DAQ

- NI 9201
- NI 9202
- NI 9203
- NI 9205
- NI 9206
- NI 9207
- NI 9208
- NI 9209
- NI 9210
- NI 9211
- NI 9212
- NI 9213
- NI 9214
- NI 9215
- NI 9216
- NI 9217
- NI 9218
- NI 9219

- NI 9220
- NI 9221
- NI 9222
- NI 9223
- NI 9224
- NI 9225
- NI 9226
- NI 9227
- NI 9228
- NI 9229
- NI 9230
- NI 9231
- NI 9232
- NI 9234
- NI 9235
- NI 9236
- NI 9237
- NI 9237 (DSUB)
- NI 9238
- NI 9239
- NI 9242
- NI 9244
- NI 9246
- NI 9247
- NI 9250
- NI 9251
- NI 9252
- NI 9253
- NI 9260
- NI 9262
- NI 9263
- NI 9264
- NI 9265
- NI 9266
- NI 9269
- NI 9326
- NI 9344

- NI 9361
- NI 9375
- NI 9401
- NI 9402
- NI 9403
- NI 9411
- NI 9421
- NI 9422
- NI 9423
- NI 9425
- NI 9426
- NI 9435
- NI 9436
- NI 9437
- NI 9469
- NI 9472
- NI 9474
- NI 9475
- NI 9476
- NI 9477
- NI 9478
- NI 9481
- NI 9482
- NI 9485
- NI 9775

CompactDAQ Chassis

- NI cDAQ-9171
- NI cDAQ-9174
- NI cDAQ-9178
- NI cDAQ-9179
- NI cDAQ-9181
- NI cDAQ-9184
- NI cDAQ-9185
- NI cDAQ-9188
- NI cDAQ-9189

- NI cDAQ-9188XT
- NI cDAQ-9191

CompactDAQ Controllers

- NI cDAQ-9132
- NI cDAQ-9133
- NI cDAQ-9134
- NI cDAQ-9135
- NI cDAQ-9136
- NI cDAQ-9137

AO Series

- NI 6703
- NI 6704
- NI 6711
- NI 6713
- NI 6722
- NI 6723
- NI 6731
- NI 6733
- NI 6738
- NI 6739

Digital I/O

- NI 6501
- NI 6503
- NI 6508
- NI 6509
- NI 6510
- NI 6511
- NI 6512
- NI 6513
- NI 6514
- NI 6515
- NI 6516

- NI 6517
- NI 6518
- NI 6519
- NI 6520
- NI 6521
- NI 6525
- NI 6527
- NI 6528
- NI 6529
- NI 6533
- NI 6534
- NI 6535
- NI 6535B
- NI 6536
- NI 6536B
- NI 6537
- NI 6537B
- NI DIO-32HS
- NI DIO-96

TIO Series

- NI 6601
- NI 6602
- NI 6608
- NI 6612
- NI 6614
- NI 6624

Dynamic Signal Acquisition (DSA)

- NI 4431
- NI 4432
- NI 4461
- NI 4462
- NI 4463
- NI 4464

- NI 4466
- NI 4472/B
- NI 4474
- NI 4480
- NI 4481
- NI 4492
- NI 4495
- NI 4496
- NI 4497
- NI 4498
- NI 4499
- NI 4610

Academic Devices

- NI ELVIS II
- NI ELVIS II+
- NI myDAQ

Low-Cost USB

- NI 6000
- NI 6001
- NI 6002
- NI 6003
- NI 6008
- NI 6009
- NI TC01

Analog Triggering

This section contains information about analog triggering for C Series, NI ELVIS II Family, DSA, E Series, M Series, S Series, and SC Express devices.

Valid Analog Trigger Sources for DSA Devices

The analog trigger source must be a channel included in your physical channel list. PFI 0 is not a valid analog trigger source. PFI 0 is reserved for digital triggers.

Analog Triggering Considerations for TestScale Modules and C Series, E Series, M Series, and S Series Devices

> ✏ **Note** Not all E Series, M Series, S Series, and C Series devices support analog triggering. Refer to the specifications for your device to determine if your device supports analog triggering.

Certain C Series, E Series, M Series, S Series devices and TestScale modules contain a single analog trigger circuit that you can configure for analog triggering. The analog trigger circuitry is a shared resource for the device, and any of the subsystems can use it. This trigger circuitry supports level and slope triggering with hysteresis as well as analog window triggering. After it is configured, the output of this circuitry appears as the Analog Comparison Event, which can be the source for various triggers and clocks within the analog input, analog output, and counter subsystems.

Sharing an Analog Trigger for C Series, E Series, M Series, and S Series Devices

Even though the analog trigger is a shared resource, only one analog input or analog output task at a time can configure and reserve it. If you want to share the analog trigger among multiple tasks, configure and reserve it in one task, and use the trigger in subsequent tasks by referring to the source of your trigger, clock, or signal of interest as the Analog Comparison Event. For tasks that support multiple types of analog triggers within the same task, all triggers must share the same configuration settings, or you receive an error. For instance, if you want to use an analog trigger for both your Start and Reference Trigger within an analog input task, the configuration settings for the start and Reference Trigger must be identical.

E Series and S Series Valid Sources for the Analog Trigger

- PFI 0

  Typically, when configuring an analog trigger, you connect your analog signal to

the PFI 0 terminal. Because PFI 0 is the trigger source for both analog and digital signals, NI-DAQmx automatically tristates this terminal when a task exporting a signal on the terminal is not in the committed or running state. This behavior when exporting a signal on PFI 0 differs from typical task-based routing with other PFI lines. It prevents accidental connections of an analog signal directly to digital circuitry, which could permanently damage the device. Also, notice that when connecting an analog signal to PFI 0, the terminal configuration is referenced single-ended.

Even when PFI 0 is not the source of your analog trigger, you cannot use PFI 0 for other digital signal routes because the analog trigger takes over the PFI 0 terminal internal to the device when it is enabled. If you try to use the analog trigger and PFI 0 for digital signals at the same time, you receive a routing error.

> **Note** On PXI-6132/6133 and PXIe-6124 devices, you cannot use PFI 0 as the source of an analog trigger. On PXI-6132/6133, the analog triggering circuitry still reserves PFI 0 for internal routing.

- Analog Input Channel

    In addition to PFI 0, analog input tasks can trigger off of one of the analog input channels being sampled. Because E Series devices use a scanning architecture, many restrictions are placed on how you can use an analog trigger when the source is one of the channels you are sampling. When you use an analog Start Trigger, the trigger channel must be the first channel in the channel list. When you use an Analog Reference or Pause Trigger, and the analog channel is the source of the trigger, there can be only one channel in the channel list. If you have more than one channel for Pause or Reference Triggers, you must use PFI 0. Since S Series devices do not use a scanning architecture, none of these restrictions apply. Therefore, for an S Series device, you can use any analog input channel as the source of the trigger regardless of how many channels are being sampled or the order of the trigger channel in the sequence.

- Scaling with PFI 0 and Analog Input Channels

    Scaling, including custom scales, is not applied if PFI 0 is the trigger source. For instance, you would specify the `DAQmx Trigger Analog Edge Level` attribute/property in volts. However, if you use an analog input channel as the

trigger source, you could use scaled units.

M Series Valid Sources for the Analog Trigger

- APFI 0 and APFI 1

  When configuring an analog trigger, connect your analog signal to either the APFI 0 or APFI 1 terminal and specify APFI 0 or APFI 1 as your trigger source.

- Analog Input Channel

  In addition to APFI 0 and APFI 1, analog input tasks can trigger off of one of the analog input channels being sampled. Because M Series devices use a scanning architecture, many restrictions are placed on how you can use an analog trigger when the source is one of the channels you are sampling. When you use an Analog Start Trigger, the trigger channel must be the first channel in the channel list. When you use an Analog Reference or Pause Trigger, and the analog channel is the source of the trigger, there can be only one channel in the channel list. If you have more than one channel for Pause or Reference Triggers, you must use APFI 0 or APFI 1.

- Scaling with APFI 0, APFI 1, and Analog Input Channels

  Scaling, including custom scales, is not applied if APFI 0 or APFI 1 is the trigger source. For instance, you would specify the DAQmx Trigger Analog Edge Level attribute/property in volts. However, if you use an analog input channel as the trigger source, you could use scaled units.

Device Calibration and Accuracy of the Analog Trigger for E Series, M Series, C Series, or S Series Devices

The trigger DACs in the analog trigger circuitry on an E Series, M Series, C Series, or S Series device typically contain four less bits of accuracy than the ADC of the device. No hardware calibration is provided for the analog trigger circuitry. In addition, the propagation delay from when a valid trigger condition is met to when the analog trigger circuitry emits the Analog Comparison Event may have an impact on your measurements if the trigger signal has a high slew rate. If you find these conditions have a noticeable impact on your measurements, you can perform software calibration on the analog trigger circuitry by configuring your task as normal and applying a

known signal for your analog trigger. Comparing the observed results against the expected results, you can calculate the necessary offsets to apply in software to fine tune the desired triggering behavior.

C Series and TestScale Module Valid Sources for the Analog Trigger

The NI 9205 and TS-15100 have no APFI 0 or APFI 1 terminal. Analog input tasks using the NI 9205 or TS-15100 can trigger off one of the analog input channels being sampled. When you use an Analog Start Trigger, the trigger channel must be the first channel from the NI 9205 or TS-15100 in the channel list, but channels from other C Series devices can come first. When you use an Analog Reference or Pause Trigger, you can use only one channel from the NI 9205 or TS-15100 in the channel list, but you can use channels from other C Series devices. You can combine Analog Start, Reference, and Pause Triggers with different configuration settings by using multiple NI 9205 devices. All analog triggers on the same device must share the same configuration settings.

The NI 9775 can only use an analog trigger that is an active channel. See **_NI 9775 Considerations_** for further restrictions on analog trigger usage. When you use a trigger with the NI 9775 in Analog Multi Edge, all of the trigger channels must come from the same device. When using multiple NI 9775 modules, those modules can be in the same task as long as the trigger source comes from one of the NI 9775 modules.

Related concepts:

- NI 9775 Considerations

Triggering Considerations for NI ELVIS II Family Devices

On NI ELVIS II Family devices, APFI 0 and APFI 1 are not valid analog trigger sources. When using a Reference Trigger, supported trigger types are analog edge, analog window, and digital edge. When using a Start Trigger, digital edge is the only supported trigger type. Valid digital trigger sources are PFI 15 and SYNC terminals. NI ELVIS II+ also supports analog reference triggering with multiple channels for oscilloscope channels.

Analog Triggering Considerations for SC Express Devices

Analog triggering on SC Express devices uses one of the device's ADCs to sample the trigger signal. If the frequency of the trigger signal is higher than the ADC's Nyquist frequency (half of the sampling rate), aliasing may prevent triggers from being detected reliably.

> **Note** Analog trigger condition thresholds are reverse-calibrated internally by NI-DAQmx, so it is recommended that you self-calibrate your SC Express device (if self-calibration is supported on your device) prior to running an analog triggered acquisition.

The frequency of the trigger signal is also limited by lowpass filtering on the device's analog front end. On the NI 4300 and NI 4310, the lowpass filter can be controlled using the AI.Lowpass.Enable and AI.Lowpass.CutoffFreq channel attributes/properties. On the NI 433x devices, the lowpass filter cutoff frequency is determined by the sampling rate. The analog trigger source can be set to any one of the channels on the device.

Analog triggering on SC Express devices supports only Reference Triggers, not Start and Pause Triggers. Digital triggering on SC Express devices does not have this restriction.

> **Note** Analog triggers are not supported on the NI 4309, NI 4322, NI 4353, or NI 4357.

> **Note** The NI 4302/4303/4304/4305, NI 4310 and NI 4340 support only analog reference triggering.

Device Calibration Considerations

Device calibration consists of verifying the measurement accuracy of a device and adjusting for any measurement error. Verification consists of measuring the performance of the device and comparing these measurements to the published

specifications. During calibration, you supply and read voltage levels or other signals using external standards, then you adjust the device calibration constants. The new calibration constants are stored in the EEPROM. These calibration constants are loaded from memory as needed to adjust for the error in the measurements taken by the device.

AO Series Calibration

Your device uses software calibration to fine-tune the analog output circuitry. The software must be programmed (or loaded) with certain numbers called calibration constants. Those constants are stored in nonvolatile memory (EEPROM) on your device or are maintained by NI-DAQmx. To achieve specification accuracy, you should self-calibrate your device just before a measurement session but after your computer and the device have been powered on and warmed up for at least 15 minutes. You should allow this same warm-up time before performing any calibration of your system. Frequent calibration produces the most stable and repeatable measurement performance. The device is not harmed in any way if you recalibrate it often.

Static AO devices, such as the NI 6703 and NI 6704, do not self-calibrate or automatically calibrate. You must use a manual procedure to calibrate static AO devices. Refer to the calibration procedure for your device.

> ✎ **Note**
>
> - Calibrating your AO device takes some time. Do not be alarmed if the Self-Calibrate or Adjust AO Series Calibration function/VI takes several seconds to execute.
> - For best results, stop any ongoing tasks and disconnect any unnecessary external connections before running calibration.

NI-DAQmx automatically loads calibration constants into the software whenever you call functions/VIs that depend on them.

Related concepts:

- Device Calibration Signal Connections for AO Series Devices

Related information:

- Calibration Procedures

C Series Calibration

Your device uses software calibration to adjust the software scaling of signals read from and produced by your device. Using a precise reference signal, your device measures and calculates scaling constants for analog input and analog output. The scaling constants are stored in nonvolatile memory (EEPROM) on your device. NI recommends that you calibrate your device just before a measurement session but after your module has been powered on and warmed up for at least 15 minutes. You should allow this same warm-up time before performing any calibration of your system. Frequent calibration produces the most stable and repeatable measurement performance. The device is not harmed in any way if you recalibrate it often.

> **Note**
>
> - Calibrating your device takes some time. Do not be alarmed if the Adjust C Series Calibration function/VI takes several seconds to execute.
> - For best results, stop any ongoing tasks and disconnect any unnecessary external connections before running calibration.
> - The NI 9205 and NI 9206 are calibrated using the M Series calibration process.

C Series devices do not self-calibrate or automatically calibrate. You must use a manual procedure to calibrate these devices.

Related information:

- Calibration Procedures

Virtual Channel Calibration Support

The following devices do not support NI-DAQmx virtual channel calibration:

- NI DAQPad-6015
- NI DAQPad-6016
- NI PCI-6010
- NI PCI-6013
- NI PCI-6014
- NI USB-6000
- NI USB-6001
- NI USB-6002
- NI USB-6003
- NI USB-6008
- NI USB-6009
- SensorDAQ

DSA Calibration

Your device contains digital correction circuitry to compensate for gain and offset errors in the analog and ADC circuitry. The gain and offset calibration constants are stored in nonvolatile memory (EEPROM) on your device. NI-DAQmx writes these calibration constants to the digital correction circuitry.

To achieve the maximum accuracy, you should perform external calibration at least once per year (the recommended external calibration interval) and perform self-calibration prior to measurement sessions or otherwise, as desired. You should calibrate your device only after your computer and the device have been powered on and warmed up for at least 15 minutes.

Self-Calibration

Self-calibration is executed with the Self Calibrate VI/function. When you self-calibrate a DSA device, you do not need signal connections. However, for devices with analog output channels, values generated on those output channels can change during the calibration process. If you have external equipment connected to the AO channels and changing the AO voltage could damage the external equipment, you should

disconnect the external equipment before performing the self-calibration.

> ✎ **Note** Some DSA devices do not support self-calibration.

External Calibration

External calibration is performed using a customized calibration program and external test equipment that has itself been calibrated to the required accuracy or standard. This operation is usually performed by a specialized metrology laboratory. The equipment and connections required to externally calibrate a device varies depending on the device category. For devices that support DC coupling, you need a stable and accurate DC voltage signal to calibrate the AI subsystem. For devices that support AC coupling, a sinusoidal source may be required for the calibration procedure instead of the DC source. NI 4461, 4462, 4463, 4464, 4480, and 4481 devices also include an adjustable frequency timebase. You need a stable sinusoidal frequency source to calibrate this timebase. The PCI-4461, PXI-4461, PXIe-4463, and USB-4431 devices also support analog output. You need a digital multimeter (DMM) to calibrate the AO subsystem. The DC voltage, frequency source, and DMM can be manually or automatically controlled and switched between channels, depending on the nature of the customized calibration program.

E Series Calibration

Your device uses hardware calibration to adjust the analog circuitry. This calibration is done with calibration digital-to-analog converters, called calDACs, that fine-tune the analog circuitry. The calDACs must be programmed (or loaded) with certain numbers called calibration constants. Those constants are stored in nonvolatile memory (EEPROM) on your device or are maintained by NI-DAQmx. NI recommends that you self-calibrate your device just before a measurement session but after your computer and the device have been powered on and warmed up for at least 15 minutes. You should allow this same warm-up time before performing any calibration of your system. Frequent calibration produces the most stable and repeatable measurement performance. The device is not harmed in any way if you recalibrate it often.

> ✎ **Note**

- Calibrating your MIO or AI device takes some time. Do not be alarmed if the Self-Calibrate or Adjust E Series Calibration function/VI takes several seconds to execute.
- For best results, stop any ongoing tasks and disconnect any unnecessary external connections before running calibration.

Calibration Constant Loading by NI-DAQmx

NI-DAQmx automatically loads calibration constants into calDACs whenever you call functions/VIs that depend on them. The following conditions apply:

- 12-bit E Series devices
    - 12-bit devices use a single set of calibration constants for both unipolar and bipolar modes of analog input.
    - One set of constants is valid for unipolar, and another set is valid for bipolar configuration of the analog output channels. When you change the polarity of an analog output channel, NI-DAQmx reloads the calibration constants for that channel.
- 16-bit E Series devices
    - Calibration constants required by the 16-bit E Series devices for unipolar analog input channels are different from those for bipolar analog input channels. If you are acquiring data from one channel, or if all of the channels you are acquiring data from are configured for the same polarity, NI-DAQmx selects the appropriate set of calibration constants for you. If you are scanning several channels, and you mix channels configured for unipolar and bipolar mode in your scan list, NI-DAQmx loads the calibration constants that correspond to the first channel in the scan list.
    - NI 6025E devices use a single set of calibration constants for both unipolar and bipolar modes of analog input.
    - One set of constants is valid for unipolar, and another set is valid for bipolar configuration of the analog output channels. When you change the polarity of an analog output channel, NI-DAQmx reloads the calibration constants for that channel.

Related concepts:

- Device Calibration Signal Connections for E Series Devices

FieldDAQ Calibration

FieldDAQ devices use software calibration to adjust the software scaling of signals read from and produced by your device. Using a precise reference signal, your device measures and calculates scaling constants for analog input. The scaling constants are stored in nonvolatile memory (EEPROM) on your device. Frequent calibration produces the most stable and repeatable measurement performance. The device is not harmed in any way if you recalibrate it often.

> ✏️ **Note**
>
> - Calibrating your device takes some time. Do not be alarmed if the Adjust FieldDAQ Calibration function/VI takes several seconds to execute.
> - For best results, stop any ongoing tasks and disconnect any unnecessary external connections before running calibration.

FieldDAQ devices do not self-calibrate or automatically calibrate. You must use a manual procedure to calibrate these devices.

FieldDAQ devices include the FD-11601, FD-11603, FD-11605, FD-11613, FD-11614, FD-11634, and FD-11637.

Related information:

- Calibration Procedures

M Series, NI 6010, NI 9205, NI 9206, and TS-15100 Calibration

Your device uses software calibration to adjust the software scaling of signals read from and produced by your device. Using calibration pulse width modulated (PWM) sources with a reference voltage, your device measures and calculates scaling constants for analog input and analog output. The scaling constants are stored in

nonvolatile memory (EEPROM) on your device. NI recommends that you self-calibrate your device just before a measurement session but after your computer and the device have been powered on and warmed up for at least 15 minutes. You should allow this same warm-up time before performing any calibration of your system. Frequent calibration produces the most stable and repeatable measurement performance. The device is not harmed in any way if you recalibrate it often.

> ✏️ **Note**
>
> - Calibrating your device takes some time. Do not be alarmed if the Self-Calibrate or Adjust M Series Calibration function/VI takes several seconds to execute.
> - For best results, stop any ongoing tasks and disconnect any unnecessary external connections before running calibration.

NI 6154 Calibration

Your device uses software calibration to adjust the software scaling of signals read from and produced by your device. Using calibration pulse-width modulated (PWM) sources with a reference voltage, your device measures and calculates scaling constants for analog input and analog output. The scaling constants are stored in nonvolatile memory (EEPROM) on your device. NI recommends that you self-calibrate your device just before a measurement session but after your computer and the device have been powered on and warmed up for at least 15 minutes. You should allow this same warm-up time before performing any calibration of your system. Frequent calibration produces the most stable and repeatable measurement performance. The device is not harmed in any way if you recalibrate it often.

> ✏️ **Note**
>
> - Calibrating your device takes some time. Do not be alarmed if the Self-Calibrate or Adjust S Series Calibration function/VI takes several seconds to execute.

- For best results, stop any ongoing tasks and disconnect any unnecessary external connections before running calibration.

NI 6614 Calibration

Only external calibration, not self-calibration, is supported for NI 6614 devices. You should allow 15 minutes of warm-up time before performing any calibration of your system. Frequent calibration produces the most stable and repeatable measurement performance. The device is not harmed in any way if you recalibrate it often.

Note

- Calibrating your device takes some time. Do not be alarmed if the DAQmx Adjust TIO Timebase Calibration function/VI takes several minutes to execute.
- For best results, stop any ongoing tasks and disconnect any unnecessary external connections before running calibration.

NI PXI-6608 Calibration

You cannot calibrate the PXI-6608 in NI-DAQmx. The device must be calibrated using the Traditional NI-DAQ (Legacy) driver. To use the NI PXI-6608 in NI-DAQmx after calibrating it in Traditional NI-DAQ (Legacy), you must do one of the following:

- Call the Traditional NI-DAQ (Legacy) Device Reset function/VI.

—or—

- Right-click the **Traditional NI-DAQ (Legacy) Devices** folder in MAX and select **Reset Driver for Traditional NI-DAQ**.

S Series Calibration

Your device uses hardware calibration to adjust the analog circuitry. This calibration is done with calibration digital-to-analog converters, called calDACs, that fine-tune the analog circuitry. The calDACs must be programmed (or loaded) with certain numbers called calibration constants. Those constants are stored in nonvolatile memory (EEPROM) on your device or are maintained by NI-DAQmx. NI recommends that you self-calibrate your device just before a measurement session but after your computer and the device have been powered on and warmed up for at least 15 minutes. You should allow this same warm-up time before performing any calibration of your system. Frequent calibration produces the most stable and repeatable measurement performance. The device is not harmed in any way if you recalibrate it often.

> **Note**
>
> - Calibrating your MIO or AI device takes some time. Do not be alarmed if the DAQmx Self-Calibrate or Adjust S Series Calibration function/VI takes several seconds to execute.
> - For best results, stop any ongoing tasks and disconnect any unnecessary external connections before running calibration.

Related concepts:

- [Device Calibration Signal Connections for S Series Devices](#)

SC Express Calibration

The NI 4300, NI 4309, NI 4310, and NI 4340 use software calibration to adjust the software scaling of signals read from and produced by your device. Using onboard calibration pulse-width modulated (PWM) sources and an external reference voltage, your device calculates scaling constants for analog input. The scaling constants are stored in nonvolatile memory (EEPROM) on your device. NI recommends that you self-calibrate your device just before a measurement session but after your computer and the device have been powered on and warmed up for at least 15 minutes. You should allow this same warm-up time before performing any calibration of your system. Frequent calibration produces the most stable and repeatable measurement performance. The device is not harmed in any way if you recalibrate it often.

> **Note**
>
> - Calibrating your device takes some time. Do not be alarmed if the Self-Calibrate or Adjust 4300 Calibration function/VI takes several minutes to execute.
> - For best results, stop any ongoing tasks and disconnect any unnecessary external connections before running calibration.

The NI 4302/4303/4304/4305, NI 4322, NI 433x, NI 4353, and NI 4357 do not self-calibrate or automatically calibrate. You must use a manual procedure to calibrate NI 4302/4303/4304/4305, NI 4322, NI 433x, NI 4353, and NI 4357 devices.

Related information:

- [Calibration Procedures](#)

SCXI-1600 Calibration

The external calibration process for the SCXI-1600 module is nearly identical to the E Series devices. However, when applying a precision voltage to the module, you must connect the signal to the EXTCAL BNC connector on the front of the SCXI-1600, instead of the AI0 analog input channel. In LabVIEW, use the Adjust E-Series Calibration VI, instead of the Adjust SC Baseboard Calibration VI.

X Series Calibration

Your device uses software calibration to adjust the software scaling of signals read from and produced by your device. Using calibration pulse width modulated (PWM) sources with a reference voltage, your device measures and calculates scaling constants for analog input and analog output. The scaling constants are stored in nonvolatile memory (EEPROM) on your device. NI recommends that you self-calibrate your device just before a measurement session but after your computer and the device have been powered on and warmed up for at least 15 minutes. You should allow this same warm-up time before performing any calibration of your system. Frequent calibration produces the most stable and repeatable measurement performance. The

device is not harmed in any way if you recalibrate it often.

> ✏️ **Note**
>
> - Calibrating your device takes some time. Do not be alarmed if the Self-Calibrate or Adjust X Series Calibration function/VI takes several seconds to execute.
> - For best results, stop any ongoing tasks and disconnect any unnecessary external connections before running calibration.

Signal Connections

This section contains information about calibration signal connections for AO Series, E Series, M Series, NI 6010, NI 6154, S Series, and X Series devices.

Related concepts:

- Device Calibration Signal Connections for AO Series Devices
- Device Calibration Signal Connections for E Series Devices
- Device Calibration Signal Connections for M Series and NI 6010 Devices
- Device Calibration Signal Connections for the NI 6154
- Device Calibration Signal Connections for NI 6614
- Device Calibration Signal Connections for S Series Devices
- Device Calibration Signal Connections for X Series Devices

Device Calibration Signal Connections for AO Series Devices

When you self-calibrate your AO Series device, no signal connections are necessary. However, values generated on the analog output channels change during the calibration process.

When externally calibrating your AO Series device, connect the signals as described below for the type of AO Series device you are calibrating. Set the reference voltage between +6.000 V and +9.999 V. Typically, you use a calibrator or other stable voltage source for the reference voltage. Do not use a power supply as its signals are not very

stable.

Follow these steps for AO Series devices:

1. Connect the positive output of your reference voltage source to the EXT REF terminal.
2. Connect the negative output of your reference voltage source to the AO GND terminal.

Related information:

- [Calibration Procedures](#)

Device Calibration Signal Connections for E Series Devices

When you self-calibrate your E Series device, no signal connections are necessary. However, values generated on the analog output channels change during the calibration process. If you have external circuitry connected to the analog output channels and you do not want changes on these channels, you should disconnect the circuitry before beginning the self-calibration.

When externally calibrating your E Series device, connect the signals as described below for the type of E Series device you are calibrating. Set the reference voltage between +6.000 V and +9.999 V. Typically, you use a calibrator or other stable voltage source for the reference voltage. Do not use a power supply as its signals are not very stable.

- 12-Bit E Series Devices

  Follow these steps for 12-bit E Series devices:

  1. Connect the positive output of your reference voltage source to physical channel ai8.
  2. Connect the negative output of your reference voltage source to the AI SENSE terminal.
  3. Connect physical channel ao0 to physical channel ai0.
  4. If your reference voltage source and your computer are floating with respect to each other, connect the AI SENSE terminal to the AI GND terminal as well as to

the negative output of your reference voltage source.

- 16-Bit E Series Devices

  Follow these steps for 16-bit E Series devices:

  1. Connect the positive output of your reference voltage source to physical channel ai0.
  2. Connect the negative output of your reference voltage source to physical channel ai8.
  3. If your reference voltage source and your computer are floating with respect to each other, connect the negative output of your reference voltage source to the AI GND terminal as well as to physical channel ai8.

Device Calibration Signal Connections for S Series Devices

When you self-calibrate your S Series device, no signal connections are necessary. However, values generated on the analog output channels change during the calibration process. If you have external circuitry connected to the analog output channels that is sensitive to these changes, you should disconnect the circuitry before beginning self-calibration.

When externally calibrating your S Series device, connect the signals as described below. Set the reference voltage to the following:

- NI PCI/PXI 6143: between 3.0 V and 4.998 V
- NI PCI/PXI 6115: between 4.995 V and 5.005 V
- NI PCI/PXI 6120: between 4.995 V and 5.005 V
- All other S Series Devices: between 6.0 V and 9.998 V

Typically, you should use a calibrator or other stable voltage source for calibration. Do not use a power supply as its signals are not very stable.

For external calibration, make the following signal connections:

1. Connect the positive output of your reference voltage source to ACH0+.
2. Connect the negative output of your reference voltage source to ACH0-.

Device Calibration Signal Connections for M Series and NI 6010 Devices

When you self-calibrate your M Series or NI 6010 device, no signal connections are necessary. However, values generated on the analog output channels change during the calibration process. If you have external circuitry connected to the analog output channels and you do not want changes on these channels, you should disconnect the circuitry before beginning the self-calibration.

When externally calibrating your M Series or NI 6010 device, connect the signals as described below for the type of device you are calibrating. Set the reference voltage between +6.0 V and +8.5 V for M Series devices, between +3.5 V and 4.0 V for NI 6010 devices. Typically, you use a calibrator or other stable voltage source for the reference voltage. Do not use a power supply as its signals are not very stable.

Follow these steps:

1. Disconnect any external connections or circuitry to your device.
2. Connect the positive output of your reference voltage source to physical channel ai0.
3. Connect the negative output of your reference voltage source to physical channel ai8.
4. If your reference voltage source and your computer are floating with respect to each other, connect the negative output of your reference voltage source to the AI GND terminal as well as to physical channel ai8.

Device Calibration Signal Connections for the NI 6154

When you self-calibrate your NI 6154 device, no signal connections are necessary. However, values generated on the analog output channels change during the calibration process.

When externally calibrating your NI 6154 device, connect the signals as described below for the type of device you are calibrating. Set the reference voltage between +6.0 V and +9.998 V. Typically, you use a calibrator or other stable voltage source for the reference voltage. Do not use a power supply as its signals are not very stable.

Follow these steps:

1. Disconnect any external connections or circuitry to your device.
2. Connect the positive output of your reference voltage source to physical channel AI0+.
3. Connect the negative output of your reference voltage source to physical channel AI0-.

Device Calibration Signal Connections for NI 6614

For signal connections needed to externally calibrate your NI 6614, refer to the calibration procedure for the NI 6614.

Device Calibration Signal Connections for X Series Devices

When you self-calibrate your X Series device, no signal connections are necessary. However, values generated on the analog output channels change during the calibration process. If you have external circuitry connected to the analog output channels and you do not want changes on these channels, you should disconnect the circuitry before beginning the self-calibration.

When externally calibrating (or adjusting) your X Series device, connect the signals as described below for the type of device you are calibrating. Set the reference voltage between +6.0 V and +8.5 V for X Series devices. You should use a calibrator or other stable voltage source for the reference voltage. Do not use a power supply as its signals are not very stable.

Follow these steps:

1. Disconnect any external connections or circuitry to your device.
2. Connect the positive output of your reference voltage source to physical channel ai0.
3. Connect the negative output of your reference voltage source to physical channel ai8.
4. If your reference voltage source and your computer are floating with respect to each other, connect the negative output of your reference voltage source to the AI GND terminal as well as to physical channel ai8.

Counters

This section contains information on counter signal connections and routing diagrams that illustrate the internal counter routing.

Averaging Support

The following devices support averaging digital frequency and period measurements:

- Parallel digital I/O modules connected to CompactDAQ chassis[1], TestScale chassis[2], CompactDAQ controllers[3], CompactRIO controllers[4] and CompactRIO Single-Board controllers[5]. Refer to ***Digital I/O Considerations for C Series Devices***.
- NI 661x devices
- X Series devices

Related concepts:

- [Digital I/O Considerations for C Series and TestScale Devices](#)
- [Configuring a Time-Based Measurement in NI-DAQmx](#)

C Series Counter Modules

The NI 9361 counter input module supports the following measurement types:

- Duty cycle
- Event counting
- Encoder position
- Frequency
- Period
- Pulse width
- Velocity

1. cDAQ-9171, 9174, 9178, 9179, 9181, 9184, 9185, 9188, 9188XT, 9189, and 9191
2. TS-15000 and TS-15010
3. cDAQ-9132, 9133, 9134, 9135, 9136, and 9137
4. cRIO-9040, 9041, 9042, 9043, 9045, 9046, 9047, 9048, 9049, 9053, 9054, 9055, 9056, 9057, and 9058
5. sbRIO-9603, 9608, 9609, 9628, 9629, and 9638

The NI 9361 has eight counters and supports adding multiple counter channels to the same task. A single task can have different measurement types. Multiple counter module channels can be added to the same task.

> ✎ **Note**
>
> - You cannot mix module and chassis counters in the same task.
> - You cannot use the chassis counter through the counter module.
> - You cannot create multiple counter tasks with a single NI 9361.
> - You cannot have more than two NI 9361 with different timing/triggering configuration in a single chassis.

Connecting Counter Signals

The default terminals used for counter measurements and generations vary from device to device. Follow the links below for information specific to your device. To override the default input terminal, set the DAQmx Channel Input Terminal attribute/property for the measurement type. For instance, if you are counting edges, you would use CI.CountEdges.Term. To override the default output terminal, set the DAQmx Channel Output Terminal attribute/property to the desired value.

Related reference:

- [AO Series, E Series, and S Series Signal Connections for Counters](#)
- [Bus-Powered M Series Signal Connections for Counters](#)
- [C Series and TestScale Module Signal Connections for Counters](#)
- [myDAQ Signal Connections for Counters](#)
- [NI ELVIS II Family Signal Connections for Counters](#)
- [TIO Signal Connections for Counters](#)
- [X Series Signal Connections for Counters](#)
- [37-Pin DSUB Signal Connections for Counters](#)
- [68-Pin M Series Signal Connections for Counters](#)
- [C Series Signal Connections for Counters](#)

Bus-Powered M Series Signal Connections for Counters

The following table lists the default input terminals for various counter measurements on bus-powered M Series devices. You can use a different PFI line for any of the input terminals. To change the PFI input for a measurement, use the NI-DAQmx channel attributes/properties.

16-PFI Line Devices (NI 6218)

Count Edges	Edges: PFI 0 Count Direction: PFI 9	Edges: PFI 3 Count Direction: PFI 10
Pulse Width Measurement	PFI 1	PFI 2
Period/Frequency Measurement (Low Frequency with One Counter)	PFI 1	PFI 2
Period/Frequency Measurement (High Frequency with Two Counters)	PFI 0	PFI 3
Period/Frequency Measurement (Large Range with Two Counters)	PFI 0	PFI 3
Semiperiod Measurement	PFI 1	PFI 2
Two-Edge Separation Measurement	• Start: PFI 9 • Stop: PFI 1	• Start: PFI 10 • Stop: PFI 2
Position Measurement	• A: PFI 0 • B: PFI • Z: PFI 1	• A: PFI 3 • B: PFI 10 • Z: PFI 2

The following table lists the output terminals for counter output. You can use a different PFI line for any of the output terminals.

Ctr0	Ctr1
PFI 4	PFI 5

16-PFI Line Devices (NI 6212/6216)

Measurement	Ctr0	Ctr1
Count Edges	• Edges: PFI 8 • Count Direction: PFI 10	• Edges: PFI 3 • Count Direction: PFI 11
Pulse Width Measurement	PFI 9	PFI 4
Period/Frequency Measurement (Low Frequency with One Counter)	PFI 9	PFI 4
Period/Frequency Measurement (High Frequency with Two Counters)	PFI 8	PFI 3
Period/Frequency Measurement (Large Range with Two Counters)	PFI 8	PFI 3
Semiperiod Measurement	PFI 9	PFI 4
Two-Edge Separation Measurement	• Start: PFI 10 • Stop: PFI 9	• Start: PFI 11 • Stop: PFI 4
Position Measurement	• A: PFI 8 • B: PFI 10 • Z: PFI 9	• A: PFI 3 • B: PFI 11 • Z: PFI 4

The following table lists the output terminals for counter output. You can use a different PFI line for any of the output terminals.

Ctr0	Ctr1
PFI 12	PFI 13

8-PFI Line Devices (Such as the NI 6210/6211/6215)

Measurement	Ctr0	Ctr1
Count Edges	• Edges: PFI 0 • Count Direction: PFI 0	• Edges: PFI 3 • Count Direction: PFI 3
Pulse Width Measurement	PFI 1	PFI 2
Period/Frequency Measurement (Low Frequency with One Counter)	PFI 1	PFI 2
Period/Frequency Measurement (High Frequency with Two Counters)	PFI 0	PFI 3
Period/Frequency Measurement (Large Range with Two Counters)	PFI 0	PFI 3
Semiperiod Measurement	PFI 1	PFI 2
Two-Edge Separation Measurement	• Start: PFI 0 • Stop: PFI 1	• Start: PFI 3 • Stop: PFI 2
Position Measurement	• A: PFI 0 • B: PFI 1 • Z: PFI 2	• A: PFI 3 • B: PFI 2 • Z: PFI 1

The following table lists the output terminals for counter output. You can use a different PFI line for any of the output terminals.

Ctr0	Ctr1
PFI 4	PFI 5

Related concepts:

- Self-Powered Compared to Bus-Powered USB Devices
- Self-Powered Versus Bus-Powered M Series USB Devices

C Series and TestScale Module Signal Connections for Counters

The following tables lists the default input terminals for various counter measurements. You can use a different PFI line for any of the input terminals. To change the PFI input for a measurement, use the NI-DAQmx channel attributes/ properties.

For information about counters on CompactRIO Single-Board controllers, refer to the Counter input and Output section in **CompactRIO Single-Board Controller Physical Channels**.

NI 9401, NI 9421, NI 9422, NI 9423, NI 9436, NI 9437 (8-Channel), and TS-15050 DIO P0

Measurement	Ctr0	Ctr1	Ctr2	Ctr3
Count Edges	• Edges: PFI 0 • Count Direction: PFI 2	• Edges: PFI 4 • Count Direction: PFI 6	• Edges: PFI 3 • Count Direction: PFI 0	• Edges: PFI 7 • Count Direction: PFI 4
Pulse Width Measurement	PFI 1	PFI 5	PFI 2	PFI 6
Period/Frequency Measurement (Low Frequency with One Counter)	PFI 1	PFI 5	PFI 2	PFI 6
Period/Frequency Measurement (High Frequency with Two Counters)	PFI 0	PFI 4	PFI 3	PFI 7
Period/Frequency Measurement (Large Range with Two Counters)	PFI 0	PFI 4	PFI 3	PFI 7
Semiperiod Measurement	PFI 1	PFI 5	PFI 2	PFI 6
Two-Edge Separation Measurement	• Start: PFI 2 • Stop: PFI 1	• Start: PFI 6 • Stop: PFI 5	• Start: PFI 0 • Stop: PFI 2	• Start: PFI 4 • Stop: PFI 6

Measurement	Ctr0	Ctr1	Ctr2	Ctr3
Position Measurement	• A: PFI 0 • B: PFI 2 • Z: PFI 1	• A: PFI 4 • B: PFI 6 • Z: PFI 5	• A: PFI 3 • B: PFI 0 • Z: PFI 2	• A: PFI 7 • B: PFI 4 • Z: PFI 6

NI 9402 and NI 9435 (4-Channel)

Measurement	Ctr0	Ctr1	Ctr2	Ctr3
Count Edges	• Edges: PFI 0 • Count Direction: PFI 2	• Edges: PFI 3 • Count Direction: PFI 1	• Edges: PFI 1 • Count Direction: PFI 0	• Edges: PFI 2 • Count Direction: PFI 3
Pulse Width Measurement	PFI 1	PFI 2	PFI 3	PFI 0
Period/Frequency Measurement (Low Frequency with One Counter)	PFI 1	PFI 2	PFI 3	PFI 0
Period/Frequency Measurement (High Frequency with Two Counters)	PFI 0	PFI 3	PFI 1	PFI 2
Period/Frequency Measurement (Large Range with Two Counters)	PFI 0	PFI 3	PFI 1	PFI 2
Semiperiod Measurement	PFI 1	PFI 2	PFI 3	PFI 0
Two-Edge Separation Measurement	• Start: PFI 2 • Stop: PFI 1	• Start: PFI 1 • Stop: PFI 2	• Start: PFI 0 • Stop: PFI 3	• Start: PFI 3 • Stop: PFI 0
Position Measurement	• A: PFI 0 • B: PFI 2 • Z: PFI 1	• A: PFI 3 • B: PFI 1 • Z: PFI 2	• A: PFI 1 • B: PFI 0 • Z: PFI 3	• A: PFI 2 • B: PFI 3 • Z: PFI 0

NI 9411 (6-Channel)

Measurement	Ctr0	Ctr1	Ctr2	Ctr3
Count Edges	• Edges: PFI 0 • Count Direction: PFI 2	• Edges: PFI 3 • Count Direction: PFI 5	• Edges: PFI 2 • Count Direction: PFI 1	• Edges: PFI 5 • Count Direction: PFI 4
Pulse Width Measurement	PFI 1	PFI 4	PFI 0	PFI 3
Period/Frequency Measurement (Low Frequency with One Counter)	PFI 1	PFI 4	PFI 0	PFI 3
Period/Frequency Measurement (High Frequency with Two Counters)	PFI 0	PFI 3	PFI 2	PFI 5
Period/Frequency Measurement (Large Range with Two Counters)	PFI 0	PFI 3	PFI 2	PFI 5
Semiperiod Measurement	PFI 1	PFI 4	PFI 0	PFI 3
Two-Edge Separation Measurement	• Start: PFI 2 • Stop: PFI 1	• Start: PFI 5 • Stop: PFI 4	• Start: PFI 1 • Stop: PFI 0	• Start: PFI 4 • Stop: PFI 3
Position Measurement	• A: PFI 0 • B: PFI 2 • Z: PFI 1	• A: PFI 3 • B: PFI 5 • Z: PFI 4	• A: PFI 2 • B: PFI 1 • Z: PFI 0	• A: PFI 5 • B: PFI 4 • Z: PFI 3

The following tables list the output terminals for counter output. You can use a different PFI line for any of the output terminals.

NI 9401, NI 9472, NI 9474, NI 9475, NI 9485 (8-Channel), and TS-15050 DIO P0

Ctr0	Ctr1	Ctr2	Ctr3	FreqOut
PFI 3	PFI 7	PFI 1	PFI 5	PFI 2

NI 9481, NI 9482, and NI 9402 (4-Channel)

Ctr0	Ctr1	Ctr2	Ctr3	FreqOut
PFI 0	PFI 3	PFI 1	PFI 2	PFI 1

NI 9361 (8-Counters)

Measurement	Ctr0	Ctr1	Ctr2	Ctr3	Ctr4	Ctr5
Count Edges	• Edges: PFI 0 • Count Direction: PFI 7 • Reset: PFI4	• Edges: PFI 1 • Count Direction: PFI 6 • Reset: PFI5	• Edges: PFI 2 • Count Direction: PFI 5 • Reset: PFI6	• Edges: PFI 3 • Count Direction: PFI 4 • Reset: PFI7	• Edges: PFI 4 • Count Direction: PFI 3 • Reset: PFI0	• Edges: PFI 5 • Count Direction: PFI 2 • Reset: PFI1
Duty Cycle Measurement	PFI 0	PFI 1	PFI 2	PFI 3	PFI 4	PFI 5
Pulse Width Measurement	PFI 0	PFI 1	PFI 2	PFI 3	PFI 4	PFI 5
Period/ Frequency Measurement (Dynamic Averaging)	PFI 0	PFI 1	PFI 2	PFI 3	PFI 4	PFI 5
Position Measurement	• A: PFI 0 • B: PFI 4 • Z: PFI 7	• A: PFI 1 • B: PFI 5 • Z: PFI 6	• A: PFI 2 • B: PFI 6 • Z: PFI 5	• A: PFI 3 • B: PFI 7 • Z: PFI 4	• A: PFI 4 • B: PFI 0 • Z: PFI 3	• A: PFI 5 • B: PFI 1 • Z: PFI 2

Measurement	Ctr0	Ctr1	Ctr2	Ctr3	Ctr4	Ctr5
Velocity	• A: PFI 0 • B: PFI 4	• A: PFI 1 • B: PFI 5	• A: PFI 2 • B: PFI 6	• A: PFI 3 • B: PFI 7	• A: PFI 4 • B: PFI 0	• A: PFI 5 • B: PFI 1

Related concepts:

- [CompactRIO Single-Board Controller Physical Channels](#)

AO Series, E Series, and S Series Signal Connections for Counters

The following table lists the default input terminals for various counter measurements. You can use a different PFI line for any of the input terminals, with the exception of the count direction terminal for edge counting. To change the PFI input for a measurement, use the NI-DAQmx channel attributes/properties.

Measurement	Ctr0	Ctr1
Count Edges	• Edges: PFI 8 • Count Direction[1]6: port0/line6	• Edges: PFI 3 • Count Direction[1]: port0/line7
Pulse Width Measurement	PFI 9	PFI 4
Period/Frequency Measurement (Low Frequency with One Counter)	PFI 9	PFI 4
Period/Frequency Measurement (High Frequency with Two Counters)	PFI 8	PFI 3
Period/Frequency Measurement (Large Range with Two Counters)	PFI 8	PFI 3
Semiperiod Measurement	PFI 9	PFI 4

The following table lists the default output terminals for counter output. You must use the default output terminal, with the exception that for Ctr0, you can select a RTSI line.

6. The count direction terminal must be tristated to use an external signal. Reset the device to ensure the terminal is tristated

Ctr0	Ctr1
CTR 0 OUT	CTR 1 OUT

myDAQ Signal Connections for Counters

The following table lists the default input terminals for various counter measurements on myDAQ. You can use a different PFI line for any of the input terminals. To change the PFI input for a measurement, use the NI-DAQmx channel attributes/properties.

Measurement	Ctr0
Count Edges	• Edges: PFI 0 • Count Direction: PFI 2
Pulse Width Measurement	PFI 1
Period/Frequency Measurement (Low Frequency with One Counter)	PFI 1
Pulse Measurement	PFI 1
Semiperiod Measurement	PFI 1
Two-Edge Separation Measurement	• Start: PFI 2 • Stop: PFI 1
Position Measurement	• A: PFI 0 • B: PFI 2 • Z: PFI 1

The following table lists the output terminals for counter output. You can use a different PFI line for the output terminal.

Ctr0
PFI 3

NI ELVIS II Family Signal Connections for Counters

The following table lists the default input terminals for various counter measurements. You can use a different PFI line for any of the input terminals. To change the PFI input for a measurement, use the NI-DAQmx channel attributes/properties.

Measurement	Ctr0	Ctr1
Count Edges	• Edges: PFI 8 • Count Direction: PFI 10	• Edges: PFI 3 • Count Direction: PFI 11
Pulse Width Measurement	PFI 9	PFI 4
Period/Frequency Measurement (Low Frequency with One Counter)	PFI 9	PFI 4
Period/Frequency Measurement (High Frequency with Two Counters)	PFI 8	PFI 3
Period/Frequency Measurement (Large Range with Two Counters)	PFI 8	PFI 3
Semiperiod Measurement	PFI 9	PFI 4
Two-Edge Separation Measurement	• Start: PFI 10 • Stop: PFI 9	• Start: PFI 11 • Stop: PFI 4
Position Measurement	• A: PFI 8 • B: PFI 10 • Z: PFI 9	• A: PFI 3 • B: PFI 11 • Z: PFI 4

The following table lists the output terminals for counter output. You can use a different PFI line for any of the output terminals.

Ctr0	Ctr1
PFI 12	PFI 13

NI mioDAQ Signal Connections for Counters

The following table lists the default input terminals for various counter measurements on mioDAQ devices. You can use a different PFI line for any of the input terminals. To change the PFI input for a measurement, use the NI-DAQmx channel attributes/ properties.

Measurement	Ctr0	Ctr1	Ctr2	Ctr3
Count Edges	• Edges: PFI 0 • Count Direction: PFI 4	• Edges: PFI 1 • Count Direction: PFI 5	• Edges: PFI 2 • Count Direction: PFI 6	• Edges: PFI 3 • Count Direction: PFI 7
Pulse Width Measurement	PFI 8	PFI 9	PFI 10	PFI 11
Period/Frequency Measurement (Low Frequency with One Counter)	PFI 8	PFI 9	PFI 10	PFI 11
Period/Frequency Measurement (High Frequency with Two Counters)	PFI 0	PFI 1	PFI 2	PFI 3
Period/Frequency Measurement (Large Range with Two Counters)	PFI 0	PFI 1	PFI 2	PFI 3
Pulse Measurement	PFI 8	PFI 9	PFI 10	PFI 11
Semiperiod Measurement	PFI 8	PFI 9	PFI 10	PFI 11
Two-Edge Separation Measurement	• Start: PFI 4 • Stop: PFI 8	• Start: PFI 5 • Stop: PFI 9	• Start: PFI 6 • Stop: PFI 10	• Start: PFI 7 • Stop: PFI 11
Position Measurement	• A: PFI 0 • B: PFI 4 • Z: PFI 8	• A: PFI 1 • B: PFI 5 • Z: PFI 9	• A: PFI 2 • B: PFI 6 • Z: PFI 10	• A: PFI 3 • B: PFI 7 • Z: PFI 11

The following table lists the output terminals for counter output. You can use a different PFI line for any of the output terminals.

Ctr0	Ctr1	Ctr2	Ctr3	FreqOut
PFI 0	PFI 1	PFI 2	PFI 3	PFI 15

TIO Signal Connections for Counters

The following table lists the default input terminals for various counter measurements. You can use a different PFI line for any of the input terminals. To change the PFI input for a measurement, use the NI-DAQmx channel properties/attributes.

> **Note** You cannot change the default PFI lines for quadrature encoder-based position measurements on NI 660x devices.

Measurement	Ctr0	Ctr1	Ctr2	Ctr3	Ctr4	Ctr5	Ctr6
Count Edges	• Edges: PFI 39 • Count Direction: PFI 37	• Edges: PFI 35 • Count Direction: PFI 33	• Edges: PFI 31 • Count Direction: PFI 29	• Edges: PFI 27 • Count Direction: PFI 25	• Edges: PFI 23 • Count Direction: PFI 21	• Edges: PFI 19 • Count Direction: PFI 17	• Edges: PFI 15 • Count Directi... PFI 13
Pulse Width Measurement	PFI 38	PFI 34	PFI 30	PFI 26	PFI 22	PFI 18	PFI 14
Period/ Frequency Measurement (Low Frequency with One Counter)	PFI 38	PFI 34	PFI 30	PFI 26	PFI 22	PFI 18	PFI 14
Period/ Frequency Measurement (High Frequency with Two Counters)	PFI 39	PFI 35	PFI 31	PFI 27	PFI 23	PFI 19	PFI 15
Period/ Frequency	PFI 39	PFI 35	PFI 31	PFI 27	PFI 23	PFI 19	PFI 15

Measurement	Ctr0	Ctr1	Ctr2	Ctr3	Ctr4	Ctr5	Ctr6
Measurement (Large Range with Two Counters)							
Semiperiod Measurement	PFI 38	PFI 34	PFI 30	PFI 26	PFI 22	PFI 18	PFI 14
Two-Edge Separation Measurement	• Start: PFI 37 • Stop: PFI 38	• Start: PFI 33 • Stop: PFI 34	• Start: PFI 29 • Stop: PFI 30	• Start: PFI 25 • Stop: PFI 26	• Start: PFI 21 • Stop: PFI 22	• Start: PFI 17 • Stop: PFI 18	• Start: 13 • Stop: 14
Position Measurement	• A: PFI 39 • B: PFI 37 • Z: PFI 38	• A: PFI 35 • B: PFI 33 • Z: PFI 34	• A: PFI 31 • B: PFI 29 • Z: PFI 30	• A: PFI 27 • B: PFI 25 • Z: PFI 26	• A: PFI 23 • B: PFI 21 • Z: PFI 22	• A: PFI 19 • B: PFI 17 • Z: PFI 18	• A: PFI • B: PFI • Z: PFI
GPS Timestamp Measurement	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Note The NI 6601 has only four counters (ctr0-ctr3). The entries in the previous table for cntr4, cntr5, cntr6, and cntr7 do not apply for that device.

The following table lists the output terminals for counter output. You can use a different PFI line for any of the output terminals.

Ctr0	Ctr1	Ctr2	Ctr3	Ctr4	Ctr5	Ctr6	Ctr7
PFI 36	PFI 32	PFI 28	PFI 24	PFI 20	PFI 16	PFI 12	PFI 8

Note The NI 6601 has only four counters (ctr0-ctr3). The entries in the previous table for cntr4, cntr5, cntr6, and cntr7 do not apply for that device.

NI 6624 Issues

The eight PFI lines listed as the defaults for counter output are dedicated for output, and they are the only terminals you can use for counter output. For example, you can use PFI 8 (the default for Ctr7) as the output terminal for any counter, but you cannot use it as an input terminal.

When using counter output, if the Idle State attribute/property is low, the optocouplers on the NI 6624 will still be driving your output load. Set the Idle State attribute/property to high to prevent driving the output after your task completes.

X Series Signal Connections for Counters

The following table lists the default input terminals for various counter measurements on X Series devices. You can use a different PFI line for any of the input terminals. To change the PFI input for a measurement, use the NI-DAQmx channel attributes/properties.

Measurement	Ctr0	Ctr1	Ctr2	Ctr3
Count Edges	• Edges: PFI 8 • Count Direction: PFI 10	• Edges: PFI 3 • Count Direction: PFI 11	• Edges: PFI 0 • Count Direction: PFI 2	• Edges: PFI 5 • Count Direction: PFI 7
Pulse Width Measurement	PFI 9	PFI 4	PFI 1	PFI 6
Period/Frequency Measurement (Low Frequency with One Counter)	PFI 9	PFI 4	PFI 1	PFI 6
Period/Frequency Measurement (High Frequency with Two Counters)	PFI 8	PFI 3	PFI 0	PFI 5
Period/Frequency Measurement (Large Range with Two Counters)	PFI 8	PFI 3	PFI 0	PFI 5
Pulse Measurement	PFI 9	PFI 4	PFI 1	PFI 6
Semiperiod Measurement	PFI 9	PFI 4	PFI 1	PFI 6

Measurement	Ctr0	Ctr1	Ctr2	Ctr3
Two-Edge Separation Measurement	• Start: PFI 10 • Stop: PFI 9	• Start: PFI 11 • Stop: PFI 4	• Start: PFI 2 • Stop: PFI 1	• Start: PFI 7 • Stop: PFI 6
Position Measurement	• A: PFI 8 • B: PFI 10 • Z: PFI 9	• A: PFI 3 • B: PFI 11 • Z: PFI 4	• A: PFI 0 • B: PFI 2 • Z: PFI 1	• A: PFI 5 • B: PFI 7 • Z: PFI 6

The following table lists the output terminals for counter output. You can use a different PFI line for any of the output terminals.

Ctr0	Ctr1	Ctr2	Ctr3
PFI 12	PFI 13	PFI 14	PFI 15

37-Pin DSUB Signal Connections for Counters

The following table lists the default input terminals for various counter measurements for devices that use the 37-Pin DSUB connector such as the NI 6010, NI 6154, and NI 623x. You can use a different PFI line for any of the input terminals. To change the PFI input for a measurement, use the NI-DAQmx channel attributes/properties.

Measurement	Ctr0	Ctr1
Count Edges	• Edges: PFI 0 • Count Direction: PFI 2	• Edges: PFI 3 • Count Direction: PFI 5
Pulse Width Measurement	PFI 1	PFI 4
Period/Frequency Measurement (Low Frequency with One Counter)	PFI 1	PFI 4
Period/Frequency Measurement (High Frequency with	PFI 0	PFI 3

Measurement	Ctr0	Ctr1
Two Counters)		
Period/Frequency Measurement (Large Range with Two Counters)	PFI 0	PFI 3
Semiperiod Measurement	PFI 1	PFI 4
Two-Edge Separation Measurement	• Start: PFI 2 • Stop: PFI 1	• Start: PFI 5 • Stop: PFI 4

The following table lists the output terminals for counter output. You can use a different PFI line for any of the output terminals.

Ctr0	Ctr1
PFI 6	PFI 7

68-Pin M Series Signal Connections for Counters

The following table lists the default input terminals for various counter measurements on M Series devices, including M Series USB devices, such as the NI USB 6259 screw terminal and NI USB-6229 BNC devices. You can use a different PFI line for any of the input terminals. To change the PFI input for a measurement, use the NI-DAQmx channel attributes/properties.

Measurement	Ctr0	Ctr1
Count Edges	• Edges: PFI 8 • Count Direction: PFI 10	• Edges: PFI 3 • Count Direction: PFI 11
Pulse Width Measurement	PFI 9	PFI 4
Period/Frequency Measurement (Low Frequency with One Counter)	PFI 9	PFI 4
Period/Frequency Measurement (High Frequency	PFI 8	PFI 3

Measurement	Ctr0	Ctr1
with Two Counters)		
Period/Frequency Measurement (Large Range with Two Counters)	PFI 8	PFI 3
Semiperiod Measurement	PFI 9	PFI 4
Two-Edge Separation Measurement	• Start: PFI 10 • Stop: PFI 9	• Start: PFI 11 • Stop: PFI 4
Position Measurement	• A: PFI 8 • B: PFI 10 • Z: PFI 9	• A: PFI 3 • B: PFI 11 • Z: PFI 4

The following table lists the output terminals for counter output. You can use a different PFI line for any of the output terminals.

Ctr0	Ctr1
PFI 12	PFI 13

Some M Series devices, including the NI 6010, NI 6154, NI 6221 (37-pin), and NI 623x, use the 37-pin DSUB connector. These devices have different counter terminal defaults. Refer to the ***37-Pin DSUB Signal Connections for Counters*** for the default input terminals on these devices. Bus-powered M Series devices, such as the NI USB-621x devices, also have different counter terminal defaults. Refer to the ***Bus-Powered M Series Signal Connections for Counters*** for the default input terminals on these devices.

Related reference:

- 37-Pin DSUB Signal Connections for Counters
- Bus-Powered M Series Signal Connections for Counters

Counter Internal Routing Diagrams
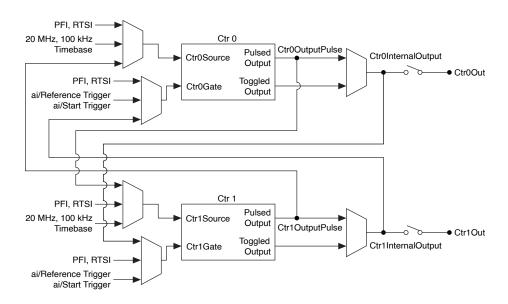
This section contains routing diagrams that illustrate the internal counter routing for AO Series, C Series with CompactDAQ chassis, E Series, M Series, S Series, TIO, X Series, and TestScale devices.

Related concepts:

- AO Series, E Series, S Series Counter Internal Routing Diagram
- Counter Internal Routing Diagrams for C Series Devices with NI cDAQ-91xx Chassis and TestScale Modules with TestScale Chassis
- NI 661x Counter Internal Routing Diagram
- X Series Counter Internal Routing Diagram
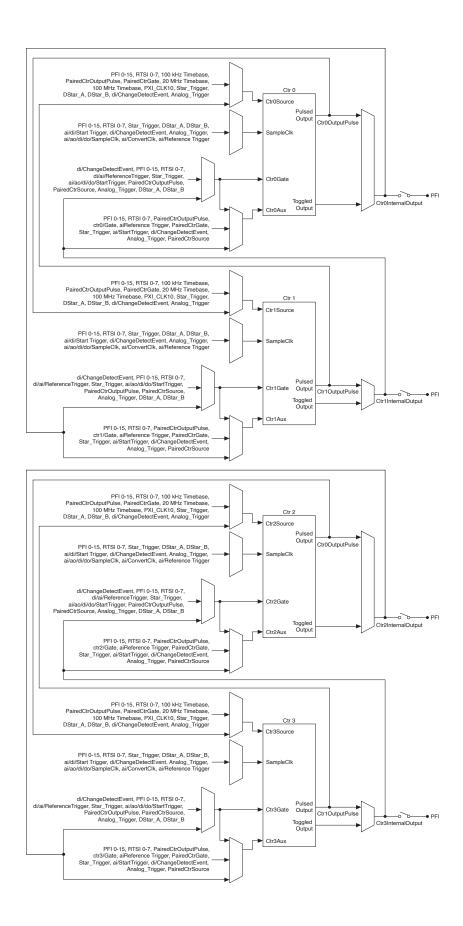
AO Series, E Series, S Series Counter Internal Routing Diagram

The following figure shows the internal routing for DAQ devices with the STC counter/timer such as E Series devices. The black circles represent terminals.

X Series Counter Internal Routing Diagram

The following figure shows the internal routing for DAQ devices with the STC III counter/timer such as X Series devices. The black circles represent terminals.
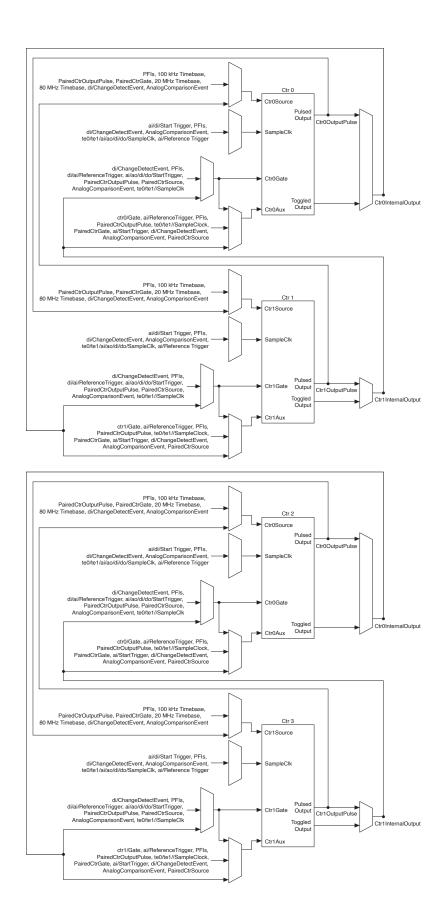
Counter Internal Routing Diagrams for C Series Devices with NI cDAQ-91*xx* Chassis and TestScale Modules with TestScale Chassis

The following figure shows the internal routing for C Series devices used with a NI cDAQ-91xx[7] or TestScale modules connected to a TestScale chassis[8]. The figure shows direct routes available. Indirect routes are available as well, but use extra internal resources. The black circles represent terminals.

---
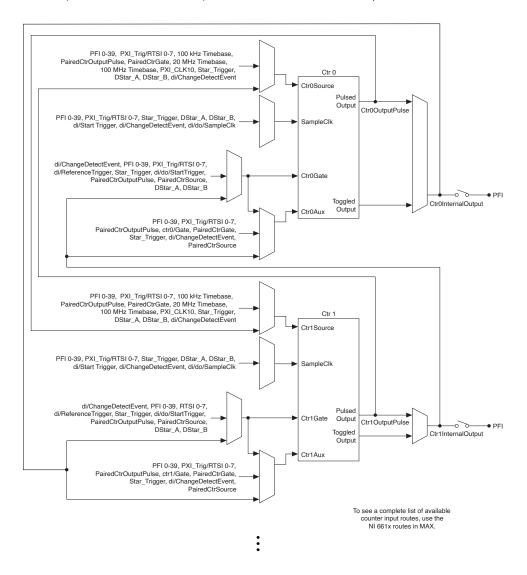
7. cDAQ-9132, 9133, 9134, 9135, 9136, 9137, 9171, 9174, 9178, 9179, 9181, 9184, 9185, 9188, 9188XT, 9189, and 9191
8. TS-15000 and TS-15010

NI 661*x* Counter Internal Routing Diagram

The following figure shows the internal routing for NI 661x devices. The black circles represent terminals. These devices have eight counters which are paired together. Only counters 0 and 1 are shown in the illustration, but the remaining pairs (counters 2 and 3, counters 4 and 5, and counters 6 and 7) are routed identically.

To see a complete list of available counter input routes, use the NI 661x routes in MAX.

Counter Input Error Reporting with C Series, M Series USB, and NI ELVIS II Devices

With C Series, NI ELVIS II Family, and M Series USB devices (except bus-powered M Series devices), buffered counter input error reporting occurs every 128 samples for high-speed USB and every 16 samples for full-speed USB. When an error is detected,

the task stops. To prevent the acquisition of incorrect samples, only data within either the 128- or 16-sample group is reported. For instance, if you attempt to acquire 256 samples, and an error occurs after sample 250, the task returns 128 samples instead of 249. If you attempt to acquire a number of samples less than or equal to 128 or 16 and an error occurs, the task returns no samples.

Related concepts:
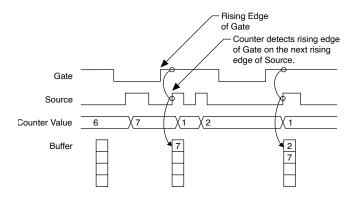
- [Self-Powered Compared to Bus-Powered USB Devices](#)
- [Self-Powered Versus Bus-Powered M Series USB Devices](#)

Duplicate Count Prevention

Duplicate count prevention (or synchronous counting mode) ensures that a counter returns correct data in applications that use a slow or non-periodic external source. Duplicate count prevention applies to any counter application such as measuring frequency or period. In such applications, the counter should store the number of times an external Source pulses between rising edges on the Gate signal.

Example Application That Works Correctly (No Duplicate Counting)

The following figure shows a buffered period measurement that uses an external signal as the Source.
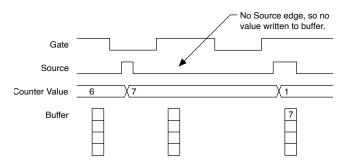
On the first rising edge of the Gate, the current count of 7 is stored. On the next rising edge of the Gate, the counter stores a 2 since two Source pulses occurred after the previous rising edge of Gate.

The counter synchronizes or samples the Gate signal with the Source signal. So the counter does not detect a rising edge in the Gate until the next Source pulse. In this example, the counter stores the values in the buffer on the first rising Source edge after the rising edge of Gate.
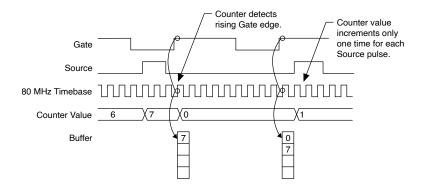
Example Application That Works Incorrectly (Duplicate Counting)

In the following figure, after the first rising edge of Gate, no Source pulses occur. So the counter does not write the correct data to the buffer.

Example Application That Prevents Duplicate Counting

With duplicate count prevention enabled, the counter synchronizes both the Source and Gate signals to the maximum timebase. By synchronizing to the timebase, the counter detects edges on the Gate even if the Source does not pulse. This enables the correct current count to be stored in the buffer even if no Source edges occur in between Gate signals. Refer to the following example.

Even if the Source pulses are long, the counter increments only once for each Source pulse.

Normally, the counter value and Counter n Internal Output signals change synchronously to the Source signal. With duplicate count prevention, the counter value and Counter n Internal Output signals change synchronously to the maximum timebase.

When To Use Duplicate Count Prevention

You should use duplicate count prevention if the following conditions are true.

- You are making a counter measurement
- You are using an external signal (such as PFI x) as the counter Source
- The frequency of the external Source is 25% of your maximum timebase or less
- You can have counter value and output to change synchronously with the maximum timebase

In all other cases, you should not use duplicate count prevention.

Enabling and Disabling Duplicate Count Prevention in NI-DAQmx

NI-DAQmx enables duplicate count prevention by default except in the following cases:

- The input terminal is an onboard timebase.
- Prescaling is enabled.
- The timing type is on demand.
- The CtrOutEvent.OutputTerm attribute/property is used in your application.

You can enable and disable duplicate count prevention in NI-DAQmx with the CI.DupCountPrevention attribute/property.

Incomplete Sample Detection

When performing a buffered time measurement of a digital signal, the initial sample is often invalid. The following devices detect such samples and discard them.

- Bus-powered M Series USB devices
- Parallel digital I/O modules connected to CompactDAQ chassis[9], TestScale chassis[10], CompactDAQ controllers[11], CompactRIO controllers[12] and CompactRIO

9. cDAQ-9171, 9174, 9178, 9179, 9181, 9184, 9185, 9188, 9188XT, 9189, and 9191

Single-Board controllers[13]. Refer to ***Digital I/O Considerations for C Series Devices***.
- NI 661x devices
- X Series devices

Related concepts:

- [Configuring a Time-Based Measurement in NI-DAQmx](#)
- [Digital I/O Considerations for C Series and TestScale Devices](#)

Prescaling

Prescaling allows the counter to count a signal that is faster than the maximum timebase of the counter. The TIO counters offer 8X and 2X prescaling on each counter. You can disable prescaling. Each prescaler consists of a small, simple counter that counts to eight (or two) and rolls over. This counter is specifically designed for this application and can count signals that are faster than the general purpose counters. The CtrN source signal on the general purpose counter is the divided signal from the simple counter.

Prescaling is for two-counter period and frequency measurements in which the measurement is made on a continuous, repetitive signal. The prescaling counter cannot be read; therefore, you cannot determine how many edges have occurred since the previous roll-over. Prescaling can be also used for counting edges provided it is acceptable to have an error of up to seven counts when using 8X prescaling or one count when using 2X prescaling.

Pulse Measurement Support

The following devices support pulse measurements:

- Parallel digital I/O modules connected to CompactDAQ chassis[14], TestScale

10. TS-15000 and TS-15010
11. cDAQ-9132, 9133, 9134, 9135, 9136, and 9137
12. cRIO-9040, 9041, 9042, 9043, 9045, 9046, 9047, 9048, 9049, 9053, 9054, 9055, 9056, 9057, and 9058
13. sbRIO-9603, 9608, 9609, 9628, 9629, and 9638

chassis[15], CompactDAQ controllers[16], CompactRIO controllers[17] and CompactRIO Single-Board controllers[18]. Refer to ***Digital I/O Considerations for C Series Devices***.

- NI 661x devices
- X Series devices

Related concepts:

- [Pulse Measurement](#)
- [Configuring a Time-Based Measurement in NI-DAQmx](#)
- [Digital I/O Considerations for C Series and TestScale Devices](#)

Sample Clock Timing Support for Time-Based Measurements

The following devices support Sample Clock timing for time-based measurements of digital signals, such as frequency, period, and two-edge separation measurements.

- Bus-powered M Series USB devices
- Parallel digital I/O modules connected to CompactDAQ chassis[19], TestScale chassis[20], CompactDAQ controllers[21], CompactRIO controllers[22] and CompactRIO Single-Board controllers[23]. Refer to ***Digital I/O Considerations for C Series Devices***.
- PXIe-6612 and 6614 devices.
- PXIe-6738 and 6739 devices.
- X Series Devices

---

14. cDAQ-9171, 9174, 9178, 9179, 9181, 9184, 9185, 9188, 9188XT, 9189, and 9191
15. TS-15000 and TS-15010
16. cDAQ-9132, 9133, 9134, 9135, 9136, and 9137
17. cRIO-9040, 9041, 9042, 9043, 9045, 9046, 9047, 9048, 9049, 9053, 9054, 9055, 9056, 9057, and 9058
18. sbRIO-9603, 9608, 9609, 9628, 9629, and 9638
19. cDAQ-9171, 9174, 9178, 9179, 9181, 9184, 9185, 9188, 9188XT, 9189, and 9191
20. TS-15000 and TS-15010
21. cDAQ-9132, 9133, 9134, 9135, 9136, and 9137
22. cRIO-9040, 9041, 9042, 9043, 9045, 9046, 9047, 9048, 9049, 9053, 9054, 9055, 9056, 9057, and 9058
23. sbRIO-9603, 9608, 9609, 9628, 9629, and 9638

> ✏ **Note** You cannot use Sample Clock timing for semi-period measurements on X Series devices.

Related concepts:

- [Configuring a Time-Based Measurement in NI-DAQmx](#)
- [Digital I/O Considerations for C Series and TestScale Devices](#)

Digital Filtering

Digital filtering rejects state transitions that do not stay at a state for a specified amount of time. For example, for an edge counting measurement with digital filtering, the device does not count an edge if the pulse width is not at least the specified time. For digital input tasks, the device does not recognize that a signal changed from one state to another unless the signal remains at that state for the specified amount of time.

This section contains information about digital filtering for C Series, DIO, M Series, SC Express, TIO, and X Series devices.

Digital Filtering Considerations for C Series Devices or or TestScale Modules

For C Series devices or or TestScale Modules, you can filter digital I/O lines and digital input signals.

Timing and Triggering Filters for Digital I/O Lines

You can configure digital filters on the device by choosing three fixed values (112.5 nS, 6.4 µS, 2.56 mS) or a custom filter value. The custom filter value must be the same for all lines across the device. For example, if you choose a filter value of 2 µS for PFI 0, any other filterable line on the device can only choose from the three fixed values and the 2 µs value selected for the custom filter. For each digital line or input terminal, there are four attributes/properties associated with these digital filters: Digital Filter Enable, Digital Filter Minimum Pulse Width, Digital Filter Timebase Source, and Digital Filter

Timebase Rate.

When you set the Digital Filter Enable to true, you must also configure the Digital Filter Minimum Pulse Width attribute/property. When you select a filter value with the Digital Filter Minimum Pulse Width attribute/property, the device uses an internal 32-bit utility counter to generate the desired filter value. If you would like to generate the filter clock using your own external signal, you can use the Digital Filter Timebase Source and Digital Filter Timebase Rate attributes/properties. You must configure both to use an external signal as the source for the digital filter. The Digital Filter Minimum Pulse Width attribute/property represents the minimum value that is guaranteed to be passed into the device. The maximum pulse width guaranteed to be blocked by the device is one filter clock tick smaller than the minimum pulse width guaranteed to pass the filter.

The following table lists the attributes/properties for terminals that can be digitally filtered.

Type	Attribute/Property
Channel	Frequency Input Terminal
	Period Input Terminal
	Count Edges Input Terminal
	Count Edges Count Direction
	Position A Input Terminal
	Position B Input Terminal
	Position Z Input Terminal
	Pulse Input Terminal (Time, Ticks, and Frequency)
	Pulse Width Input Terminal
	Two-Edge First Input Terminal
	Two-Edge Second Input Terminal
	Semi-Period Input Terminal
	Counter Input Timebase Source (External Only)
	Counter Output Timebase Source (External Only)

22222

2222222222222222

Type	Attribute/Property
Timing	Sample Clock Source
Triggering	Arm Start Digital Edge Source
	Pause Analog Level Source
	Pause Analog Window Source
	Pause Digital Level Source
	Reference Analog Edge Source
	Reference Analog Window Source
	Reference Digital Edge Source
	Start Analog Edge Source
	Start Analog Window Source
	Start Digital Edge Source

Filters for Digital Input Signals

Filters are also available on digital input lines, such as cDAQ1Mod1/port0/line0, but the filters do not support the fixed values mentioned previously or external timebase sources. The minimum filter pulse width for digital input lines is 50 nS and can be set in increments of 25 nS. All digital input lines on a module must use the same minimum filter pulse width. The maximum pulse width guaranteed to be rejected by the filter is half the pulse width guaranteed to pass the filter.

This filtering circuitry exists on the chassis and is available for all digital lines that exist on parallel digital modules. With a parallel digital module, input or output data is communicated in parallel between the module and the chassis backplane rather than being communicated serially.

Parallel Digital Input Modules for C Series Devices and TestScale Modules

- NI 9344
- NI 9401
- NI 9402
- NI 9411
- NI 9421

- NI 9422
- NI 9423
- NI 9435
- NI 9436
- NI 9437
- TS-15050 DIO P0

> ✎ **Note** The NI 9361 also performs digital filtering, but its filtering is done on the module rather than on the chassis. Refer to the *NI 9361 Datasheet* for more information about its filtering capabilities.

Digital Filtering Considerations for DIO Devices

Digital filtering is enabled by default on all isolated DIO devices that support digital filtering. The default minimum pulse width is 0.1 ms or 100 µs. Refer to the following table for a list of devices and their digital filtering settings.

Digital Filtering Setting	Devices
Digital Filtering Enabled by Default	- PCI-6510 - PCI-6511 - PCI-6514 - PCI-6515 - PCI-6518 - PCI-6519 - PCI-6527 - PCI-6528 - PXI-6511 - PXI-6514 - PXI-6515 - PXI-6527 - PXI-6528 - PXI-6529 - USB-6525

Digital Filtering Setting	Devices
Digital Filtering Disabled by Default	- PCI-6509 - PCIe-6509 - PXI-6509 - PXIe-6509
Digital Filtering Not Supported	- PCI-6503 - PCI-6512 - PCI-6513 - PCI-6516 - PCI-6517 - PCI-DIO-96 - PXI-6508 - PXI-6512 - PXI-6513 - USB-6501

Digital Filtering Considerations for TIO-Based Devices

> **Note** For digital filtering with NI 661x devices, refer to **_Digital Filtering Considerations for X Series and NI 661x Devices._**

There are two methods for filtering and synchronizing digital signals. One method is to synchronize the input signal to the maximum onboard timebase on the device. To do this, set Digital Synchronization Enable to true.

The other method is to pass the input of any PFI line through a digital debouncing filter. Each PFI line can independently select from four fixed values (5 µs, 1 µs, 500 ns, 100 ns) and one custom filter value. The custom filter value must be the same for each PFI line. That is, if you choose a filter value of 2 µs for a PFI line, other PFI lines on the device at the same time can only choose from the four fixed values and the 2 µs value selected as the custom filter value. For each counter input property, there are four attributes/properties associated with digital debounce filtering: Digital Filter Enable, Digital Filter Minimum Pulse Width, Digital Filter Timebase Source, and Digital Filter

Timebase Rate.

When you set the Digital Filter Enable to true, you must also configure the Digital Filter Minimum Pulse Width attribute/property. This value represents the minimum value that is guaranteed to pass into the TIO. The minimum pulse width guaranteed to be blocked is one-half of the Digital Filter Minimum Pulse Width attribute/property. When you select a custom filter value with the Minimum Pulse Width attribute/property, NI-DAQmx uses an internal 32-bit utility counter to generate the desired filter value. If you would like to generate the filter clock using your own external signal, you can use the Digital Filter Timebase Source and Digital Filter Timebase Rate attributes/properties. You must configure both to use an external signal as the source for the digital filter.

You cannot set both Digital Filter Enable and Digital Synchronization Enable to true at the same time. You can use only one of these digital filtering methods at a time.

The following table lists the counter input terminals that can be digitally filtered.

Type	Attribute/Property
Channel	Frequency Input Terminal
	Period Input Terminal
	Count Edges Input Terminal
	Count Edges Count Direction
	Position A Input Terminal
	Position B Input Terminal
	Position Z Input Terminal
	Pulse Width Input Terminal
	Two-Edge First Input Terminal
	Two-Edge Second Input Terminal
	Semi-Period Input Terminal
	Counter Input Timebase Source
	Counter Output Timebase Source
Timing	Sample Clock Source

Type	Attribute/Property
Triggering	Start Trigger Source
	Pause Trigger Source
	Arm Start Trigger Source

Digital Filtering Considerations for X Series and NI 661*X* Devices

For X Series and NI 661x devices, you can filter digital I/O lines and input signals.

You can configure digital filters on the device by choosing three fixed values (90 nS, 5.12 µS, 2.56 mS). For each digital line or input terminal, there are four attributes/ properties associated with these digital filters: Digital Filter Enable, Digital Filter Minimum Pulse Width, Digital Filter Timebase Source, and Digital Filter Timebase Rate.

When you set the Digital Filter Enable to true, you must also configure the Digital Filter Minimum Pulse Width attribute/property. This value represents the minimum value that is guaranteed to be passed into the device. The maximum pulse guaranteed to be blocked by the device varies by digital line. Refer to your device documentation for details. When you select a custom filter value with the Minimum Pulse Width attribute/ property, NI-DAQmx uses an internal 32-bit utility counter to generate the desired filter value. If you would like to generate the filter clock using your own external signal, you can use the Digital Filter Timebase Source and Digital Filter Timebase Rate attributes/ properties. You must configure both to use an external signal as the source for the digital filter.

When using digital filtering on port 0, you can enable bus mode using the Enable Bus Mode attribute/property. When enabled, the device observes changes to multiple lines as a single change if these conditions are met:

- The skew between them is less than the minimum pulse width of the filter.
- The lines remain stable for an additional minimum pulse width of the filter.

When using any other signal on the device that supports filtering (excluding port 0), you can select a custom filter value in addition to the three fixed values mentioned previously. The custom filter value must be the same for all lines across the device. For example, if you choose a filter value of 2 µS for PFI 0, any other filterable line on the

device can only choose from the three fixed values and the 2 µs value selected for the custom filter.

The following table lists the attributes/properties for terminals that can be digitally filtered.

Type	Attribute/Property
Channel	Digital Input Channel
	Frequency Input Terminal
	Period Input Terminal
	Count Edges Input Terminal
	Count Edges Count Direction
	Position A Input Terminal
	Position B Input Terminal
	Position Z Input Terminal
	Pulse Input Terminal
	Pulse Width Input Terminal
	Two-Edge First Input Terminal
	Two-Edge Second Input Terminal
	Semi-Period Input Terminal
	Counter Input Timebase Source
	Counter Output Timebase Source
Timing	Sample Clock Source
	AI Convert Clock Source
Triggering	Arm Start Trigger Source
	Pause Analog Level Source
	Pause Analog Window Source
	Pause Trigger Source
	Reference Analog Edge Source

Type	Attribute/Property
	Reference Analog Window Source
	Reference Digital Edge Source
	Start Analog Edge Source
	Start Analog Window Source
	Start Trigger Source

Digital Filtering Considerations for SC Express Devices

For SC Express devices, you can filter timing/triggering input signals.

You can configure digital filters on the device by choosing three fixed values (90 ns, 5.12 µs, 2.56 ms). For each digital line or input terminal, there are four attributes/ properties associated with these digital filters: DI.DigFltr.Enable, DI.DigFltr.MinPulseWidth, DI.DigFltr.TimebaseSrc, and DI.DigFltr.TimebaseRate.

When you set the DI.DigFltr.Enable to true, you must also configure the DI.DigFltr.MinPulseWidth attribute/property. This value represents the minimum value that is guaranteed to be passed into the device. When you select a custom filter value with the DI.DigFltr.MinPulseWidth attribute/property, NI-DAQmx uses an internal 32-bit utility counter to generate the desired filter value.

You can select a custom filter value in addition to the three fixed values mentioned previously. The custom filter value must be the same for all lines across the device. For example, if you choose a filter value of 2 µs for PFI 0, any other filterable line on the device can only be set to one of the three fixed values or the 2 µs value selected for the custom filter.

The following table lists the attributes/properties for terminals that can be digitally filtered.

Type	Attribute/Property
Timing	Sample Clock Source (SampClk.Src)
Triggering	Pause Digital Level Source (Pause.DigLvl.Src)

Type	Attribute/Property
	Reference Analog Edge Source (Ref.AnlgEdge.Src)
	Reference Analog Window Source (Ref.AnlgWin.Src)
	Reference Digital Edge Source (Ref.DigEdge.Src)
	Start Digital Edge Source (Start.DigEdge.Src)

FieldDAQ Filtering

The FD-11601, FD-11603, FD-11605, FD-11634, and FD-11637 can use filtering to provide an accurate representation of in-band signals while rejecting out-of-band signals. The filters discriminate between signals based on the frequency range, or bandwidth, of the signal. Use the AI.Filter.Enable DAQmx Channel property to enable or disable a filter. You can then specify the center or cutoff frequency (AI.Filter.Freq), the filter order (AI.Filter.Order), and choose a Brickwall, Butterworth, or Comb filter response (AI.Filter.Response). The filter selection applies to all channels on a bank.

Refer to your device user guide for additional information on filtering.

NI 9202, NI 9252, and NI 9253 Filtering

The NI 9202, NI 9252, and NI 9253 use a combination of analog and digital filtering to provide an accurate representation of in-band signals while rejecting out-of-band signals. The filters discriminate between signals based on the frequency range, or bandwidth, of the signal. The filtering is always enabled and the filter configuration applies to all channels on the module.

In NI-DAQmx, you can specify the center or cutoff frequency (AI.Filter.Freq), the filter order (AI.Filter.Order), and choose a Comb filter response (AI.Filter.Response). The NI 9252/9253 also provide a Butterworth filter.

Refer to the hardware datasheets for additional details on filtering for these devices.

Digital I/O

This section contains information specific to DIO devices.

Change Detection

This section contains information about change detection for C Series, DIO, and M Series devices.

Related concepts:

- Change-Detection Considerations for C Series and M Series Devices
- Change Detection Considerations for NI 6527 Devices

Change Detection Considerations for NI 6527 Devices

The ChangeDetect.Overflowed attribute/property uses the change detection overflow circuitry on a DIO device to determine if an overflow occurred. The NI 6527 change detection overflow circuitry does not detect an overflow if a single rising edge and a single falling edge are detected prior to reading a sample. It will detect overflows if two rising edges or two falling edges occur prior to reading a sample.

Change-Detection Considerations for C Series and M Series Devices

When performing a buffered change-detection task with an M Series device or an NI CompactDAQ system, the parallel digital input circuitry is automatically reserved and used for the task. Non-buffered tasks, including hardware-timed single point, do not reserve or use the parallel digital input circuitry.

Digital I/O Considerations for C Series and TestScale Devices

Digital I/O module capabilities depend on the type of digital signals that the module can measure or generate and the chassis the module is used in.

Serial digital I/O modules: NI 9375, NI 9403, NI 9425, NI 9426, NI 9476, NI 9477, NI 9478, TS-15120, and TS-15130.

Parallel digital I/O modules: NI 9344, NI 9401, NI 9402, NI 9411, NI 9421, NI 9422, NI

9423, NI 9435, NI 9436, NI 9437, NI 9472, NI 9474, NI 9475, NI 9481, NI 9482, NI 9485, and TS-15050 DIO P0.

Onboard parallel digital I/O modules on sbRIO-9628 and 9638: DIO 0-3, DIO 4-11, DIO 12-19, and DIO 20-27.

The modules can perform the following tasks.

Serial and parallel modules:

- Software and hardware-timed digital input/output tasks[24]

Parallel modules:

- Counter/timer tasks
- Accessing PFI signal tasks (can be used in up to two slots)

Related concepts:

- Software-Timed Tasks

Sample Clock Timing for Digital I/O

You can use sample clock timing for digital I/O on the following devices. [25]

AO Series 1

- NI 673x

24. Timed digital input/output restrictions:
    - You cannot use parallel and serial modules together on the same hardware timed task, unless they are in separate cDAQ chassis using multichassis device tasks.
    - You cannot use serial modules for triggering.
    - You cannot do both static and timed tasks at the same time on a single serial module.
    - You can only do hardware timing in one direction at a time on a serial module.
25. There is no dedicated onboard sample clock for digital I/O on these devices. You must use a different clock, typically the AI or AO Sample Clock.

CompactDAQ

- cDAQ-9132
- cDAQ-9133
- cDAQ-9134
- cDAQ-9135
- cDAQ-9136
- cDAQ-9137
- cDAQ-9138/9139
- cDAQ-9171
- cDAQ-9174
- cDAQ-9178
- cDAQ-9179
- cDAQ-9181
- cDAQ-9184
- cDAQ-9185
- cDAQ-9188
- cDAQ-9188XT
- cDAQ-9189
- cDAQ-9191

CompactRIO

- cRIO-9040
- cRIO-9041
- cRIO-9042
- cRIO-9043
- cRIO-9045
- cRIO-9046
- cRIO-9047
- cRIO-9048
- cRIO-9049
- cRIO-9053
- cRIO-9054
- cRIO-9055
- cRIO-9056
- cRIO-9057
- cRIO-9058

- sbRIO-9603
- sbRIO-9608
- sbRIO-9609
- sbRIO-9628
- sbRIO-9629
- sbRIO-9638

M Series 1

- NI 622x
- NI 625x
- NI 628x

S Series 1

- NI 6115
- NI 6120
- NI 6132
- NI 6133

NI 653x

- PCI-6533 (DIO-32HS)
- PXI-6533
- PCI-6534
- PXI-6534
- PCIe-6535
- PXIe-6535
- PCIe-6536
- PXIe-6536
- PCIe-6537
- PXIe-6537

NI 661x Devices

- NI 6612
- NI 6614

X Series Devices

All X Series devices support sample clock timing for digital I/O.

Related concepts:

- [Digital I/O Considerations for C Series and TestScale Devices](#)
- [Digital I/O Considerations for C Series Devices](#)

Handshake Timing Devices

You can use handshake timing for digital I/O on the following devices:

- PCI-6025E
- PCI-6533 (DIO-32HS)
- PCI-6534
- PCI-DIO-24
- PCI-DIO-96
- PCIe-6535
- PCIe-6536
- PCIe-6537
- PXI-6025E
- PXI-6508
- PXI-6533
- PXI-6534
- PXIe-6535
- PXIe-6536
- PXIe-6537

Burst Handshaking Timing Defaults for NI 653*X* Devices

The following table lists the default terminals used for burst handshake timing.

> ✏️ **Note** The NI 6533 and NI 6534 have two timing engines, Timing Engine 1 and Timing Engine 0. Each timing engine is associated with PFI lines. The timing

engine you use is determined by the digital lines you use. If the least significant port is Port 0, NI-DAQmx picks Timing Engine 0. If the least significant port is Port 2, NI-DAQmx picks Timing Engine 1.

Device	Pause Trigger Default	Ready for Transfer Event Default
PCI-6533 (DIO-32HS), PXI-6533, PCI-6534, PXI-6534	PFI 2 (Timing Engine 0), PFI 3 (Timing Engine 1)	PFI 6 (Timing Engine 0), PFI 7 (Timing Engine 1)
PCIe-6535, PXIe-6535, PCIe-6536, PXIe-6536, PCIe-6537, PXIe-6537	PFI 0	PFI 1

The recommended sample clock terminal for burst handshake timing is PFI 4 (Timing Engine 0) or PFI 5 (Timing Engine 1).

Burst Handshake Timing for Digital I/O

You can use burst handshake timing for digital I/O on the following devices:

- PCI-6533 (DIO-32HS)
- PCI-6534
- PCIe-6535
- PCIe-6536
- PCIe-6537
- PXI-6533
- PXI-6534
- PXIe-6535
- PXIe-6536
- PXIe-6537

Handshaking Line Configuration

NI 6533/6534 devices have two timing engines, each of which use a set of default lines for handshaking and burst handshaking. You can specify a different timing engine to use the handshaking lines associates with that timing engine.

Related concepts:

- [NI 6533, 6534 Timing Engines](#)
- [Handshake Timing Defaults](#)
- [Burst Handshaking Timing Defaults for NI 653x Devices](#)

Handshake Timing Defaults

The following table lists the default terminals used for handshake timing for NI 653x devices.

> ✏️ **Note** The NI 6533 and NI 6534 have two timing engines, Timing Engine 1 and Timing Engine 0. Each timing engine is associated with PFI lines. The timing engine you use is determined by the digital lines you use. If the least significant port is Port 0, NI-DAQmx picks Timing Engine 0. If the least significant port is Port 2, NI-DAQmx picks Timing Engine 1.

Device	Handshake Trigger Source Terminal Default	Handshake Event Output Terminal Default
PCI-6533 (DIO-32HS), PXI-6533, PCI-6534, PXI-6534	PFI 2 (Timing Engine 0), PFI 3 (Timing Engine 1)	PFI 6 (Timing Engine 0), PFI 7 (Timing Engine 1)
PCIe-6535, PXIe-6535, PCIe-6536, PXIe-6536, PCIe-6537, PXIe-6537	PFI 0	PFI 1

The recommended sample clock terminal for burst handshake timing is PFI 4 (Timing Engine 0) or PFI 5 (Timing Engine 1).

Watchdog Timers

Watchdog timers are a hardware feature that you can use to detect a failure in the software controlling the device. Software failures could include a system crash or a loop rate that is slower than you intend. To use a watchdog timer, you must use a watchdog timer task. When you create a watchdog timer task, you specify the timeout value for the watchdog timer and a set of expiration states for output physical channels on the device. The channels go to those expiration states if the watchdog

timer expires. In addition, you cannot perform any actions with the task until you reset it.

Your application must continuously reset the watchdog timer to prevent it from expiring. For example, if you have a digital I/O application, and you expect a loop in the application to acquire and analyze data 10 times per second, you should set the watchdog timer to expire in 100 ms and reset the timer inside the digital I/O loop. If the loop does not execute once every 100 ms, the watchdog timer expires and the device goes into the expired state. You must then clear the expiration or reset the device.

Also, you can use the Expiration Trigger to cause the watchdog timer to expire. Set the timeout of the watchdog timer task to -1 to disable expiration due to timeout if you want the Expiration Trigger to be the only mechanism to cause expiration.

Note For X Series devices, PFI lines used for routing either by the Connect Terminals function/VI or by a task and also guarded by a watchdog timer will go to their assigned safe states if the watchdog timer expires. The routes or tasks using the PFI lines will not necessarily be notified of the expiration.

Note For the NI 9260, all channels on a watchdog task must use the same expiration state. If the expiration state is set to Voltage, you can specify different voltage levels for each channel. If the expiration state is set to No Change, the channels revert to their idle output behavior on expiration.

Pause Triggering

This section contains information about Pause Triggering for AO Series, DSA, E Series, M Series, S Series, SC Express, and TIO devices.

Pause Trigger Considerations for AO Series Devices

The source of your sample clock can affect when your generation resumes after the deassertion of a Pause Trigger.

Analog Output

When you generate analog output signals, the generation pauses as soon as the Pause Trigger is asserted. If the source of your sample clock is the onboard clock, the generation resumes as soon as the Pause Trigger is deasserted. The NI 6733 and NI 6251 behave as if an external clock is being used, even if the source of the sample clock is the onboard clock. For these devices, the generation resumes as soon as the Pause Trigger is deasserted and another edge of the sample clock is received.

Pause Trigger
Sample Clock

If you are using any signal other than the onboard clock as the source of your sample clock, the generation resumes as soon as the Pause Trigger is deasserted and another edge of the sample clock is received, as shown in the following figure.

Pause Trigger
Sample Clock

Counters

Continuous pulse-train generation: The pulse-train generation pauses as soon as the Pause Trigger is asserted, not at the end of a pulse. The pulse train resumes after the Pause Trigger is deasserted. A Pause Trigger elongates either the high or low pulse depending on which one was being generated at the time the Pause Trigger was asserted.

Pause Trigger
Pulse Train

Nonbuffered edge counting: The counter stops counting edges as soon as the Pause Trigger is asserted and resumes counting edges after the Pause Trigger is deasserted.

Pause Trigger Considerations for C Series Devices

The source of your sample clock often can affect when your acquisition or generation pauses and resumes with the assertion and deassertion of a Pause Trigger.

Counter Input for the NI 9361
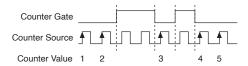
When you acquire counter input signals, the acquisition pauses as soon as Pause Trigger is asserted. If the source of your sample clock is the onboard clock, the acquisition resumes after the Pause Trigger is deasserted, and the leftover period (A) elapses. The leftover period is the unknown time in the sample clock period in which the pause trigger occurred. In the following image, T represents the signal period and A represents the unknown time between the clock pulse and the pause trigger.

If you are using any signal other than the onboard clock as the source of your sample clock, the acquisition resumes as soon as the Pause Trigger is deasserted and another edge of the sample clock is received as shown in the following figure.

To pause an edge counting task, use the `CountEdges.Gate` attributes/properties.

Pause Trigger Considerations for DSA Devices
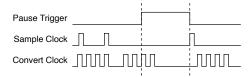
DSA devices do not support Pause Triggering.

Pause Trigger Considerations for E Series and M Series Devices
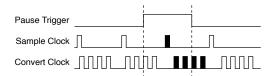
The source of your sample clock often can affect when your acquisition or generation pauses and resumes with the assertion and deassertion of a Pause Trigger.

Analog Input

When you measure analog input signals and the Pause Trigger is asserted, the current sample across all channels finishes before pausing. For instance, if you are sampling four channels and the second channel is being sampled at the time the Pause Trigger is asserted, the second, third, and fourth channels complete their sample before the acquisition pauses. If you are using the onboard clock as the source of your sample clock, the acquisition resumes as soon as the Pause Trigger is deasserted.

If you are using any signal other than the onboard clock as the source of your sample clock, the acquisition resumes as soon as the Pause Trigger is deasserted and another edge of the sample clock is received as shown in the following figure.

Analog Output

When you generate analog output signals, the generation pauses as soon as the Pause Trigger is asserted. If the source of your sample clock is the onboard clock, the generation resumes as soon as the Pause Trigger is deasserted.
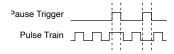
If you are using any signal other than the onboard clock as the source of your sample clock, the generation resumes as soon as the Pause Trigger is deasserted and another edge of the sample clock is received as shown in the following figure.

Counters

Continuous pulse-train generation: The pulse-train generation pauses as soon as the Pause Trigger is asserted, not at the end of a pulse. The pulse train resumes after the Pause Trigger is deasserted. A Pause Trigger elongates either the high or low pulse depending on which one was being generated at the time the Pause Trigger was asserted.

Nonbuffered edge counting: The counter stops counting edges as soon as the Pause Trigger is asserted and resumes counting edges after the Pause Trigger is deasserted.

Pause Trigger Considerations for S Series Devices

The source of your sample clock often can affect when your acquisition or generation pauses and resumes with the assertion and deassertion of a Pause Trigger.

Analog Input and Analog Output

When you generate analog output signals or acquire analog input signals, the generation/acquisition pauses as soon as the Pause Trigger is asserted. If the source of your sample clock is the onboard clock, the generation/acquisition resumes as soon as the Pause Trigger is deasserted.

If you are using any signal other than the onboard clock as the source of your sample clock, the generation/acquisition resumes as soon as the Pause Trigger is deasserted and another edge of the sample clock is received as shown in the following figure.

Pause triggers also require special consideration when used on a device with a pipelined ADC. See **Timing Considerations for S Series** for how pipelined ADCs and Pause Triggers can affect your measurement.

Related concepts:
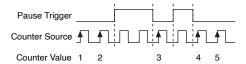
- Timing Considerations for S Series

Counters

Continuous pulse-train generation: The pulse-train generation pauses as soon as the Pause Trigger is asserted, not at the end of a pulse. The pulse train resumes after the Pause Trigger is deasserted. A Pause Trigger elongates either the high or low pulse depending on which one was being generated at the time the Pause Trigger was asserted.

Nonbuffered edge counting: The counter stops counting edges as soon as the Pause Trigger is asserted and resumes counting edges after the Pause Trigger is deasserted.
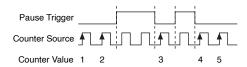
> **Note** The NI 6154 does not support pause triggering.

Pause Trigger Considerations for TIO Devices

Counters

Nonbuffered Edge Counting

The counter stops counting edges as soon as the Pause Trigger is asserted and resumes counting edges after the Pause Trigger is deasserted.

Continuous Pulse-Train Generation

The pulse-train generation pauses as soon as the Pause Trigger is asserted, not at the end of a pulse. The pulse train resumes after the Pause Trigger is deasserted. A Pause Trigger elongates either the high or low pulse depending on which one was being generated at the time the Pause Trigger was asserted.

Pause Trigger Considerations for SC Express Devices
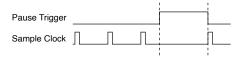
> ✏️ **Note** NI 433x devices and the NI 4340 do not support pause triggering.

The source of your sample clock often can affect when your acquisition pauses and resumes with the assertion and deassertion of a Pause Trigger.

Analog Input and Analog Output for Simultaneous Sampling SC Express Devices

When you generate analog output signals or acquire analog input signals on a simultaneous sampling SC Express device, such as the NI 4322 or NI 4300, the generation/acquisition pauses as soon as the Pause Trigger is asserted. If the source of your sample clock is the onboard clock, the generation/acquisition resumes as soon as the Pause Trigger is deasserted.

If you are using any signal other than the onboard clock as the source of your sample clock, the generation/acquisition resumes as soon as the Pause Trigger is deasserted and another edge of the sample clock is received, as shown in the following figure.
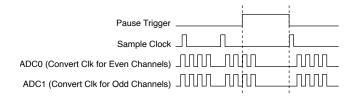
Analog Input for the NI 4353
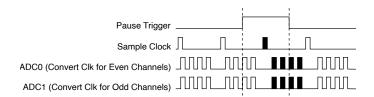
The NI 4353 has three ADCs. There is ADC0, which is for the even analog input channels (for instance, ai4, ai6, and ai10), ADC1, which is for the odd analog input channels (for instance, ai1, ai3, and ai5), and ADC2, which is for the cold-junction compensation

channels.

If the Pause Trigger is asserted, and you sample from multiple cold-junction compensation channels, multiple odd-numbered analog input channels, or multiple even-numbered analog output channels, the current sample across all channels finishes before pausing. For instance, if you are sampling eight channels, ai0:7, and ai4 is being sampled when the Pause Trigger is asserted, the remaining four channels complete their sample before the acquisition pauses. If you are using the onboard clock as the source of your sample clock, the acquisition resumes as soon as the Pause Trigger is deasserted.

If you are using any signal other than the onboard clock as the source of your sample clock, the acquisition resumes as soon as the Pause Trigger is deasserted and another edge of the sample clock is received, as shown in the following figure.
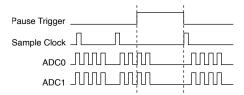
Analog Input for the NI 4357

The NI 4357 has five ADCs. The ADCs are staggered in the following ways:
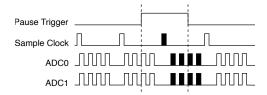
- ADC0 maps to the analog input channels ai0, ai5, ai10, ai15
- ADC1 maps to the analog input channels ai1, ai6, ai11, ai16
- ADC2 maps to the analog input channels ai2, ai7, ai12, ai17
- ADC3 maps to the analog input channels ai3, ai8, ai13, ai18
- ADC4 maps to the analog input channels ai4, ai9, ai14, ai19

If the Pause Trigger is asserted, and you sample from multiple channels on the same ADC, the current sample across all channels finishes before pausing. For instance, if

you are sampling eight channels, ai0:7, and ai4 is being sampled when the Pause Trigger is asserted, the remaining four channels complete their sample before the acquisition pauses. If you are using the onboard clock as the source of your sample clock, the acquisition resumes as soon as the Pause Trigger is deasserted.

If you are using any signal other than the onboard clock as the source of your sample clock, the acquisition resumes as soon as the Pause Trigger is deasserted and another edge of the sample clock is received, as shown in the following figure.

Physical Channels

This section contains information about physical channels for AO Series, bus-powered M Series, C Series, E Series, M Series, NI 6010, NI 6154, NI 6221 (37-pin), NI 623x, NI 6533/6534, NI 6535/6536/6537, NI ELVIS II Family, S Series, SC Express, SCXI, SCC, TIO, USB DAQ, and X Series devices.

AO Series Physical Channels

`Dev1` in physical channel names is the default device name for AO Series devices. You can change these names in MAX.

Related reference:

- [AO Series, E Series, and S Series Signal Connections for Counters](#)

Analog Output

An AO Series device has between four and 64 analog output physical channels named `Dev1/ao0` to `Dev1/ao63`.

For more detailed information on the device physical AO characteristics, refer to your device user manual and specifications.

Digital Input and Output

All AO Series devices have eight lines of digital input and output named `Dev1/port0/line0` through `Dev1/port0/line7`. These lines belong to a single port, and the physical channel `Dev1/port0` refers to all eight lines at once.

Counter Input and Output

All AO Series devices have two counter/timers referred to by the physical channel names `Dev1/ctr0` and `Dev1/ctr1`. Unlike the other I/O types, these physical channel names do not refer to terminals on the I/O connector but instead to circuits within the device. There are three primary terminals associated with each counter. These are the terminals used as the SOURCE, GATE and OUT functions. NI-DAQmx has default values for these terminals. For counter input tasks, if you know whether your signal provides the SOURCE or GATE function and wire your signal to the default, you do not have to set the Input Terminal attribute/property.

Counter	SOURCE Default	GATE Default	OUT Default
Dev1/ctr0	PFI 8	PFI 9	CTR 0 OUT
Dev/ctr1	PFI 3	PFI 4	CTR 1 OUT

C Series and TestScale Module Physical Channels

In physical channel names:

- `cDAQ1Mod1` is the default device name for a C Series device plugged into a USB or Standalone CompactDAQ chassis, where cDAQ1 is the default chassis device name, and Mod1 refers to the slot number

- `TS1Mod1` is the default device name for a TestScale module plugged into a Standalone TestScale chassis, where TS1 is the default chassis device name, and Mod1 refers to the slot number

For C Series devices plugged into a network CompactDAQ chassis, such as the NI cDAQ-9188, the default chassis device name is the host name of the chassis. You can change these names in MAX. For C Series devices plugged into a supported CompactRIO[26] or CompactRIO Single-Board[27] controller, Mod1 is the default device name.

Related concepts:

- [Digital I/O Considerations for C Series and TestScale Devices](#)
- [CompactRIO Single-Board Controller Physical Channels](#)
- [Digital I/O Considerations for C Series Devices](#)

Related reference:

- [C Series and TestScale Module Signal Connections for Counters](#)
- [C Series Signal Connections for Counters](#)

Analog Input

The following table lists the number and naming of analog input physical channels for C Series devices and TestScale modules.

Device	Number of Channels	Naming
NI 9218, NI 9250, NI 9251	2	`cDAQ1Mod1/ai0` to `cDAQ1Mod1/ai1`
NI 9225, NI 9230, NI 9232, NI 9246, NI 9247	3	`cDAQ1Mod1/ai0` to `cDAQ1Mod1/ai2`
NI 9210, NI 9211, NI 9215, NI 9217, NI 9219, NI 9222, NI 9223, NI 9227, NI 9229, NI 9234, NI 9237, NI 9238, NI	4	`cDAQ1Mod1/ai0` to `cDAQ1Mod1/ai3`

---

26. cRIO-9040, 9041, 9042, 9043, 9045, 9046, 9047, 9048, 9049, 9053, 9054, 9055, 9056, 9057, and 9058.
27. sbRIO-9603, 9608, 9609, 9628, 9629, and 9638.

Device	Number of Channels	Naming
9239, NI 9775		
NI 9242, NI 9244	4	`cDAQ1Mod1/ai0` to `cDAQ1Mod1/ai2`, `cDAQ1Mod1/neutral`
NI 9201, NI 9203, NI 9212, NI 9216, NI 9221, NI 9226, NI 9224, NI 9228, NI 9231, NI 9235, NI 9236, NI 9252, NI 9253	8	`cDAQ1Mod1/ai0` to `cDAQ1Mod1/ai7`
NI 9202, NI 9207, NI 9208, NI 9209, NI 9213, NI 9214 , NI 9220	16	`cDAQ1Mod1/ai0` to `cDAQ1Mod1/ai15`
NI 9205, NI 9206	32	`cDAQ1Mod1/ai0` to `cDAQ1Mod1/ai31`
TS-15100	32	`TS1Mod1/ai0` to `TS1Mod1/ai31`

On the NI 9205 and NI 9206, you can configure channels 0-7 and 16-23 as the positive channel of a differential pair. If N is this channel, channel N + 8 is the negative input of the pair. For instance, if you configure channel 1 in differential mode, the positive input is channel 1, and channel 9 is the negative input. Use only the physical channel name of the positive channel (not both) when creating a differential channel.

You can use channels from multiple analog input C Series devices in the same NI-DAQmx task.

- Up to three analog input tasks can run at a given time per chassis in a cDAQ-91xx[28] or TestScale[29] chassis.
- Up to eight analog input tasks can run a given time in a CompactRIO controller or a CompactRIO Single-Board controller.
- Up to two analog input tasks with a C Series Delta-Sigma device can run at a given time per chassis in a cDAQ-91xx[30] or TestScale[31] chassis.

28. cDAQ-9132, 9133, 9134, 9135, 9136, 9137, 9174, 9178, 9179, 9184, 9185, 9188, 9188XT, and 9189
29. TS-15000 and TS-15010
30. cDAQ-9132, 9133, 9134, 9135, 9136, 9137, 9174, 9178, 9179, 9184, 9185, 9188, 9188XT, and 9189
31. TS-15000 and TS-15010

- Up to eight analog input tasks with a C Series Delta-Sigma device can run at a given time in a CompactRIO controller or a CompactRIO Single-Board controller.

Related concepts:

- C Series Device Groupings

Strain and Wheatstone Bridge Measurements

The NI 9235, NI 9236, and NI 9237 support only the AI Strain Gage, AI Force Bridge, AI Pressure Bridge, AI Torque Bridge, AI Bridge (V/V), and AI Custom Voltage With Excitation channel types.

When using the NI 9218, NI 9219, NI 9235, NI 9236, or NI 9237 with an AI Custom Voltage With Excitation channel, you must set the AI.Excit.UseForScaling attribute/property to true. This attribute/property causes the channel to return ratiometric data: Vin/Vex. The NI 9219, NI 9235, NI 9236, and NI 9237 modules perform this division in hardware.

For the NI 9219, NI-DAQmx requires the AI.Excit.Val attribute/property to be set to 2.5 V for AI Strain Gage and AI Custom Voltage With Excitation channel types and to 500 µA for resistance and RTD measurements. The actual excitation voltage or current output by the NI 9219 varies with the sensor resistance or the load being measured.

For the NI 9218, NI-DAQmx requires the AI.Excit.Val attribute/property to be set to 3.3 V or 2 V for AI Strain Gage and AI Custom Voltage With Excitation (bridge mode), to 12 V for AI Custom Voltage With Excitation (no bridge mode) for powered sensors, and to 2 mA for IEPE.

NI 9218, NI 9219, NI 9235, NI 9236, and NI 9237 devices return a voltage ratio rather than a voltage. Therefore, use the AI.Bridge.InitialRatio attribute/property to specify the initial voltage ratio, or set the AI.Bridge.InitialVoltage attribute/property to the ratio Vin/Vex returned by the device, multiplied by Vex.

The NI 9219 does not have quarter bridge completion circuitry, which affects AI Strain Gage Quarter Bridge I channels and AI Custom Voltage With Excitation Quarter Bridge channels (but not AI Strain Gage Quarter Bridge II channels). With these channels, the NI 9219 performs a 2-wire resistance measurement on the active gage element, then NI-DAQmx uses software scaling to convert the resistance measurement into a bridge

ratio. For these channels, the polynomial coefficients specified by the AI.DevScalingCoeff attribute/property convert unscaled data into Ohms, not into V/V. Likewise, the AI.Rng.High/AI.Rng.Low attributes/properties should be specified in units of Ohms, not V/V.

When the NI 9219 is in quarter bridge mode, you need to use the AI.Bridge.NomResistance attribute/property to control whether the channel uses the 120 Ω range or the 350 Ω range.

The NI 9218 has an internal full bridge. However, it can support quarter-bridge and half-bridge measurements with the proper accessories.

NI 9218 Powered Sensor Measurements

The NI 9218 supports powered sensor measurements. For voltage powered sensor measurements, you can use either the DAQmx Create Channel function/VI (the AI Custom Voltage with Excitation measurement type) with the Bridge Configuration set to No Bridge, or you can use the DAQmx Create Channel function/VI (voltage measurement type) with the AI.Excit.Val attribute/property. For current powered sensor measurements, you have to use the DAQmx Create Channel function/VI (current measurement type) with the AI.Excit.Val attribute/property.

Analog Output

The following table lists the number and naming of analog output physical channels for C Series devices.

Device	Number of Channels	Naming
NI 9260	2	`cDAQ1Mod1/ao0` to `cDAQ1Mod1/ao1`
NI 9263, NI 9265	4	`cDAQ1Mod1/ao0` to `cDAQ1Mod1/ao3`
TS-15110	4	`TS1Mod1/ao0` to `TS1Mod1/ao3`
NI 9262	6	`cDAQ1Mod1/ao0` to `cDAQ1Mod1/ao5`
NI 9266	8	`cDAQ1Mod1/ao0` to `cDAQ1Mod1/ao7`
NI 9264	16	`cDAQ1Mod1/ao0` to `cDAQ1Mod1/ao15`

You can use channels from multiple analog output C Series devices in the same analog output task. With certain cDAQ-91xx[32] or TestScale[33] chassis, if the task is hardware-timed and uses only onboard memory, there is a limit of 16 channels per task, but if the task is software-timed or does not use only onboard memory, the number of channels is limited only by the number of devices. Only one hardware-timed analog output task per CompactDAQ chassis can run at a given time. With a CompactRIO controller or CompactRIO single-board controller, up to eight hardware-timed analog output tasks can run at a given time.

When using the NI 9262, NI 9263, NI 9264, NI 9265, NI 9266, or TS-15110 you can run only one type of timing at a time. You can have one software-timed task per channel or one hardware-timed task running on a device at one time, but you cannot have a combination of timing on that device. For instance, you can run up to four software-timed tasks on the NI 9265 concurrently, but running one hardware-timed task with one software-timed task generates an error.

When using the NI 9260, if you start a task that uses a different channel than the one you are currently using, the channel no longer in use reverts to the calibrated 0 V value.

Digital Input and Output

The following table lists the number, type, and naming of digital input/output lines for C Series devices.

Device	Lines	Type	Naming
NI 9344	4	digital input	`cDAQ1Mod1/port0/line0` to `cDAQ1Mod1/port0/line3`
	4	digital output	`cDAQ1Mod1/port1/line0` to `cDAQ1Mod1/port1/line3`
NI 9402, NI 9435, NI 9481, 9482	4	digital input and/or output	`cDAQ1Mod1/port0/line0` to `cDAQ1Mod1/port0/line3`
NI 9411	6	digital input	`cDAQ1Mod1/port0/line0` to `cDAQ1Mod1/port0/line5`
NI 9401, NI 9421, NI 9422, NI 9423, NI	8	digital input	`cDAQ1Mod1/port0/line0` to

32. cDAQ-9132, 9133, 9134, 9135, 9136, 9137, 9174, 9178, 9179, 9184, 9185, 9188, 9188XT, and 9189
33. TS-15000 and TS-15010

Device	Lines	Type	Naming
9472, NI 9474, NI 9475, NI 9485		and/or output	`cDAQ1Mod1/port0/line7`
TS-15050 DIO P0	8	digital input and/or output	`TS1Mod1/port0/line0` to `TS1Mod1/port0/line7`
NI 9436, NI 9437	8	digital input	`cDAQ1Mod1/port0/line0` to `cDAQ1Mod1/port0/line7`
NI 9375	16	digital input	`cDAQ1Mod1/port0/line0` to `cDAQ1Mod1/port0/line15`
	16	digital output	`cDAQ1Mod1/port1/line0` to `cDAQ1Mod1/port1/line15`
NI 9478	16	digital output	`cDAQ1Mod1/port0/line0` to `cDAQ1Mod1/port0/line15`
NI 9403, NI 9425, NI 9426, NI 9476, NI 9477	32	digital input and/or output	`cDAQ1Mod1/port0/line0` to `cDAQ1Mod1/port0/line31`
TS-15120, TS-15130	32	digital input and/or output	`TS1Mod1/port0/line0` to `TS1Mod1/port0/line31`

Note Digital I/O module capabilities depend on the type of digital signals that the module can measure or generate and the chassis the module is used in. Refer to *Digital I/O Considerations for C Series* for more information.

Chassis Counter Input and Output

The CompactDAQ chassis, CompactDAQ controllers, CompactRIO controllers, and CompactRIO single-board controllers have four counters. You can use counters/timers with a C Series device in any slot. CompactRIO single-board controllers feature onboard I/O connectors in addition to C Series device access through the RMC connector. See *CompactRIO Single-Board Controller Physical Channels* for more information and counter terminal default configuration of CompactRIO single-board controller onboard I/O connectors.

These chassis also have a 4-bit frequency output generator, referred to as cDAQ1Modx/freqout, where x is the slot in which the module is located.

Each counter has four primary terminals associated with it. These are the terminals used as the SOURCE, GATE, AUX, and OUT functions. NI-DAQmx has default values for these terminals. For counter input tasks, if you know whether your signal provides the SOURCE, AUX, or GATE function and wire your signal to the default, you do not have to set the Input Terminal attribute/property.

Counter/timers are only supported on parallel digital I/O modules. Refer to ***Digital I/O Considerations for C Series*** for more information.

The following table shows the counter terminal defaults for 8-channel DIO/DI/DO C Series devices.

PFI Signal	Physical Channel Name
PFI 0	`ctr0 src/ctr2 aux`
PFI 1	`ctr0 gate/ctr2 out`
PFI 2	`ctr0 aux/ctr2 gate/freqout`
PFI 3	`ctr0 out/ctr2 src`
PFI 4	`ctr1 src/ctr3 aux`
PFI 5	`ctr1 gate/ctr3 out`
PFI 6	`ctr1 aux/ctr3 gate`
PFI 7	`ctr1 out/ctr3 src`

The following table shows the counter terminal defaults for 6-channel DIO/DI/DO C Series devices.

PFI Signal	Physical Channel Name
PFI 0	`ctr0 src/ctr2 gate`
PFI 1	`ctr0 gate/ctr2 aux/ctr2 out/freqout`
PFI 2	`ctr0 aux/ctr0 out/ctr2 src`
PFI 3	`ctr1 src/ctr3 gate`
PFI 4	`ctr1 gate/ctr3 aux/ctr3 out`

PFI Signal	Physical Channel Name
PFI 5	`ctr1 aux/ctr1 out/ctr3 src`

The following table shows the counter terminal defaults for 4-channel DIO/DI/DO C Series devices.

PFI Signal	Physical Channel Name
PFI 0	`ctr0 src/ctr0 out/ctr2 aux/ctr3 gate`
PFI 1	`ctr0 gate/ctr1 aux/ctr2 src/ctr2 out/freqout`
PFI 2	`ctr0 aux/ctr1 gate/ctr3 src/ctr3 out`
PFI 3	`ctr1 src/ctr1 out/ctr2 gate/ctr3 aux`

Module Counter Input and Output

The following table lists the number, type, and naming of counter input lines for C Series counter modules.

Device	Counter	Type	Names
NI 9361	8	Counter input	`cDAQ1Mod1/ctr0` to `cDAQ1Mod1/ctr7`

There are 8 PFI lines, from PFI0 to PFI7. Each counter can use any of the input PFI lines to perform measurements. Two counters can use the same PFI line as inputs for their measurements, as long as the input configurations are the same for both counters.

The following table shows the terminal defaults for edge counting measurements for C Series counter modules.

Counter	Input terminal	Reset	Direction
ctr0	PFI0	PFI4	PFI7
ctr1	PFI1	PFI5	PFI6
ctr2	PFI2	PFI6	PFI5

Counter	Input terminal	Reset	Direction
ctr3	PFI3	PFI7	PFI4
ctr4	PFI4	PFI0	PFI3
ctr5	PFI5	PFI1	PFI2
ctr6	PFI6	PFI2	PFI1
ctr7	PFI7	PFI3	PFI0

The following table shows the terminal defaults for position measurements for C Series counter modules.

Counter	A	B	Z
ctr0	PFI0	PFI4	PFI7
ctr1	PFI1	PFI5	PFI6
ctr2	PFI2	PFI6	PFI5
ctr3	PFI3	PFI7	PFI4
ctr4	PFI4	PFI0	PFI3
ctr5	PFI5	PFI1	PFI2
ctr6	PFI6	PFI2	PFI1
ctr7	PFI7	PFI3	PFI0

The following table shows the terminal defaults for velocity measurements for C Series counter modules that support velocity measurements.

Counter	A	B
ctr0	PFI0	PFI4
ctr1	PFI1	PFI5
ctr2	PFI2	PFI6
ctr3	PFI3	PFI7
ctr4	PFI4	PFI0
ctr5	PFI5	PFI1

Counter	A	B
ctr6	PFI6	PFI2
ctr7	PFI7	PFI3

The following table shows the terminal defaults for frequency, period, duty cycle, and pulse width measurements for C Series counter modules.

Counter	Input Terminal
ctr0	PFI0
ctr1	PFI1
ctr2	PFI2
ctr3	PFI3
ctr4	PFI4
ctr5	PFI5
ctr6	PFI6
ctr7	PFI7

Connections for PFI lines should be based on the terminal configuration property. PFI + and PFI- should be connected in differential mode, and PFI + and COM should be connected for RSE. The default terminal configuration for all counter input terminals is RSE.

CompactRIO Single-Board Controller Physical Channels

The sbRIO-9603, 9608, and 9609 controllers do not have onboard physical channels. The sbRIO-9628, 9629, and 9638 controllers have the following onboard IO characteristics.

Analog Input

Device	Number of Channels	Naming
sbRIO-9628, sbRIO-9629, sbRIO-9638	16	`Conn0_AI/ai0` to `Conn0_AI/ai15`

Analog Output

Device	Number of Channels	Naming
sbRIO-9628, sbRIO-9629, sbRIO-9638	4	`Conn0_AO/ao0` to `Conn0_AO/ao3`

Digital Input and Output

Device	Lines	Type	Naming
sbRIO-9628, sbRIO-9629	4	digital input and/or output	`Conn0_DIO0-3/port0/line0` to `Conn0_DIO0-3/port0/line3`
sbRIO-9638	4	digital input and/or output	`Conn0_DIO0-3/port0/line0` to `Conn0_DIO0-3/port0/line3`

	8	digital input and/or output	`Conn1_DIO4-11/port0/line4` to `Conn1_DIO4-11/port0/line11`
			
			`Conn1_DIO12-19/port0/line12` to `Conn1_DIO12-19/port0/line19`
			
			`Conn1_DIO20-27/port0/line20` to `Conn1_DIO20-27/port0/line27`

Chassis Counter Input and Output

The sbRIO-9628, 9629, and 9638 have four counters. You can use counters/timers with a C Series device using the RMC connector or with onboard I/O on models where these I/O connectors are available. The following information applies to CompactRIO Single-Board controller onboard I/O. See **C Series Physical Channels** for more information and counter terminal default configuration of C Series device I/O using the RMC connector.

These chassis also have a 4-bit frequency output generator, referred to as `devicename/freqout`, where `devicename` is the onboard IO module name. For example, `Conn1_DIO4-11/freqout`.

Each counter has four primary terminals associated with it. These are the terminals used as the SOURCE, GATE, AUX, and OUT functions. NI-DAQmx has default values for these terminals. For counter input tasks, if you know whether your signal provides the SOURCE, AUX, or GATE function and wire your signal to the default, you do not have to set the Input Terminal attribute/property.

Counter/timers are supported on the DIO 0-3, DIO 4-11, DIO 12-19 and DIO 20-27 onboard DIO devices and on parallel digital I/O modules. Refer to **_Digital I/O Considerations for C Series_** and the counter section in **_C Series Physical Channels_** for more information about C Series device specific behavior.

The following table shows the counter terminal defaults for the DIO 0-3 onboard device.

Table 1. DIO 0-3 onboard device

PFI Signal	Physical Channel Name
PFI 0	`ctr0 src/ctr0 out/ctr2 aux/ctr3 gate`
PFI 1	`ctr0 gate/ctr1 aux/ctr2 src/ctr2 out/freqout`
PFI 2	`ctr0 aux/ctr1 gate/ctr3 src/ctr3 out`
PFI 3	`ctr1 src/ctr1 out/ctr2 gate/ctr3 aux`

The following table shows the counter terminal defaults for the DIO 4-11 onboard device.

Table 2. DIO 4-11 onboard device

PFI Signal	Physical Channel Name
PFI 4	`ctr0 src/ctr2 aux`
PFI 5	`ctr0 gate/ctr2 out`
PFI 6	`ctr0 aux/ctr2 gate/freqout`
PFI 7	`ctr0 out/ctr2 src`

PFI Signal	Physical Channel Name
PFI 8	ctr1 src/ctr3 aux
PFI 9	ctr1 gate/ctr3 out
PFI 10	ctr1 aux/ctr3 gate
PFI 11	ctr1 out/ctr3 src

The following table shows the counter terminal defaults for the DIO 12-19 onboard device.

Table 3. DIO 12-19 onboard device.

PFI Signal	Physical Channel Name
PFI 12	ctr0 src/ctr2 aux
PFI 13	ctr0 gate/ctr2 out
PFI 14	ctr0 aux/ctr2 gate/freqout
PFI 15	ctr0 out/ctr2 src
PFI 16	ctr1 src/ctr3 aux
PFI 17	ctr1 gate/ctr3 out
PFI 18	ctr1 aux/ctr3 gate
PFI 19	ctr1 out/ctr3 src

The following table shows the counter terminal defaults for the DIO 20-27 onboard device

Table 4. DIO 20-27 onboard device

PFI Signal	Physical Channel Name
PFI 20	ctr0 src/ctr2 aux
PFI 21	ctr0 gate/ctr2 out
PFI 22	ctr0 aux/ctr2 gate/freqout
PFI 23	ctr0 out/ctr2 src
PFI 24	ctr1 src/ctr3 aux

PFI Signal	Physical Channel Name
PFI 25	`ctr1 gate/ctr3 out`
PFI 26	`ctr1 aux/ctr3 gate`
PFI 27	`ctr1 out/ctr3 src`

E Series Physical Channels

`Dev1` in physical channel names is the default device name for E Series devices. You can change these names in MAX.

Related reference:

- [AO Series, E Series, and S Series Signal Connections for Counters](#)

Analog Input

A 16-channel E Series device has physical channels ranging from `Dev1/ai0` to `Dev1/ai15`. You can configure only channels 0 through 7 in differential mode. When you configure a channel in differential mode, the channel is the positive input and channel plus eight is the negative input. For instance, if you configure channel 1 in differential mode, the positive input is channel 1, and channel 9 is the negative input.

A 64-channel E Series device has physical channels ranging from `Dev1/ai0` to `Dev1/ai63`. You can configure channels in banks of every other eight beginning with 0 through 7 as the positive channel of a differential pair (0-7, 16-23, 32-39, and 48-55). If N is this channel, channel N + 8 is the negative input of the pair.

Use only the physical channel name of the positive channel when creating a differential channel (not both).

Analog Output

An E Series device that supports analog output has two analog output physical channels named `Dev1/ao0` and `Dev1/ao1`.

Digital Input and Output

All E Series devices except the NI 6025E have eight lines of digital input and output named `Dev1/port0/line0` through `Dev1/port0/line7`. These lines belong to a single port, and the physical channel Dev1/port0 refers to all eight lines at once.

The NI 6025E has 32 lines of digital input and output with eight lines belonging to one of four ports. The names are of the form `Dev1/portP/line0` through `Dev1/portP/line7`, where P ranges from 0 through 3. There are also four physical channel names that refer to all eight lines in a port at once of the form `Dev1/portP`, where P ranges from 0 through 3. There are two more physical channel names that refer to all the lines in multiple consecutive ports. They are both of the form `Dev1/portP_N`, where P is the port number of the lowest numbered port, and N is the total number of lines. All 32 lines at once can be configured as a single virtual channel with the physical channel name `Dev1/port0_32`. You can configure the two ports that can be handshaked as a single virtual channel by using the physical channel name `Dev1/port1_16`.

Counter Input and Output

All E Series devices have two counter/timers referred to by the physical channel names `Dev1/ctr0` and `Dev1/ctr1`. Unlike the other I/O types, these physical channel names do not refer to terminals on the I/O connector but instead to circuits within the device. There are three primary terminals associated with each counter. These are the terminals used as the SOURCE, GATE and OUT functions. NI-DAQmx has default values for these terminals. For counter input tasks, if you know whether your signal provides the SOURCE or GATE function and wire your signal to the default, you do not have to set the Input Terminal attribute/property.

Counter	SOURCE Default	GATE Default	OUT Default
Dev1/ctr0	PFI 8	PFI 9	CTR 0 OUT
Dev/ctr1	PFI 3	PFI 4	CTR 1 OUT

FieldDAQ Physical Channels

In physical channel names, <product name>-<serial#> is the default device name. For

example, an FD-11613 with serial number 1C6A39C shows up in MAX as `FD11613-1C6A39C`. MAX also appends the bank number. For example, `FD11613-1C6A39C-Bank1`. You can change these names in MAX.

FieldDAQ devices include the FD-11601, FD-11603, FD-11605, FD-11613, FD-11614, FD-11634, and FD-11637.

Analog Input

The following table lists the number and naming of analog input physical channels for FieldDAQ devices.

Device	Number of Channels	Naming

FD-11601	8	`FD-11601-xxxxxxx-Bank1/ai0` to `FD-11601-xxxxxxx-Bank1/ai3`
		
		`FD-11601-xxxxxxx-Bank2/ai0` to `FD-11601-xxxxxxx-Bank2/ai3`

FD-11603	8	`FD-11603-xxxxxxx-Bank1/ai0` to `FD-11603-xxxxxxx-Bank1/ai3`
		
		`FD-11603-xxxxxxx-Bank2/ai0` to `FD-11603-xxxxxxx-Bank2/ai3`

FD-11605	8	`FD-11605-xxxxxxx-Bank1/ai0` to `FD-11605-xxxxxxx-Bank1/ai3`
		
		`FD-11605-xxxxxxx-Bank2/ai0` to `FD-11605-xxxxxxx-Bank2/ai3`

FD-11613	8	`FD-11613-xxxxxxx-Bank1/ai0` to `FD-11613-xxxxxxx-Bank1/ai7`
FD-11614	16	`FD-11614-xxxxxxx-Bank1/ai0` to `FD-11614-xxxxxxx-Bank1/ai7` `FD-11614-xxxxxxx-Bank2/ai0` to

Device	Number of Channels	Naming
		`FD-11614-xxxxxxx-Bank2/ai7`

FD-11634	8	`FD-11634-xxxxxxx-Bank1/ai0` to `FD-11634-xxxxxxx-Bank1/ai3`
		
		`FD-11634-xxxxxxx-Bank2/ai0` to `FD-11634-xxxxxxx-Bank2/ai3`

FD-11637	8	`FD-11637-xxxxxxx-Bank1/ai0` to `FD-11637-xxxxxxx-Bank1/ai3`
		
		`FD-11637-xxxxxxx-Bank2/ai0` to `FD-11637-xxxxxxx-Bank2/ai3`

Strain and Wheatstone Bridge Measurements FD-11637

The FD-11637 supports only the AI Strain Gage, AI Rosette Strain Gage, AI Force Bridge, AI Pressure Bridge, AI Torque Bridge, AI Bridge (V/V), and AI Custom Voltage With Excitation channel types.

When using the FD-11637 with an AI Custom Voltage With Excitation channel, you must set the AI.Excit.UseForScaling attribute/property to true. This attribute/property causes the channel to return ratiometric data: Vin/Vex.

The FD-11637 returns a voltage ratio rather than a voltage. Therefore, use the AI.Bridge.InitialRatio attribute/property to specify the initial voltage ratio, or set the AI.Bridge.InitialVoltage attribute/property to the ratio Vin/Vex returned by the device, multiplied by Vex.

When the FD-11637 is in quarter bridge mode, you need to use the AI.Bridge.NomResistance attribute/property to control whether the channel uses the 120 Ω range or the 350 Ω range.

The FD-11637 has internal half-bridge completion, and quarter-bridge completion for 120 Ω and 350 Ω strain gauges.

M Series

The section contains information about M Series Physical Channels, Bus-Powered M Series Physical Channels, NI 6221 (37-Pin) Device Physical Channels, and NI 623x Physical Channels.

Related concepts:

- M Series Physical Channels
- Bus-Powered M Series Physical Channels
- NI 6221 (37-Pin) Device Physical Channels
- NI 623x Physical Channels
- Self-Powered Compared to Bus-Powered USB Devices
- Self-Powered Versus Bus-Powered M Series USB Devices

M Series Physical Channels

In physical channel names, `Dev1` is the default device name for M Series devices. You can change these names in MAX.

Related reference:

- 68-Pin M Series Signal Connections for Counters

Analog Input

Depending on your M Series device, you can have from 16 to 80 analog input channels. A 16-channel M Series device has physical channels ranging from `Dev1/ai0` to `Dev1/ai15`, a 32-channel device from `Dev1/ai0` to `Dev1/ai31`, and so on. You can configure the first eight channels as the positive channel of a differential pair. If N is this channel, channel N + 8 is the negative input of the pair. For instance, if you configure channel 1 in differential mode, the positive input is channel 1, and channel 9 is the negative input. For devices with more than 16 AI channels, 16-23, 32-39, 48-55, and 64-71 are also positive channels of a differential pair.

Use only the physical channel name of the positive channel (not both) when creating a differential channel.

Analog Output

An M Series device that supports two analog outputs has analog output physical channels named `Dev1/ao0` and `Dev1/ao1`.

An M Series device that supports four analog outputs has analog output physical channels named `Dev1/ao0`, `Dev1/ao1`, `Dev1/ao2`, and `Dev1/ao3`.

Digital Input and Output

All M Series devices have eight, 16, or 32 lines of digital input and output named `Dev1/port0/line0` through `Dev1/port0/line7`, `Dev1/port0/line0` through `Dev1/port0/line15`, or `Dev1/port0/line0` through `Dev1/port0/line31`. These lines belong to a single port, and the physical channel `Dev1/port0` refers to all eight or 32 lines at once. Port 0 can perform both hardware-timed and static digital operations.

M Series devices have two more ports, port 1 and port 2. Port 1 has eight digital I/O lines, `Dev1/port1/line0` through `Dev1/port1/line7`. Port 2 has eight digital I/O lines, `Dev1/port2/line0` through `Dev1/port2/line7`. Port 1 and port 2 can be used as static digital I/O lines or PFI lines, PFI 0..15. When any of PFI lines 0..15 is used as a digital I/O signal, it uses the physical channel name shown in the following table.

PFI Signal	Physical Channel Name
PFI 0	`Dev1/port1/line0`
PFI 1	`Dev1/port1/line1`
PFI 2	`Dev1/port1/line2`
PFI 3	`Dev1/port1/line3`
PFI 4	`Dev1/port1/line4`
PFI 5	`Dev1/port1/line5`
PFI 6	`Dev1/port1/line6`
PFI 7	`Dev1/port1/line7`
PFI 8	`Dev1/port2/line0`

PFI Signal	Physical Channel Name
PFI 9	`Dev1/port2/line1`
PFI 10	`Dev1/port2/line2`
PFI 11	`Dev1/port2/line3`
PFI 12	`Dev1/port2/line4`
PFI 13	`Dev1/port2/line5`
PFI 14	`Dev1/port2/line6`
PFI 15	`Dev1/port2/line7`

Physical channel `Dev1/port2` refers to all eight lines, `Dev1/port2/line0:7`, at once.

Counter Input and Output

All M Series devices have two counter/timers referred to by the physical channel names `Dev1/ctr0` and `Dev1/ctr1` . Unlike the other I/O types, these physical channel names do not refer to terminals on the I/O connector but instead to circuits within the device. There are four primary terminals associated with each counter. These are the terminals used as the SOURCE, GATE, AUX, and OUT functions. NI-DAQmx has default values for these terminals. For counter input tasks, if you know whether your signal provides the SOURCE, AUX, or GATE function and wire your signal to the default, you do not have to set the Input Terminal attribute/property.

Counter	SOURCE Default	GATE Default	AUX Default	OUT Default
Counter	SOURCE Default	GATE Default	AUX Default	OUT Default
Dev1/ctr0	PFI 8	PFI 9	PFI 10	PFI 12
Dev1/ctr1	PFI 3	PFI 4	PFI 11	PFI 13

Bus-Powered M Series Physical Channels

In physical channel names, `Dev1` is the default device name for M Series devices. You can change these names in MAX.

Related concepts:

- [Self-Powered Compared to Bus-Powered USB Devices](#)
- [Self-Powered Versus Bus-Powered M Series USB Devices](#)

Related reference:

- [Bus-Powered M Series Signal Connections for Counters](#)

Analog Input

Depending on your M Series device, you can have from 16 to 32 analog input channels. A 16-channel M Series device has physical channels ranging from `Dev1/ai0` to `Dev1/ai15`, a 32-channel device from `Dev1/ai0` to `Dev1/ai31`, and so on. You can configure the first eight channels as the positive channel of a differential pair. If N is this channel, channel N + 8 is the negative input of the pair. For instance, if you configure channel 1 in differential mode, the positive input is channel 1, and channel 9 is the negative input. For devices with more than 16 AI channels, 16-23, 32-39, 48-55, and 64-71 are also positive channels of a differential pair.

Use only the physical channel name of the positive channel (not both) when creating a differential channel.

Analog Output

An M Series device that supports two analog outputs has analog output physical channels named `Dev1/ao0` and `Dev1/ao1`.

An M Series device that supports four analog outputs has analog output physical channels named `Dev1/ao0`, `Dev1/ao1`, `Dev1/ao2`, and `Dev1/ao3`.

Digital Input and Output

Most bus-powered M Series devices have two ports, port 0 and port 1. For devices with eight PFI lines, Port 0 has four digital input lines, `Dev1/port0/line0` through `Dev1/port0/line3`, and port 1 has four digital output lines, `Dev1/port1/line0` through `Dev1/port1/line3`. For devices with 16 PFI lines such as the NI 6218, Port 0 has eight digital input lines, `Dev1/port0/line0` through `Dev1/`

`port0/line7`, and port 1 has eight digital output lines, `Dev1/port1/line0` through `Dev1/port1/line7`. You can use port 0 as static digital input lines or input PFI lines. You can use port 1 as static digital output lines or output PFI lines. When any of PFI lines 0..15 is used as a digital I/O signal, it uses the physical channel name shown in the following table.

PFI Signal	Physical Channel Name
PFI 0	`Dev1/port0/line0`
PFI 1	`Dev1/port0/line1`
PFI 2	`Dev1/port0/line2`
PFI 3	`Dev1/port0/line3`
PFI 4	`Dev1/port1/line0`
PFI 5	`Dev1/port1/line1`
PFI 6	`Dev1/port1/line2`
PFI 7	`Dev1/port1/line3`
PFI 8	`Dev1/port0/line4`
PFI 9	`Dev1/port0/line5`
PFI 10	`Dev1/port0/line6`
PFI 11	`Dev1/port0/line7`
PFI 12	`Dev1/port1/line4`
PFI 13	`Dev1/port1/line5`
PFI 14	`Dev1/port1/line6`
PFI 15	`Dev1/port1/line7`

Physical channel `Dev1/port0` refers to all four or eight lines, `Dev1/port1/line0:3` or `Dev1/port1/line0:7`, at once. Physical channel `Dev1/port1` refers to all four or eight lines, `Dev1/port1/line0:3` or `Dev1/port1/line0:7`, at once.

The NI 6212 and NI 6216 have 16 lines of digital input and output named `Dev1/port0/line0` through `Dev1/port0/line15`. These lines belong to a single port,

and the physical channel `Dev1/port0` refers to all 16 lines at once. Port 0 can perform static digital I/O operations only.

The NI 6212 and NI 6216 have two more ports, port 1 and port 2. Port 1 has eight digital I/O lines, `Dev1/port1/line0` through `Dev1/port1/line7`. Port 2 has eight digital I/O lines, `Dev1/port2/line0` through `Dev1/port2/line7`. Port 1 and port 2 can be used as static digital I/O lines or PFI lines, PFI 0..15. When any of PFI lines 0..15 is used as a digital I/O signal, it uses the physical channel name shown in the following table.

PFI Signal	Physical Channel Name
PFI 0	`Dev1/port1/line0`
PFI 1	`Dev1/port1/line1`
PFI 2	`Dev1/port1/line2`
PFI 3	`Dev1/port1/line3`
PFI 4	`Dev1/port1/line4`
PFI 5	`Dev1/port1/line5`
PFI 6	`Dev1/port1/line6`
PFI 7	`Dev1/port1/line7`
PFI 8	`Dev1/port2/line0`
PFI 9	`Dev1/port2/line1`
PFI 10	`Dev1/port2/line2`
PFI 11	`Dev1/port2/line3`
PFI 12	`Dev1/port2/line4`
PFI 13	`Dev1/port2/line5`
PFI 14	`Dev1/port2/line6`
PFI 15	`Dev1/port2/line7`

Physical channel `Dev1/port1` refers to all eight lines, `Dev1/port1/line0:7`, at once. Physical channel `Dev1/port2` refers to all eight lines, `Dev1/port2/line0:7`, at once.

Counter Input and Output

All M Series devices have two counter/timers referred to by the physical channel names Dev1/ctr0 and Dev1/ctr1. Unlike the other I/O types, these physical channel names do not refer to terminals on the I/O connector but instead to circuits within the device. There are four primary terminals associated with each counter. These are the terminals used as the SOURCE, GATE, AUX, and OUT functions. NI-DAQmx has default values for these terminals. For counter input tasks, if you know whether your signal provides the SOURCE, AUX, or GATE function and wire your signal to the default, you do not have to set the Input Terminal attribute/property.

16-PFI Line Devices (NI 6218)

Counter	SOURCE Default	GATE Default	AUX Default	OUT Default
Dev1/ctr0	PFI 0	PFI 1	PFI 9	PFI 4
Dev1/ctr1	PFI 3	PFI 2	PFI 10	PFI 5

16-PFI Line Devices (NI 6212/6216)

Counter	SOURCE Default	GATE Default	AUX Default	OUT Default
Dev1/ctr0	PFI 8	PFI 9	PFI 10	PFI 12
Dev1/ctr1	PFI 3	PFI 4	PFI 11	PFI 13

8-PFI Line Devices (Such as the NI 6210/6211/6215)

Counter	SOURCE Default	GATE Default	AUX Default	OUT Default
Dev1/ctr0	PFI 0	PFI 1	PFI 0	PFI 4
Dev1/ctr1	PFI 3	PFI 2	PFI 3	PFI 5

NI 6221 (37-Pin) Device Physical Channels

In physical channel names, `Dev1` is the default device name for NI 6221 (37-pin) devices. You can change these names in MAX.

Analog Input

A 16-channel NI 6221 (37-pin) device has physical channels ranging from `Dev1/ai0` to `Dev1/ai15`. You can configure only channels 0 through 7 in differential mode. When you configure a channel in differential mode, the channel is the positive input and channel plus eight is the negative input. For instance, if you configure channel 1 in differential mode, the positive input is channel 1, and channel 9 is the negative input.

Analog Output

NI 6221 (37-pin) devices have two analog outputs corresponding to two analog output physical channels named `Dev1/ao0` and `Dev1/ao1`.

Digital Input and Output

NI 6221 (37-pin) devices have two ports, port 0 and port 1. Port 0 has two digital I/O lines, `Dev1/port0/line0` and `Dev1/port0/line1`. Port 1 has eight digital I/O lines, `Dev1/port1/line0` through `Dev1/port1/line7`. Port 1 can be used as static digital I/O lines or input PFI lines, PFI 0..7. When any of PFI lines 0..7 is used as a digital I/O signal, it uses the physical channel name shown in the following table.

PFI Signal	Physical Channel Name
PFI 0	`Dev1/port1/line0`
PFI 1	`Dev1/port1/line1`
PFI 2	`Dev1/port1/line2`
PFI 3	`Dev1/port1/line3`
PFI 4	`Dev1/port1/line4`
PFI 5	`Dev1/port1/line5`
PFI 6	`Dev1/port1/line6`
PFI 7	`Dev1/port1/line7`

Physical channel `Dev1/port0` refers to both lines, `Dev1/port0/line0:1`, at once. Physical channel `Dev1/port1` refers to all eight lines, `Dev1/port1/line0:7`, at once.

Counter Input and Output

NI 6221 (37-pin) devices have two counter/timers referred to by the physical channel names `Dev1/ctr0` and `Dev1/ctr1`. Unlike the other I/O types, these physical channel names do not refer to terminals on the I/O connector but instead to circuits within the device. There are four primary terminals associated with each counter. These are the terminals used as the SOURCE, GATE, AUX, and OUT functions. NI-DAQmx has default values for these terminals. For counter input tasks, if you know whether your signal provides the SOURCE, AUX, or GATE function and wire your signal to the default, you do not have to set the Input Terminal attribute/property.

Counter	SOURCE Default	GATE Default	AUX Default	OUT Default
Dev1/ctr0	PFI 0	PFI 1	PFI 2	PFI 6
Dev1/ctr1	PFI 3	PFI 4	PFI 5	PFI 7

In addition, the NI 6221 (37-pin) has one frequency generator. The output terminal of the frequency generator is FREQOUT. The default for FREQOUT is PFI 5. When using FREQOUT, you can continue to use both ctr0 and ctr1 to perform other operations.

NI 623*x* Physical Channels

In physical channel names, `Dev1` is the default device name for NI 623x devices. You can change these names in MAX.

Related reference:

- 37-Pin DSUB Signal Connections for Counters

Analog Input

Refer to the following table for the physical channel naming conventions for each 623x device.

Device	Differential		RSE	
	+	**-**	**+**	**-**
NI 6230	`Dev1/ai` n	`Dev1/ai` (n+4)	`Dev1/ai0..Dev1/ai` n, where n=0..3	AI Gnd
NI 6232	`Dev1/ai` n	`Dev1/ai` (n+8)	`Dev1/ai0..Dev1/ai` n, where n=0..7	AI Gnd
NI 6233	`Dev1/ai` n	`Dev1/ai` (n+8)	`Dev1/ai0..Dev1/ai` n, where n=0..7	AI Gnd
NI 6236	`Dev1/ai` n+	`Dev1/ai` n-	`Dev1/ai0..Dev1/ai` n, where n=0..3	AI Gnd
NI 6238	`Dev1/ai` n+	`Dev1/ai` n-	`Dev1/ai0..Dev1/ai` n, where n=0..7	AI Gnd
NI 6239	`Dev1/ai` n+	`Dev1/ai` n-	`Dev1/ai0..Dev1/ai` n, where n=0..7	AI Gnd

Analog Output

NI 623x devices have m analog outputs corresponding to m analog output physical channels ranging from `Dev1/ao0` to `Dev1/ao(m-1)`. Refer to the hardware documentation for the number of analog outputs for your device.

Digital Input and Output

NI 623x devices have two ports, port 0 and port 1. Port 0 has six digital input lines, `Dev1/port0/line0` through `Dev1/port0/line5`. Port 1 has four digital output lines, `Dev1/port1/line0` through `Dev1/port1/line3`. You can use port 0 as static digital input lines or input PFI lines, PFI 0..5. You can use port 1 as static digital output lines or output PFI lines, PFI 6..9. When any of PFI lines 0..9 is used as a digital I/O signal, it uses the physical channel name shown in the following table.

PFI Signal	Physical Channel Name	Direction
PFI 0	`Dev1/port0/line0`	Input
PFI 1	`Dev1/port0/line1`	Input
PFI 2	`Dev1/port0/line2`	Input
PFI 3	`Dev1/port0/line3`	Input
PFI 4	`Dev1/port0/line4`	Input
PFI 5	`Dev1/port0/line5`	Input
PFI 6	`Dev1/port1/line0`	Output

PFI Signal	Physical Channel Name	Direction
PFI 7	`Dev1/port1/line1`	Output
PFI 8	`Dev1/port1/line2`	Output
PFI 9	`Dev1/port1/line3`	Output

Physical channel `Dev1/port0` refers to all six input lines, `Dev1/port0/line0:5`, at once. Physical channel `Dev1/port1` refers to all four output lines, `Dev1/port1/line0:3`, at once.

Tristating Digital Output Channels (NI 6230/6236)

NI 6230/6236 devices support tristating for port 1, the four digital output lines. The power-on state default is to tristate port 1. Tristating is supported only for the entire port at a time, not on a per-line basis. For instance, port 1 remains tristated as long as no lines on port 1 are toggled to generate a value. After a line on port 1 is toggled, all lines on the port are driven to logic high or logic low depending on what you choose. The default is logic low.

Counter Input and Output

NI 623x devices have two counter/timers referred to by the physical channel names `Dev1/ctr0` and `Dev1/ctr1`. Unlike the other I/O types, these physical channel names do not refer to terminals on the I/O connector but instead to circuits within the device. There are four primary terminals associated with each counter. These are the terminals used as the SOURCE, GATE, AUX, and OUT functions. NI-DAQmx has default values for these terminals. For counter input tasks, if you know whether your signal provides the SOURCE, AUX, or GATE function and wire your signal to the default, you do not have to set the Input Terminal attribute/property.

Counter	SOURCE Default	GATE Default	AUX Default	OUT Default
Dev1/ctr0	PFI 0	PFI 1	PFI 2	PFI 6
Dev1/ctr1	PFI 3	PFI 4	PFI 5	PFI 7

In addition, NI 623x devices have one frequency generator. The output terminal of the frequency generator is FREQOUT. The default for FREQOUT is PFI 8. When using

FREQOUT, you can continue to use both ctr0 and ctr1 to perform other operations.

NI 623x devices use the same default terminals for common counter applications as other 37-Pin DSUB devices.

myDAQ Physical Channels

In physical channel names, `myDAQ1` is the default device name for myDAQ. You can change these names in MAX.

Related reference:

- [myDAQ Signal Connections for Counters](#)

Analog Input

myDAQ has two analog input channels, `myDAQ1/ai0` and `myDAQ1/ai1`. These channels support only the differential terminal configuration. You can configure the first channel, `myDAQ1/ai0`, as the positive channel of a differential pair. Use only the physical channel name of the positive channel (not both) when creating a differential channel.

myDAQ also has a DMM physical channel, myDAQ1/dmm, available from the device. On myDAQ, the DMM physical channels cannot be used in the same task with other channels. There are also two audio input channels, myDAQ1/audioInputLeft and myDAQ1/audioInputRight. These channels support only the referenced signal-ended terminal configuration. Analog input tasks do not work simultaneously with audio input.

Analog Output

myDAQ has two analog outputs with analog output physical channels named `myDAQ1/ao0` and `myDAQ1/ao1`. myDAQ also has two audio output channels, `myDAQ1/audioOutputLeft` and `myDAQ1/audioOutputRight`. Analog output tasks do not work simultaneously with audio out.

Digital Input and Output

myDAQ has eight lines of digital input and output named `myDAQ1/port0/line0` through `myDAQ1/port0/line7`. These lines belong to a single port, and the physical channel `myDAQ1/port0` refers to all 8 lines at once.

Counter Input and Output

myDAQ has one counter/timer referred to by the physical channel name `myDAQ1/ctr0`. Unlike the other I/O types, these physical channel names do not refer to terminals on the I/O connector but instead to circuits within the device. There are four primary terminals associated with each counter. These are the terminals used as the SOURCE, GATE, AUX, and OUT functions. NI-DAQmx has default values for these terminals. For counter input tasks, if you know whether your signal provides the SOURCE, AUX, or GATE function and wire your signal to the default, you do not have to set the Input Terminal attribute/property.

Counter	SOURCE Default	GATE Default	AUX Default	OUT Default
myDAQ1/ctr0	PFI 0 (DIO 0)	PFI 1 (DIO 1)	PFI 2 (DIO 2)	PFI 3 (DIO 3)

NI 6010 Physical Channels

In physical channel names, `Dev1` is the default device name for NI 6010 devices. You can change these names in MAX.

Related reference:

- [37-Pin DSUB Signal Connections for Counters](#)

Analog Input

A 16-channel NI 6010 device has physical channels ranging from `Dev1/ai0` to `Dev1/ai15`. You can configure only channels 0 through 7 in differential mode. When you configure a channel in differential mode, the channel is the positive input and channel plus eight is the negative input. For instance, if you configure channel 1 in differential mode, the positive input is channel 1, and channel 9 is the negative input.

Use only the physical channel name of the positive channel (not both) when creating a differential channel.

Analog Output

NI 6010 devices have two analog outputs corresponding to two analog output physical channels named `Dev1/ao0` and `Dev1/ao1`.

Digital Input and Output

NI 6010 devices have two ports, port 0 and port 1. Port 0 has six digital input lines, `Dev1/port0/line0` through `Dev1/port0/line5`. Port 1 has four digital output lines, `Dev1/port1/line0` through `Dev1/port1/line3`. You can use port 0 as static digital input lines or input PFI lines, PFI 0..5. You can use port 1 as static digital output lines or output PFI lines, PFI 6..9. When any of PFI lines 0..9 is used as a digital I/O signal, it uses the physical channel name shown in the following table.

PFI Signal	Physical Channel Name
PFI 0	`Dev1/port0/line0`
PFI 1	`Dev1/port0/line1`
PFI 2	`Dev1/port0/line2`
PFI 3	`Dev1/port0/line3`
PFI 4	`Dev1/port0/line4`
PFI 5	`Dev1/port0/line5`
PFI 6	`Dev1/port1/line0`
PFI 7	`Dev1/port1/line1`
PFI 8	`Dev1/port1/line2`
PFI 9	`Dev1/port1/line3`

Physical channel `Dev1/port0` refers to all six input lines, `Dev1/port0/line0:5`, at once. Physical channel `Dev1/port1` refers to all four output lines, `Dev1/port1/line0:3`, at once.

Counter Input and Output

NI 6010 devices have two counter/timers referred to by the physical channel names `Dev1/ctr0` and `Dev1/ctr1`. Unlike the other I/O types, these physical channel names do not refer to terminals on the I/O connector but instead to circuits within the device. There are four primary terminals associated with each counter. These are the terminals used as the SOURCE, GATE, AUX, and OUT functions. NI-DAQmx has default values for these terminals. For counter input tasks, if you know whether your signal provides the SOURCE, AUX, or GATE function and wire your signal to the default, you do not have to set the Input Terminal attribute/property.

Counter	SOURCE Default	GATE Default	AUX Default	OUT Default
Dev1/ctr0	PFI 0	PFI 1	PFI 2	PFI 6
Dev1/ctr1	PFI 3	PFI 4	PFI 5	PFI 7

NI 6154 Physical Channels

In physical channel names, `Dev1` is the default device name for NI 6154 devices. You can change these names in MAX.

Related reference:

- [37-Pin DSUB Signal Connections for Counters](#)

Analog Input

A 4-channel NI 6154 device has physical channels ranging from `Dev1/ai0` to `Dev1/ai3`.

Analog Output

NI 6154 devices have four analog outputs corresponding to four analog output physical channels ranging from `Dev1/ao0` to `Dev1/ao3`.

Digital Input and Output

NI 6154 devices have two ports, port 0 and port 1. Port 0 has six digital input lines, `Dev1/port0/line0` through `Dev1/port0/line5`. Port 1 has four digital output lines, `Dev1/port1/line0` through `Dev1/port1/line3`. You can use port 0 as static digital input lines or input PFI lines, PFI 0..5. You can use port 1 as static digital output lines or output PFI lines, PFI 6..9. When any of PFI lines 0..9 is used as a digital I/O signal, it uses the physical channel name shown in the following table.

PFI Signal	Physical Channel Name	Direction
PFI 0	`Dev1/port0/line0`	Input
PFI 1	`Dev1/port0/line1`	Input
PFI 2	`Dev1/port0/line2`	Input
PFI 3	`Dev1/port0/line3`	Input
PFI 4	`Dev1/port0/line4`	Input
PFI 5	`Dev1/port0/line5`	Input
PFI 6	`Dev1/port1/line0`	Output
PFI 7	`Dev1/port1/line1`	Output
PFI 8	`Dev1/port1/line2`	Output
PFI 9	`Dev1/port1/line3`	Output

Physical channel `Dev1/port0` refers to all six input lines, `Dev1/port0/line0:5`, at once. Physical channel `Dev1/port1` refers to all four output lines, `Dev1/port1/line0:3`, at once.

Counter Input and Output

NI 6154 devices have two counter/timers referred to by the physical channel names `Dev1/ctr0` and `Dev1/ctr1`. Unlike the other I/O types, these physical channel names do not refer to terminals on the I/O connector but instead to circuits within the device. There are four primary terminals associated with each counter. These are the terminals used as the SOURCE, GATE, AUX, and OUT functions. NI-DAQmx has default values for these terminals. For counter input tasks, if you know whether your signal

provides the SOURCE, AUX, or GATE function and wire your signal to the default, you do not have to set the Input Terminal attribute/property.

Counter	SOURCE Default	GATE Default	AUX Default	OUT Default
Dev1/ctr0	PFI 0	PFI 1	PFI 2	PFI 6
Dev1/ctr1	PFI 3	PFI 4	PFI 5	PFI 7

In addition, the NI 6154 has one frequency generator. The output terminal of the frequency generator is FREQOUT. The default for FREQOUT is PFI 8. When using FREQOUT, you can continue to use both ctr0 and ctr1 to perform other operations.

The NI 6154 uses the same default terminals for common counter applications as other 37-Pin DSUB devices.

NI 6533/6534 Device Physical Channels

Digital Input and Output

All NI 6533/6534 devices have 32 individually configurable lines of digital input and output that are grouped into four 8-bit ports.

Port	NI-DAQmx Physical Channel Name (Lines)	NI-DAQmx Physical Channel Name (Ports)[34]
Port 0	`Dev1/port0/line0 — Dev1/port0/line7`	`Dev1/port0`
Port 1	`Dev1/port1/line0 — Dev1/port1/line7`	`Dev1/port1`
Port 2	`Dev1/port2/line0 — Dev1/port2/line7`	`Dev1/port2`
Port 3	`Dev1/port3/line0 — Dev1/port3/line7`	`Dev1/port3`

For Ports 0 through 3, you can configure a port width of eight, 16, or 32 bits. To

34. This physical channel name refers to all eight lines in a port at once.

configure a 32-bit port, use the physical channel name Dev1/port0_32. To configure a 16-bit port, use channel names that refer to all the lines in multiple consecutive ports: `Dev1/port`P_N, where P is the port number of the lowest numbered port, and N is the total number of lines. For instance, combining Port 2 and 3 into a 16-bit port, you would specify `Dev1/port2_16` as the physical channel.

NI 6533/6534 devices also have eight fixed-direction lines, grouped into two ports, that use PFI lines. Port 4 is used for input operations; Port 5 is for output.

Port 4 and port 5 can be used as static digital I/O lines or PFI lines. When any of these PFI lines is used as a digital I/O signal, it uses the physical channel name shown in the following table.

PFI Signal	Physical Channel Name
PFI0	Dev1/port4/line0
PFI1	Dev1/port4/line1
PFI2	Dev1/port4/line2
PFI3	Dev1/port4/line3
PFI4	Dev1/port5/line2
PFI5	Dev1/port5/line3
PFI6	Dev1/port5/line0
PFI7	Dev1/port5/line1

NI 6535/6536/6537 Physical Channels

The 32 individually configurable lines of digital input and output on the NI 6535/6536/6537 are grouped into four 8-bit ports, as shown in the following table. You can choose to configure an entire port for a particular task or only a particular physical channel.

The following table shows the channel names you can use to configure your task in NI-DAQmx. Dev1 in physical channel names is the default device name for NI 6535/6536/6537 devices. You can change the device name in MAX.

Port	NI-DAQmx Physical Channel Name (Lines)	NI-DAQmx Physical Channel Name (Ports)[35]
Port 0	`Dev1/port0/line0 — Dev1/port0/line7`	`Dev1/port0`
Port 1	`Dev1/port1/line0 — Dev1/port1/line7`	`Dev1/port1`
Port 2	`Dev1/port2/line0 — Dev1/port2/line7`	`Dev1/port2`
Port 3	`Dev1/port3/line0 — Dev1/port3/line7`	`Dev1/port3`
Port 4[36]	`Dev1/port4/line0 — Dev1/port4/line5`	`Dev1/port4`

For Ports 0 through 3, you can configure a port width of 8, 16, or 32 bits. To configure a 32-bit port, use the physical channel name `Dev1/port0_32`. To configure a 16-bit port, use channel names that refer to all the lines in multiple consecutive ports: `Dev1/port`P_N, where P is 0 or 2 (whichever port number is lower) and N is the total number of lines. For example, to combine Port 2 and 3 into a 16-bit port, specify `Dev1/port2_16` as the physical channel.

Using NI-DAQmx, you can configure each physical channel individually by inverting or tristating the physical channel.

Related concepts:

- Sample Timing Types

PFI Lines

The NI 6535/6536/6537 has six PFI lines. These bidirectional digital lines allow you to route synchronization and timing signals to and from the I/O connector. The following table lists the PFI lines and their timing function.

PFI 0	General-purpose PFI line

---

35. This physical channel name refers to all eight lines in a port at once.
36. Port 4 is composed of the six PFI lines.

PFI 1	General-purpose PFI line
PFI 2	General-purpose PFI line
PFI 3	General-purpose PFI line
PFI 4	General-purpose PFI or the generation Sample clock terminal
PFI 5	General-purpose PFI or the acquisition Sample clock terminal

You can use control lines as extra data lines while using the On Demand sample timing type.

Related concepts:

- Sample Timing Types

NI ELVIS II Family Physical Channels

In physical channel names, `Dev1` is the default device name for the NI ELVIS II Family device. You can change these names in MAX.

Related reference:

- NI ELVIS II Family Signal Connections for Counters

Analog Input

NI ELVIS II Family devices have 16 analog input channels, ranging from `Dev1/ai0` to `Dev1/ai15`. You can configure the first eight channels as the positive channel of a differential pair. If N is this channel, channel N + 8 is the negative input of the pair. For instance, if you configure channel 1 in differential mode, the positive input is channel 1, and channel 9 is the negative input.

Use only the physical channel name of the positive channel (not both) when creating a differential channel.

NI ELVIS II Family devices have two oscilloscope physical channels, `Dev1/scopeCh0` and `Dev1/scopeCh1`, available from the NI ELVIS II Family benchtop workstation. The NI ELVIS II Family benchtop workstation also has a DMM physical channel, `Dev1/`

`dmm`. On the NI ELVIS II Family devices, the DMM and oscilloscope physical channels cannot be used in the same task with other channels. However, scopeCh0 and scopeCh1 can be used together.

Analog Output

NI ELVIS II Family devices have two analog outputs with analog output physical channels named `Dev1/ao0` and `Dev1/ao1`. NI ELVIS II Family devices also have two function generator channels, `Dev1/fgen` and `Dev1/fgenBNC`, and two variable power supply channels, `Dev1/vpsPos` (the positive variable power supply) and `Dev1/vpsNeg` (the negative variable power supply). The function generator channels, `Dev1/fgen` and `Dev1/fgenBNC`, use the FGEN Create Channel function/VI. These function generator terminals also share the same resource.

Digital Input and Output

NI ELVIS II Family devices have 24 lines of digital input and output named `Dev1/port0/line0` through `Dev1/port0/line23`. These lines belong to a single port, and the physical channel `Dev1/port0` refers to all 24 lines at once. Port 0 can perform static digital operations.

NI ELVIS II Family devices have two more ports, port 1 and port 2. Port 1 has eight digital I/O lines, `Dev1/port1/line0` through `Dev1/port1/line7`. Port 2 has seven digital I/O lines, `Dev1/port2/line0` through `Dev1/port2/line6`. Port 1 and port 2 can be used as static digital I/O lines or PFI lines, PFI 0..14. When any of PFI lines 0..14 is used as a digital I/O signal, it uses the physical channel name shown in the following table.

PFI Signal	Physical Channel Name
PFI 0	`Dev1/port1/line0`
PFI 1	`Dev1/port1/line1`
PFI 2	`Dev1/port1/line2`
PFI 3	`Dev1/port1/line3`
PFI 4	`Dev1/port1/line4`
PFI 5	`Dev1/port1/line5`

PFI Signal	Physical Channel Name
PFI 6	`Dev1/port1/line6`
PFI 7	`Dev1/port1/line7`
PFI 8	`Dev1/port2/line0`
PFI 9	`Dev1/port2/line1`
PFI 10	`Dev1/port2/line2`
PFI 11	`Dev1/port2/line3`
PFI 12	`Dev1/port2/line4`
PFI 13	`Dev1/port2/line5`
PFI 14	`Dev1/port2/line6`

Physical channel Dev1/port1 refers to all eight lines, `Dev1/port1/line0:7`, at once. Physical channel Dev1/port2 refers to all seven lines, `Dev1/port2/line0:6`, at once.

Counter Input and Output

NI ELVIS II Family devices have two counter/timers referred to by the physical channel names `Dev1/ctr0` and `Dev1/ctr1`. Unlike the other I/O types, these physical channel names do not refer to terminals on the I/O connector but instead to circuits within the device. There are four primary terminals associated with each counter. These are the terminals used as the SOURCE, GATE, AUX, and OUT functions. NI-DAQmx has default values for these terminals. For counter input tasks, if you know whether your signal provides the SOURCE, AUX, or GATE function and wire your signal to the default, you do not have to set the Input Terminal attribute/property.

Counter	SOURCE Default	GATE Default	AUX Default	OUT Default
Dev1/ctr0	PFI 8	PFI 9	PFI 10	PFI 12
Dev1/ctr1	PFI 3	PFI 4	PFI 11	PFI 13

> ✏️ **Note** PFI 15 refers to the same terminal as `dev1/fgenBNC` and both cannot be used at the same time. PFI 15 is used as an external clock, as a trigger, and for counter input operations.

NI mioDAQ Physical Channels

In physical channel names, `Dev1` is the default device name for mioDAQ devices. You can change these names in MAX.

Analog Input

Depending on your mioDAQ device, you can have from 16 to 32 analog input channels. A 16-channel mioDAQ device has physical channels ranging from `Dev1/ai0` to `Dev1/ai15`, a 32-channel device from `Dev1/ai0` to `Dev1/ai31`.

Use only the physical channel name of the positive channel (not both) when creating a differential channel.

Analog Output

A mioDAQ device that supports two analog outputs has analog output physical channels named `Dev1/ao0` and `Dev1/ao1`.

A mioDAQ device that supports four analog outputs has analog output physical channels named `Dev1/ao0`, `Dev1/ao1`, `Dev1/ao2`, and `Dev1/ao3`.

Digital Input and Output

All mioDAQ devices have 16 lines of digital input and output named `Dev1/port0/line0` through `Dev1/port0/line15`. These lines belong to a single port, and the physical channel Dev1/port0 refers to all 16 lines at once. Port 0 can perform both hardware-timed and static digital operations.

The mioDAQ device digital lines can also be used as pins for Counter/Timer operations and Trigger routing. Such pins are sometimes referred to as Programmable Function

Interface (PFI) pins. On the mioDAQ devices, the same physical pins can be used as either digital I/O or PFI. The channel name mapping of PFI pins to digital I/O physical channels is shown in the following table.

PFI Signal	Physical Channel Name
PFI 0:15	`Dev1/port0/line0:15`

Counter Input and Output

All mioDAQ devices have four counter/timers referred to by the physical channel names `Dev1/ctr0`, `Dev1/ctr1`, `Dev1/ctr2`, and `Dev1/ctr3`. Unlike the other I/O types, these physical channel names do not refer to terminals on the I/O connector but instead to circuits within the device. There are five primary signals associated with each counter: SOURCE, GATE, AUX, OUT, and sample clock. NI-DAQmx has default terminals for these signals, except for the sample clock.

For counter input tasks, if you know whether your signal provides the SOURCE, AUX, or GATE function and wire your signal to the default, you do not have to set the Input Terminal attribute/property.

Counter	SOURCE Default	GATE Default	AUX Default	OUT Default
Dev1/ctr0	PFI 0	PFI 8	PFI 4	PFI 0
Dev1/ctr1	PFI 1	PFI 9	PFI 5	PFI 1
Dev1/ctr2	PFI 2	PFI 10	PFI 6	PFI 2
Dev1/ctr3	PFI 3	PFI 11	PFI 7	PFI 3

S Series Physical Channels

`Dev1` in physical channel names is the default device name for S Series devices. You can change these names in MAX.

Related reference:

- [AO Series, E Series, and S Series Signal Connections for Counters](#)

Analog Input

An S Series device has between two and eight analog input physical channels named `Dev1/ai0` to `Dev1/ai7`.

Analog Output

An S Series device that supports analog output has two analog output physical channels named `Dev1/ao0` and `Dev1/ao1`.

Digital Input and Output

All S Series devices have eight lines of digital input and output named `Dev1/port0/line0` through `Dev1/port0/line7`. These lines belong to a single port, and the physical channel `Dev1/port0` refers to all eight lines at once.

Counter Input and Output

All S Series devices have two counter/timers referred to by the physical channel names `Dev1/ctr0` and `Dev1/ctr1`. Unlike the other I/O types, these physical channel names do not refer to terminals on the I/O connector but instead to circuits within the device. There are three primary terminals associated with each counter. These are the terminals used as the SOURCE, GATE and OUT functions. NI-DAQmx has default values for these terminals. For counter input tasks, if you know whether your signal provides the SOURCE or GATE function and wire your signal to the default, you do not have to set the Input Terminal attribute/property.

Counter	SOURCE Default	GATE Default	OUT Default
Dev1/ctr0	PFI 8	PFI 9	CTR 0 OUT
Dev/ctr1	PFI 3	PFI 4	CTR 1 OUT

SCXI and SCC Physical Channels

`SC1Mod1` is the default device name for an SCXI module, where SC1 is the default chassis ID, and Mod1 refers to the slot number. These names can be changed in MAX.

Analog Input

An SCXI module usually has eight or 32 analog input channels; refer to your device documentation to be sure. These physical channel names are of the form `SC1Mod<slot#>/ai0` to `SC1Mod<slot#>/ai`N, where <slot#> is the chassis slot number of the module, and N equals the number of analog input channels on the module minus one. For example, `SCI1Mod1/ai31` is the highest numbered physical channel for a module with 32 analog input channels.

An SCC module has either one or two physical channels named `SCC1Mod`<J connector#>aiN, where <J connector#> is the number of the J connector where the SCC module resides, and N is the channel number. SCC1 is the SCC connector block ID (for example, `SCC1Mod1/ai0`).

NI PXI-4224 Only—You cannot scan channel ai7 and the CJC channel simultaneously in a task, since the CJC channel is multiplexed to channel 7. However, when you make a thermocouple measurement on ai0:7 with internal CJC, NI-DAQmx automatically reads the CJC channel at the beginning of the measurement and then scans the rest of the channels correctly.

Analog Output

An SCXI module has some number of output channels for voltage or current. These physical channel names are of the form `SC1Mod<slot#>/ao0` to `SC1Mod<slot#>/ao`N, where <slot#> is the chassis slot number of the module, and N equals the number of analog output channels on the module minus one. For example, `SC1Mod1/ao5` is the highest numbered physical channel on a module with six channels.

Digital Input and Output

An SCXI digital module has eight, 16, or 32 lines named `SC1Mod<slot#>/port0/line0` through `SC1Mod<slot#>/port0/line`N, where <slot#> is the chassis slot number of the module, and N is the number of digital lines minus one. For example, SC1Mod1/port0/line31 is the highest numbered line for a module with 32 lines. These lines belong to a single port and the physical channel named SC1Mod<slot#>/port0 refers to all the lines at once.

An SCC module has either one digital input line or one digital output line with names of the form `SCC1Mod<J connector#>di`N or `SCC1Mod<J connector#>do`N, where <J connector#> is the number of the J connector where the SCC module resides, and N is the channel number. SCC1 is the SCC connector block ID (for example, `SCC1Mod1/di0`).

SC Express Physical Channels

In physical channel names, `PXI1Slot1` is the default device name for SC Express devices, where PXI1 is the chassis number, and Slot1 refers to the slot number. You can change these names in MAX.

Analog Input

SC Express Devices have from four to 32 analog input channels. A 4-channel device has physical channels ranging from `PXISlot1/ai0` to `PXISlot1/ai3`, an 8-channel device has physical channels ranging from `PXISlot1/ai0` to `PXISlot1/ai7`, and a 32-channel device has channels ranging from `PXISlot1/ai0` to `PXISlot1/ai31`.

You can use channels from multiple analog input SC Express devices in the same NI-DAQmx task.

Analog Output

NI 4322 devices have eight analog output channels ranging from `PXI1Slot1/ao0` to `PXISlot1/ao7`. You can use channels from multiple analog output SC Express devices in the same NI-DAQmx task.

Strain and Wheatstone Bridge Measurements

The NI 4330 and 4331 devices support only the AI Strain Gage, AI Force Bridge, AI Pressure Bridge, AI Torque Bridge, AI Bridge (V/V), and AI Custom Voltage With Excitation channel types. The NI 4339 supports AI Voltage in addition to the previously mentioned measurement types.

When using NI 4330 and 4331 devices with an AI Custom Voltage With Excitation

channel, you must set the AI.Excit.UseForScaling attribute/property to true. This attribute/property causes the channel to return ratiometric data: Vin/Vex. These devices perform this division in hardware. The NI 4339 does not require the AI.Excit.UseForScaling attribute/property. Disabling this property allows for the NI 4339 to perform voltage measurements while providing voltage excitation.

NI 4330 and 4331 devices return a voltage ratio rather than a voltage. Therefore, use the AI.Bridge.InitialRatio attribute/property to specify the initial voltage ratio, or set the AI.Bridge.InitialVoltage attribute/property to the ratio Vin/Vex returned by the device, multiplied by Vex.

SensorDAQ Physical Channels

SensorDAQ has analog input, analog output, digital I/O, and counter channels. It also has three analog sensor channels (labeled Ch. 1, Ch. 2, and Ch. 3 on the device) and one digital sensor channel (labeled DIG on the device) for use with Vernier sensors. Please contact Vernier for additional information on the analog and digital sensor channels.

Analog Input

SensorDAQ has two AI physical channels, `Dev1/ai0` and `Dev1/ai1`. When you configure a channel in differential mode, `Dev1/ai0` is the positive input and `Dev1/ai1` is the negative input.

Analog Output

SensorDAQ has two AO physical channels, `Dev1/ao0` and `Dev1/ao1`.

Digital Input/Output

SensorDAQ has four digital I/O physical channels.

Counter Input and Output

There is one counter channel referred to by the physical channel name `Dev1/ctr0`. pfi0 is the terminal used for this physical channel.

TIO Physical Channels

Counter Input and Output

TIO devices have up to eight counter/timers referred to by the physical channel names `Dev1/ctr0` to `Dev1/ctr7`. Unlike the other I/O types, these physical channel names do not refer to terminals on the I/O connector but instead to circuits within the device.

There are four primary terminals associated with each TIO counter. These are the terminals used as the SOURCE, GATE, AUX, and OUT functions. NI-DAQmx has default values for these terminals. For counter input tasks, if you know whether your signals provide the SOURCE, GATE, or AUX functions and wire your signal to the default input, you do not have to set the Input Terminal attribute/property.

Counter	SOURCE Default	GATE Default	AUX Default	OUT Default
Dev1/ctr0	PFI 39	PFI 38	PFI 37	PFI 36
Dev1/ctr1	PFI 35	PFI 34	PFI 33	PFI 32
Dev1/ctr2	PFI 31	PFI 30	PFI 29	PFI 28
Dev1/ctr3	PFI 27	PFI 26	PFI 25	PFI 24
Dev1/ctr4	PFI 23	PFI 22	PFI 21	PFI 20
Dev1/ctr5	PFI 19	PFI 18	PFI 17	PFI 16
Dev1/ctr6	PFI 15	PFI 14	PFI 13	PFI 12
Dev1/ctr7	PFI 11	PFI 10	PFI 9	PFI 8

Note The NI 6601 has only four counters (ctr0–ctr3). The entries in the previous table for cntr4, cntr5, cntr6, and cntr7 do not apply for that device.

Digital Input and Output

NI 661x devices have two ports, port 0 and port 1. Port 0 has 32 lines of digital input and output named `Dev1/port0/line0` through `Dev1/port0/line31`. Port 1 has 8 lines of digital input and output named `Dev1/port1/line0` through `Dev1/`

`port1/line7`. The physical channel Dev1/port0 refers to all 32 lines at once, and the physical channel Dev1/port1 refers to all 8 lines at once. Port 0 can perform hardware-timed and static digital operations, while port 1 can perform only static digital operations.

Port 0 and port 1 can be used as static PFI lines, PFI 0..39. When any of PFI lines 0..39 is used as a digital I/O signal, it uses the physical channel name shown in the following table.

PFI Signal	Physical Channel Name
PFI 0	`Dev1/port0/line0`
PFI 1	`Dev1/port0/line1`
PFI 2	`Dev1/port0/line2`
PFI 3	`Dev1/port0/line3`
PFI 4	`Dev1/port0/line4`
PFI 5	`Dev1/port0/line5`
PFI 6	`Dev1/port0/line6`
PFI 7	`Dev1/port0/line7`
PFI 8	`Dev1/port0/line8`
PFI 9	`Dev1/port0/line9`
PFI 10	`Dev1/port0/line10`
PFI 11	`Dev1/port0/line11`
PFI 12	`Dev1/port0/line12`
PFI 13	`Dev1/port0/line13`
PFI 14	`Dev1/port0/line14`
PFI 15	`Dev1/port0/line15`
PFI 16	`Dev1/port0/line16`
PFI 17	`Dev1/port0/line17`
PFI 18	`Dev1/port0/line18`
PFI 19	`Dev1/port0/line19`

PFI Signal	Physical Channel Name
PFI 20	`Dev1/port0/line20`
PFI 21	`Dev1/port0/line21`
PFI 22	`Dev1/port0/line22`
PFI 23	`Dev1/port0/line23`
PFI 24	`Dev1/port0/line24`
PFI 25	`Dev1/port0/line25`
PFI 26	`Dev1/port0/line26`
PFI 27	`Dev1/port0/line27`
PFI 28	`Dev1/port0/line28`
PFI 29	`Dev1/port0/line29`
PFI 30	`Dev1/port0/line30`
PFI 31	`Dev1/port0/line31`
PFI 32	`Dev1/port1/line1`
PFI 33	`Dev1/port1/line2`
PFI 34	`Dev1/port1/line3`
PFI 35	`Dev1/port1/line4`
PFI 36	`Dev1/port1/line5`
PFI 37	`Dev1/port1/line6`
PFI 38	`Dev1/port1/line7`
PFI 39	`Dev1/port1/line8`

USB DAQ Physical Channels

`Dev1` in physical channel names is the default device name for USB Series DAQ devices. You can change these names in MAX.

Analog Input

The following table lists the number and naming of analog input physical channels for

USB DAQ devices.

Device	Number of Channels	Naming
NI USB-9211A, NI USB-9215A, NI USB-9219, NI USB-9229, NI USB-9234, NI USB-9237, NI USB-9239	4	`cDAQ1Mod1/ai0` to `cDAQ1Mod1/ai3`
NI USB-9201, NI USB-9221	8	`cDAQ1Mod1/ai0` to `cDAQ1Mod1/ai7`
NI USB-9213	16	`cDAQ1Mod1/ai0` to `cDAQ1Mod1/ai15`

Strain and Wheatstone Bridge Measurements

The NI 9235, NI 9236, and NI 9237 support only the AI Strain Gage, AI Force Bridge, AI Pressure Bridge, AI Torque Bridge, AI Bridge (V/V), and AI Custom Voltage With Excitation channel types.

When using the NI USB-9219 or I USB-9237 with an AI Custom Voltage With Excitation channel, you must set the AI.Excit.UseForScaling attribute/property to true. This attribute/property causes the channel to return ratiometric data: Vin/Vex. The NI USB-9219 and NI USB-9237 modules perform this division in hardware.

For the NI USB-9219, NI-DAQmx requires the AI.Excit.Val attribute/property to be set to 2.5 V for AI Strain Gage and AI Custom Voltage With Excitation channel types and to 500 µA for resistance and RTD measurements. The actual excitation voltage or current output by the NI USB-9219 varies with the sensor resistance or the load being measured.

NI USB-9219 and NI USB-9237 devices return a voltage ratio rather than a voltage. Therefore, use the AI.Bridge.InitialRatio attribute/property to specify the initial voltage ratio, or set the AI.Bridge.InitialVoltage attribute/property to the ratio Vin/Vex returned by the device, multiplied by Vex.

The NI USB-9219 does not have quarter bridge completion circuitry, which affects AI Strain Gage Quarter Bridge I channels and AI Custom Voltage With Excitation Quarter Bridge channels (but not AI Strain Gage Quarter Bridge II channels). With these channels, the NI USB-9219 performs a 2-wire resistance measurement on the active

gage element, then NI-DAQmx uses software scaling to convert the resistance measurement into a bridge ratio. For these channels, the polynomial coefficients specified by the AI.DevScalingCoeff attribute/property convert unscaled data into Ohms, not into V/V. Likewise, the AI.Rng.High/AI.Rng.Low attributes/properties should be specified in units of Ohms, not V/V.

When the NI USB-9219 is in quarter bridge mode, you need to use the AI.Bridge.NomResistance attribute/property to control whether the channel uses the 120 Ω range or the 350 Ω range.

Analog Output

The following table lists the number and naming of analog output physical channels for USB DAQ devices.

Device	Number of Channels	Naming
NI USB-9263, NI USB-9265	4	`cDAQ1Mod1/ao0` to `cDAQ1Mod1/ao3`
NI USB-9264	16	`cDAQ1Mod1/ao0` to `cDAQ1Mod1/ao15`

When using the NI USB-9263, NI-USB 9264, or NI USB-9265, you can run only one type of timing at a time. You can have one software-timed task per channel or one hardware-timed task running on a device at one time, but you cannot have a combination of timing on that device. For instance, you can run up to four software-timed tasks on an the NI USB-9263 concurrently, but running one hardware-timed task with one software-timed task generates an error. Additionally, the NI USB-9263, NI-USB 9264, and NI USB-9265 can run only one hardware-timed analog output task per device at a given time.

X Series Physical Channels

In physical channel names, `Dev1` is the default device name for X Series devices. You can change these names in MAX.

Related reference:

- [X Series Signal Connections for Counters](#)

Analog Input

Depending on your X Series device, you can have from eight to 208 analog input channels. An 8-channel X Series device has physical channels ranging from `Dev1/ai0` to `Dev1/ai7`, a 208-channel device from `Dev1/ai0` to `Dev1/ai207`.

Use only the physical channel name of the positive channel (not both) when creating a differential channel.

Analog Output

An X Series device that supports two analog outputs has analog output physical channels named `Dev1/ao0` and `Dev1/ao1`.

An X Series device that supports four analog outputs has analog output physical channels named `Dev1/ao0`, `Dev1/ao1`, `Dev1/ao2`, and `Dev1/ao3`.

Digital Input and Output

All X Series devices have eight or 32 lines of digital input and output named `Dev1/port0/line0` through `Dev1/port0/line7` or `Dev1/port0/line0` through `Dev1/port0/line31`. These lines belong to a single port, and the physical channel Dev1/port0 refers to all eight or 32 lines at once. Port 0 can perform both hardware-timed and static digital operations.

X Series devices have two more ports, port 1 and port 2. Port 1 has eight digital I/O lines, `Dev1/port1/line0` through `Dev1/port1/line7`. Port 2 has eight digital I/O lines, `Dev1/port2/line0` through `Dev1/port2/line7`. Port 1 and port 2 can be used as static digital I/O lines or PFI lines, PFI 0..15. When any of PFI lines 0..15 is used as a digital I/O signal, it uses the physical channel name shown in the following table.

PFI Signal	Physical Channel Name
PFI 0	`Dev1/port1/line0`
PFI 1	`Dev1/port1/line1`
PFI 2	`Dev1/port1/line2`

PFI Signal	Physical Channel Name
PFI 3	`Dev1/port1/line3`
PFI 4	`Dev1/port1/line4`
PFI 5	`Dev1/port1/line5`
PFI 6	`Dev1/port1/line6`
PFI 7	`Dev1/port1/line7`
PFI 8	`Dev1/port2/line0`
PFI 9	`Dev1/port2/line1`
PFI 10	`Dev1/port2/line2`
PFI 11	`Dev1/port2/line3`
PFI 12	`Dev1/port2/line4`
PFI 13	`Dev1/port2/line5`
PFI 14	`Dev1/port2/line6`
PFI 15	`Dev1/port2/line7`

Physical channel `Dev1/port1` refers to all eight lines, `Dev1/port1/line0:7`, at once. Physical channel `Dev1/port2` refers to all eight lines, `Dev1/port2/line0:7`, at once.

Counter Input and Output

All X Series devices have four counter/timers referred to by the physical channel names `Dev1/ctr0`, `Dev1/ctr1`, `Dev1/ctr2`, and `Dev1/ctr3`. Unlike the other I/O types, these physical channel names do not refer to terminals on the I/O connector but instead to circuits within the device. There are five primary signals associated with each counter: SOURCE, GATE, AUX, OUT, and sample clock. NI-DAQmx has default terminals for these signals, except for the sample clock. For counter input tasks, if you know whether your signal provides the SOURCE, AUX, or GATE function and wire your signal to the default, you do not have to set the Input Terminal attribute/property.

Counter	SOURCE Default	GATE Default	AUX Default	OUT Default
Dev1/ctr0	PFI 8	PFI 9	PFI 10	PFI 12

Counter	SOURCE Default	GATE Default	AUX Default	OUT Default
Dev1/ctr1	PFI 3	PFI 4	PFI 11	PFI 13
Dev1/ctr2	PFI 5	PFI 6	PFI 7	PFI 15
Dev1/ctr3	PFI 0	PFI 1	PFI 2	PFI 14

Internal Channels

On some devices, you can either acquire a signal present on the I/O connector or acquire a signal that is being generated from the internal multiplexer (an internal channel). The internal channels available on the multiplexer are typically used for calibration purposes, but you can also sample them as you would a physical signal present on the I/O connector. To read from one of these internal channels, you must use the internal channel as the device's physical channel (for example, `Dev1/_aignd_vs_aignd`) when creating the virtual channel.

Related concepts:

- Internal Channels for C Series Devices and TestScale Modules
- Internal Channels for DSA Devices
- Internal Channels for E Series Devices
- Internal Channels for FieldDAQ devices
- Internal Channels for M Series and NI 6010 Devices
- Internal Channels for NI ELVIS II Family
- Internal Channels for NI USB-TC01
- Internal Channels for the NI PXI-42xx
- Internal Channels for S Series Devices
- Internal Channels for SC Express Devices
- SCXI Internal Channels
- Internal Channels for USB DAQ Devices
- Internal Channels for X Series Multiplexed Sampling Devices
- Internal Channels for X Series Simultaneous Sampling Devices
- Internal Channels for C Series Devices

Internal Channels for C Series Devices and TestScale Modules

Internal Channels for the NI 9210 Device

Internal Channel Name	Description
_aignd_vs_aignd	A referenced single-ended terminal with the positive and negative terminals both connected to the ground reference for analog input.
_calref_vs_aignd	A differential terminal with the positive terminal connected to the internal calibration reference voltage and the negative terminal connected to the ground reference for analog input.
_cjtemp	A referenced single-ended terminal with the positive terminal connected to the onboard temperature sensor and the negative terminal connected to the ground reference for analog input. This channel is used for cold-junction compensation.

Internal Channels for the NI 9211 Device

Internal Channel Name	Description
_aignd_vs_aignd	A differential terminal with the positive and negative terminals both connected to the ground reference for analog input.
_calref_vs_aignd	A differential terminal with the positive terminal connected to the internal calibration reference voltage and the negative terminal connected to the ground reference for analog input.
_cjtemp	A differential terminal with the positive terminal connected to the onboard temperature sensor and the negative terminal connected to the ground reference for analog input. This channel is used for cold-junction compensation.

Internal Channels for the NI 9212 Device

Internal Channel Name	Description
_cjtemp0	A differential terminal with the positive terminal connected to the onboard temperature sensor and the negative terminal connected to the ground reference for analog input. This channel is used for cold-junction compensation.
_cjtemp1	This channel is used for cold-junction compensation.

Internal Channels for the NI 9213 Device

Internal Channel Name	Description
_aignd_vs_aignd	A differential terminal with the positive and negative terminals both connected to the ground reference for analog input.
_cjtemp	A differential terminal with the positive terminal connected to the onboard temperature sensor and the negative terminal connected to the ground reference for analog input. This channel is used for cold-junction compensation.

Internal Channels for the NI 9214 Device

Internal Channel Name	Description
_aignd_vs_aignd	A differential terminal with the positive and negative terminals both connected to the ground reference for analog input.
_cjtemp0	A differential terminal with the positive terminal connected to the onboard temperature sensor and the negative terminal connected to the ground reference for analog input. This channel is used for cold-junction compensation.
_cjtemp1	This channel is used for cold-junction compensation.
_cjtemp2	This channel is used for cold-junction compensation.

Internal Channels for the NI 9205, NI 9206, and TS-15100 Devices

Internal Channel Name	Description
_aignd_vs_aignd	A differential terminal with the positive and negative terminals both connected to the ground reference for analog input.
_calref_vs_aignd	A differential terminal with the positive terminal connected to the internal calibration reference voltage and the negative terminal connected to the ground reference for analog input.
_aignd_vs_aisense	A differential terminal with the positive terminal connected to the ground reference for analog input and the negative terminal connected to physical channel AI SENSE.
_calSrcHi_vs_aignd	A differential terminal with the positive terminal connected to the calibration PWM and the negative terminal connected to the ground

Internal Channel Name	Description
	reference for analog input.
_calref_vs_calSrcHi	A differential terminal with the positive terminal connected to the internal calibration reference voltage and the negative terminal connected to the calibration PWM.
_calSrcHi_vs_calSrcHi	A differential terminal with the positive and negative terminals connected to the calibration PWM.
_aignd_vs_calSrcHi	A differential terminal with the positive terminal connected to the ground reference for analog input and the negative terminal connected to the calibration PWM..
_calSrcMid_vs_aignd	A differential terminal with the positive terminal connected to the calibration PWM and the negative terminal connected to the ground reference for analog input. _calSrcMid is the divided down version of _calSrcHi.
_boardTempSensor_vs_aignd	A differential terminal with the positive terminal connected to the onboard temperature sensor and the negative terminal connected to the ground reference for analog input.
_ai0_vs_calSrcHi	A differential terminal with the positive terminal connected to physical channel ai0 and the negative terminal connected to the calibration PWM.
_ai8_vs_calSrcHi	A differential terminal with the positive terminal connected to physical channel ai8 and the negative terminal connected to the calibration PWM.

Internal Channels for the NI 9219 Device

Internal Channel Name	Description
_cjtemp0	A differential terminal with the positive terminal connected to the onboard temperature sensor and the negative terminal connected to the ground reference for analog input. This channel is used for cold-junction compensation for analog input channel 0.
_cjtemp1	This channel is used for cold-junction compensation for analog input channel 1.
_cjtemp2	This channel is used for cold-junction compensation for analog input channel 2.
_cjtemp3	This channel is used for cold-junction compensation for analog input channel 3.

Internal Channels for the CompactDAQ and CompactRIO Systems*

Internal Channel Name	Description
_ctr0	This physical channel name does not refer to a terminal on the I/O connector but instead to a circuit within the device. You must set the Input Terminal or Output Terminal attributes/properties that are appropriate for the measurement/generation being performed.
_ctr1	This physical channel name does not refer to a terminal on the I/O connector but instead to a circuit within the device. You must set the Input Terminal or Output Terminal attributes/properties that are appropriate for the measurement/generation being performed.
_ctr2	This physical channel name does not refer to a terminal on the I/O connector but instead to a circuit within the device. You must set the Input Terminal or Output Terminal attributes/properties that are appropriate for the measurement/generation being performed.
_ctr3	This physical channel name does not refer to a terminal on the I/O connector but instead to a circuit within the device. You must set the Input Terminal or Output Terminal attributes/properties that are appropriate for the measurement/generation being performed.
_freqout	This physical channel name does not refer to a terminal on the I/O connector but instead to a circuit within the device. You must set the Output Terminal attribute/property that is appropriate for the generation being performed.

* CompactDAQ chassis, TestScale chassis, CompactDAQ controller, CompactRIO controllers and CompactRIO Single-Board controllers:

- cDAQ-9132, 9133, 9134, 9135, 9136, 9137, 9171, 9174, 9178, 9179, 9181, 9184, 9185, 9188, 9188XT, 9189, and 9191
- cRIO-9040, 9041, 9042, 9043, 9045, 9046, 9047, 9048, 9049, 9053, 9054, 9055, 9056, 9057, and 9058
- TS-15000 and TS-15010
- sbRIO-9603, 9608, 9609, 9628, 9629, and 9638

Internal Channels for DSA Devices

On a DSA device, you can either acquire a signal present on the I/O connector, or you

can acquire a signal that is generated from the internal calibration multiplexer. The channels available on this multiplexer are typically used for calibration purposes, but you can also sample them as you would a physical signal present on the I/O connector. To read from one of these internal channels, you must use one of the device's AI physical channels (for instance, Dev1/ai0) when creating the channel to select which ADC to use, then set the appropriate string value on the Input Source channel attribute/property.

The following table lists internal channels for DSA devices.

Internal Channel Name	Supported Devices	Description
_external_channel	All DSA devices	The source of the AI channel is the device input connector, or an accessory connected to the device connector.
_5Vref_vs_aignd	4461/ 4462/ 4464, 447x	The source of the AI channel is the onboard reference signal (for example, +5V).
_pos10V_vs_aignd	4464	The source of the AI channel is the onboard reference signal (for example, +10V).
_neg10V_vs_aignd	4464	The source of the AI channel is the onboard reference signal (for example, -10V).
_pos1V_vs_aignd	4464	The source of the AI channel is the onboard reference signal (for example, +1V).
_pos316mV_vs_aignd	4464	The source of the AI channel is the onboard reference signal (for example, +316mV).
_aignd_vs_pos10V	4464	The source of the AI channel is the onboard reference signal (for example, +10V). The negative terminal is connected to the ground reference for analog input.
_aignd_vs_neg10V	4464	The source of the AI channel is the onboard reference signal (for example, -10V). The negative terminal is connected to the ground reference for analog input.
_ao0_vs_ao0neg [1] 37	4461 only	The source of the AI channel is the onboard analog output channel 0.
_ao1_vs_ao1neg [1]	4461 only	The source of the AI channel is the onboard analog output channel 1.

37. The AO internal channels are valid only for devices with AO physical channels.

Internal Channel Name	Supported Devices	Description
_aignd_vs_aignd	4461/ 4462/ 4464, 447x, 449x	The source of the AI channel is the onboard ground signal.
_ref_sqwv_vs_aignd	449x	The source of the AI channel is the onboard reference square wave signal.

For all DSA devices, only one internal channel can be read at a time, although the same internal channel can be read on multiple physical channels (with additional restrictions for NI 447x devices). For example, you cannot simultaneously read the internal 5 V reference on one physical channel and the analog ground on another physical channel.

The NI 447x AI physical channels are grouped into pairs, for instance {ai0, ai1}, {ai2, ai3}, and so on. NI 447x devices cannot read an internal channel on more than one physical channel group, and when reading an internal channel, both physical channels in the group are connected to the internal channel source. For example, if the Input Source for channel ai0 is set to 5Vref_vs_aignd and the Input Source for channel ai1 is left at the default value of _external_channel, ai1 still reads the internal channel 5Vref_vs_aignd since ai0 and ai1 are in the same physical channel group.

Internal Channels for E Series Devices

The following table is a list of internal physical channels for all E Series devices. Different E Series devices have different subsets of channels. These channels are typically for self-calibration, and you can sample them as you would a physical channel present on the I/O connector.

Internal Channel Name	Description
_aognd_vs_aognd	A differential terminal with the positive and negative terminals both connected to the ground reference for analog output.
_aognd_vs_aignd	A differential terminal with the positive terminal connected to the ground reference for analog output and the negative terminal connected to the ground reference for analog input.

Internal Channel Name	Description
_ao0_vs_aognd	A differential terminal with the positive terminal connected to physical channel ao0 and the negative terminal connected to the ground reference for analog output.
_ao1_vs_aognd	A differential terminal with the positive terminal connected to physical channel ao1 and the negative terminal connected to the ground reference for analog output.
_calref_vs_calref	A differential terminal with the positive terminal connected to the onboard 5 V reference and the negative terminal connected to the onboard 5 V reference.
_calref_vs_aignd	A differential terminal with the positive terminal connected to the onboard 5 V reference and the negative terminal connected to the ground reference for analog input.
_ao0_vs_calref	A differential terminal with the positive terminal connected to physical channel ao0 and the negative terminal connected to the onboard 5 V reference.
_ao1_vs_calref	A differential terminal with the positive terminal connected to physical channel ao1 and the negative terminal connected to the onboard 5 V reference.
_ao1_vs_ao0	A differential terminal with the positive terminal connected to physical channel ao1 and the negative terminal connected to physical channel ao0.
_boardTempSensor_vs_aignd	A differential terminal with the positive terminal connected to the onboard temperature sensor and the negative terminal connected to the ground reference for analog input.
_aignd_vs_aignd	A differential terminal with the positive terminal connected to the ground reference for analog input and the negative terminal connected the ground reference for analog input.
_caldac_vs_aignd	A differential terminal with the positive terminal connected to the onboard calibration DAC and the negative terminal connected to the ground reference for analog input.
_caldac_vs_calref	A differential terminal with the positive terminal connected to the onboard calibration DAC and the negative terminal connected to the onboard 5 V reference.
_PXI_SCXIbackplane_vs_aignd	A terminal where a signal being conditioned by a SCXI module is measured across the PXI/SCXI backplane and not the I/O

Internal Channel Name	Description
	connector. Reading from this channel is valid only on PXI devices inserted in the rightmost PXI slot of a PXI/SCXI combination chassis.

Internal Channels for FieldDAQ devices

Internal channels for the FD-11613

The following internal channels are available.

Internal Channel Name	Description
FD11613-xxxxxxx-Bank1/_cjtemp0	This channel is used for cold-junction compensation.
FD11613-xxxxxxx-Bank1/_cjtemp1	This channel is used for cold-junction compensation.

Internal channels for the FD-11614

The following internal channels are available.

Internal Channel Name	Description
FD11614-xxxxxxx-Bank1/_cjtemp0	This channel is used for cold-junction compensation.
FD11614-xxxxxxx-Bank1/_cjtemp1	This channel is used for cold-junction compensation.
FD11614-xxxxxxx-Bank2/_cjtemp0	This channel is used for cold-junction compensation.
FD11614-xxxxxxx-Bank2/_cjtemp1	This channel is used for cold-junction compensation.

Internal Channels for M Series and NI 6010 Devices

Internal Channel Name	Description
_aignd_vs_aignd	A differential terminal with the positive and negative terminals both connected to the ground reference for analog input.
_ao0_vs_aognd	A differential terminal with the positive terminal connected to physical channel ao0 and the negative terminal connected to the

Internal Channel Name	Description
	ground reference for analog output.
_ao1_vs_aognd	A differential terminal with the positive terminal connected to physical channel ao1 and the negative terminal connected to the ground reference for analog output.
_ao2_vs_aognd	(For M Series devices only.) A differential terminal with the positive terminal connected to physical channel ao2 and the negative terminal connected to the ground reference for analog output.
_ao3_vs_aognd	(For M Series devices only.) A differential terminal with the positive terminal connected to physical channel ao3 and the negative terminal connected to the ground reference for analog output.
_calref_vs_aignd	A differential terminal with the positive terminal connected to the internal calibration reference voltage and the negative terminal connected to the ground reference for analog input.
_aignd_vs_aisense	A differential terminal with the positive terminal connected to the ground reference for analog input and the negative terminal connected to physical channel AI SENSE.
_aignd_vs_aisense2	A differential terminal with the positive terminal connected to the ground reference for analog input and the negative terminal connected to physical channel AI SENSE2.
_calSrcHi_vs_aignd	A differential terminal with the positive terminal connected to the calibration PWM and the negative terminal connected to the ground reference for analog input.
_calref_vs_calSrcHi	A differential terminal with the positive terminal connected to the internal calibration reference voltage and the negative terminal connected to the calibration PWM.
_calSrcHi_vs_calSrcHi	A differential terminal with the positive and negative terminals connected to the calibration PWM.
_aignd_vs_calSrcHi	A differential terminal with the positive terminal connected to the ground reference for analog input and the negative terminal connected to the calibration PWM.
_calSrcMid_vs_aignd	A differential terminal with the positive terminal connected to the calibration PWM and the negative terminal connected to the ground reference for analog input. _calSrcMid is the divided down version of _calSrcHi.
_calSrcLo_vs_aignd	A differential terminal with the positive terminal connected to the

Internal Channel Name	Description
	calibration PWM and the negative terminal connected to the ground reference for analog input. _calSrcLo is the divided down version of _calSrcHi.
_ai0_vs_calSrcHi	A differential terminal with the positive terminal connected to physical channel ai0 and the negative terminal connected to the calibration PWM.
_ai8_vs_calSrcHi	A differential terminal with the positive terminal connected to physical channel ai8 and the negative terminal connected to the calibration PWM.
_boardTempSensor_vs_aignd	A differential terminal with the positive terminal connected to the onboard temperature sensor and the negative terminal connected to the ground reference for analog input..
_PXI_SCXIbackplane_vs_aignd	A terminal where a signal being conditioned by a SCXI module is measured across the PXI/SCXI backplane and not the I/O connector. Reading from this channel is valid only on PXI devices inserted in the rightmost PXI slot of a PXI/SCXI combination chassis.

Internal Channels for NI ELVIS II Family

Internal Channel Name	Description
_aignd_vs_aignd	A differential terminal with the positive and negative terminals both connected to the ground reference for analog input.
_ao0_vs_aognd	A differential terminal with the positive terminal connected to physical channel ao0 and the negative terminal connected to the ground reference for analog output.
_ao1_vs_aognd	A differential terminal with the positive terminal connected to physical channel ao1 and the negative terminal connected to the ground reference for analog output.
_calref_vs_aignd	A differential terminal with the positive terminal connected to the internal calibration reference voltage and the negative terminal connected to the ground reference for analog input.
_aignd_vs_aisense	A differential terminal with the positive terminal connected to the ground reference for analog input and the negative terminal connected to physical channel AI SENSE.

Internal Channel Name	Description
_aignd_vs_aisense2	A differential terminal with the positive terminal connected to the ground reference for analog input and the negative terminal connected to physical channel AI SENSE2.
_calSrcHi_vs_aignd	A differential terminal with the positive terminal connected to the calibration PWM and the negative terminal connected to the ground reference for analog input.
_calref_vs_calSrcHi	A differential terminal with the positive terminal connected to the internal calibration reference voltage and the negative terminal connected to the calibration PWM.
_calSrcHi_vs_calSrcHi	A differential terminal with the positive and negative terminals connected to the calibration PWM.
_aignd_vs_calSrcHi	A differential terminal with the positive terminal connected to the ground reference for analog input and the negative terminal connected to the calibration PWM.
_calSrcMid_vs_aignd	A differential terminal with the positive terminal connected to the calibration PWM and the negative terminal connected to the ground reference for analog input. _calSrcMid is the divided down version of _calSrcHi.
_calSrcLo_vs_aignd	A differential terminal with the positive terminal connected to the calibration PWM and the negative terminal connected to the ground reference for analog input. _calSrcLo is the divided down version of _calSrcHi.
_ai0_vs_calSrcHi	A differential terminal with the positive terminal connected to physical channel ai0 and the negative terminal connected to the calibration PWM.
_ai8_vs_calSrcHi	A differential terminal with the positive terminal connected to physical channel ai8 and the negative terminal connected to the calibration PWM.
_boardTempSensor_vs_aignd	A differential terminal with the positive terminal connected to the onboard temperature sensor and the negative terminal connected to the ground reference for analog input..
_ai16:31	A differential or RSE terminal used to access internally connected AI terminals.
_vpsPosCurrent	A differential terminal connected to the shunt resistor on the variable power supply.

Internal Channel Name	Description
_vpsNegCurrent	A differential terminal connected to the shunt resistor on the variable power supply.
_vpsPos_vs_gnd	A differential terminal connected to a voltage divider on the variable power supply.
_vpsNeg_vs_gnd	A differential terminal connected to a voltage divider on the variable power supply.
_dutNeg	Reads the voltage equivalent of Current into the DUT- pin.
_base	Reads the voltage at the base pin (base voltage for 3-wire analyzer).
_dutPos	Reads the voltage at the DUT+ pin.
_fgenImpedance	Internally routes the function generator to the impedance analyzer circuit.
_ao0Impedance	Internally routes AO 0 to the impedance analyzer for Three-Wire Current-Voltage Analyzer measurements.

Internal Channels for the NI PXI-42*xx*

The following table is a list of internal physical channels for the PXI-42xx devices. The subset of channels present on your device depends on the specific E Series device being used. These channels are typically for self-calibration, and you can sample them as you would a physical channel present on the I/O connector

Internal Channel Name	Description
_cjTemp	A cold-junction compensation channel for measuring the temperature at the I/O connector when making thermocouple measurements.
_aignd_vs_aignd	A differential terminal with the positive terminal connected to the ground reference for analog input and the negative terminal connected the ground reference for analog input.
_caldac_vs_aignd	A differential terminal with the positive terminal connected to the onboard calibration DAC and the negative terminal connected to the ground reference for analog input.
_calref_vs_aignd	A differential terminal with the positive terminal connected to the onboard 5 V reference and the negative terminal connected to the

Internal Channel Name	Description
	ground reference for analog input.
_boardTempSensor_vs_aignd	A differential terminal with the positive terminal connected to the onboard temperature sensor and the negative terminal connected to the ground reference for analog input.
PXI_SCXIbackplane_vs_aignd	A terminal where a signal being conditioned by a SCXI module is measured across the PXI/SCXI backplane and not the I/O connector. Reading from this channel is valid only on PXI devices inserted in the rightmost PXI slot of a PXI/SCXI combination chassis.
_extcal_vs_aignd	A differential terminal with the positive terminal connected to the external calibration input and the negative terminal connected to the ground reference for analog input.
_pod_calrefPos_vs_aignd	A differential terminal with the positive terminal connected to the internal positive reference voltage and the negative terminal connected to the ground reference for analog input.
_pod_calrefNeg_vs_aignd	A differential terminal with the positive terminal connected to the internal negative reference voltage and the negative terminal connected to the ground reference for analog input.

Internal Channels for NI USB-TC01

Internal Channel Name	Description
_cjtemp	A differential terminal with the positive terminal connected to the onboard temperature sensor and the negative terminal connected to the ground reference for analog input. This channel is used for cold-junction compensation.

Internal Channels for S Series Devices

On an S Series device, you can either acquire a signal present on the I/O connector, or you can acquire a signal that is being generated from the internal calibration multiplexer. The channels available on this multiplexer are typically used for calibration purposes, but you can also sample them as you would a physical signal present on the I/O connector. To read from one of these internal channels, you must use one of the device's AI physical channels (Dev1/ai0 through Dev1/ai7) when

creating the virtual channel. The physical channel specifies the ADC for the internal channel. You can then set the appropriate string value on the Input Source channel attribute/property.

> ✎ **Note** All S Series devices must have the same Input Source setting on all channels. The NI PCI-6110 and NI PCI-6111 devices cannot acquire from more than one ADC at a time when using an internal channel.

- NI PCI-6110, NI PCI-6111, NI 6115, NI 6120 Internal Channels
  - _external_channel
  - _aognd_vs_aognd
  - _aognd_vs_aignd
  - _ao0_vs_aognd
  - _ao1_vs_aognd
  - _calref_vs_calref
  - _calref_vs_aignd
  - _ao0_vs_calref
  - _ao1_vs_calref
- NI PCI-6143 Internal Channels
  - _external_channel
  - _aignd_vs_aignd
  - _calref_vs_aignd
  - _calSrcHi_vs_aignd
  - _calref_vs_calSrcHi
  - _calSrcHi_vs_calSrcHi
  - _aignd_vs_calSrcHi
- NI PXI-6132/6133 Internal Channels
  - _external_channel
  - _aignd_vs_aignd
  - _calref_vs_aignd
  - _calSrcMid_vs_aignd
  - _calSrcHi_vs_aignd
  - _calref_vs_calSrcHi
  - _calSrcMid_vs_calSrcHi
  - _calSrcHi_vs_calSrcHi
  - _aignd_vs_calSrcHi

- NI PCI-6154 Internal Channels
  - _external_channel
  - _aignd_vs_aignd
  - _calref_vs_aignd
  - _calSrcHi_vs_aignd
  - _aignd_vs_calSrcHi
  - _calref_vs_calSrcHi
  - _calSrcHi_vs_calSrcHi
  - _aox_vs_aognd

The following table describes all S Series internal channels.

Internal Channel Name	Description
_external_channel	The differential terminal on the I/O connector that is typically used for acquiring data.
_aignd_vs_aignd	A differential terminal with the positive and negative terminals both connected to the ground reference for analog input.
_aognd_vs_aognd	A differential terminal with the positive and negative terminals both connected to the ground reference for analog output.
_aognd_vs_aignd	A differential terminal with the positive terminal connected to the ground reference for analog output and the negative terminal connected to the ground reference for analog input.
_aox_vs_aognd	A differential terminal with the positive terminal connected to the analog output physical channel, such as ao0, and the negative terminal connected to the ground reference for analog output.
_ao1_vs_aognd	A differential terminal with the positive terminal connected to physical channel ao1 and the negative terminal connected to the ground reference for analog output.
_calref_vs_calref	A differential terminal with the positive terminal connected to the internal calibration reference voltage and the negative terminal connected to the internal calibration reference voltage.
_calref_vs_aignd	A differential terminal with the positive terminal connected to the internal calibration reference voltage and the negative terminal connected to the ground reference for analog input.
_ao0_vs_calref	A differential terminal with the positive terminal connected to physical

Internal Channel Name	Description
	channel ao0 and the negative terminal connected to the internal calibration reference voltage.
_ao1_vs_calref	A differential terminal with the positive terminal connected to physical channel ao1 and the negative terminal connected to the internal calibration reference voltage.
_calSrcHi_vs_aignd	A differential terminal with the positive terminal connected to the calibration PWM and the negative terminal connected to the ground reference for analog input.
_calref_vs_calSrcHi	A differential terminal with the positive terminal connected to the internal calibration reference voltage and the negative terminal connected to the calibration PWM.
_calSrcHi_vs_calSrcHi	A differential terminal with the positive and negative terminals connected to the calibration PWM.
_aignd_vs_calSrcHi	A differential terminal with the positive terminal connected to the ground reference for analog input and the negative terminal connected to the calibration PWM.
_calSrcMid_vs_aignd	A differential terminal with the positive terminal connected to the calibration PWM and the negative terminal connected to the ground reference for analog input. _calSrcMid is the divided down version of _calSrcHi.
_calSrcMid_vs_calSrcHi	A differential terminal with the positive and negative terminals connected to the calibration PWM. _calSrcMid is the divided down version of _calSrcHi.

Internal Channels for SC Express Devices

On an NI 4300 or NI 433x device, you can either acquire a signal present on the I/O connector, or you can acquire an onboard signal, via an internal multiplexer. The channels available on this multiplexer are typically used for calibration purposes, but you can also sample them as you would a physical signal present on the I/O connector. To read from one of these internal channels, you must use one of the device's AI physical channels (Dev1/ai0 through Dev1/ai7) when creating the virtual channel. The physical channel specifies the ADC for the internal channel. You can then set the appropriate string value on the Input Source channel attribute/property.

On the NI 4353, you can acquire internal channels by specifying their names for the physical channels input of the DAQmx Create Channel VI/function.

Internal Channels for the NI 4302 and 4303

Input Source	Description
_cjtemp	The temperature read by a digital temperature sensor located on the TB-4302 accessory. This channel is used for cold-junction compensation.

Input Sources for the NI 4300 and 4310

Input Source	Description
_external_channel	The source of the AI channel is the device input connector, or an accessory connected to the device connector.
_aignd_vs_aignd	The source of the AI channel is the onboard ground signal.
_calref_vs_aignd	The source of the AI channel is the onboard reference signal.
_calSrcHi_vs_aignd	The source of the AI channel has the positive terminal connected to the onboard calibration PWM and the negative terminal connected to the onboard ground reference.
_aignd_vs_calSrcHi	The source of the AI channel has the positive terminal connected to the onboard ground reference and the negative terminal connected to the onboard calibration PWM.
_calref_vs_calSrcHi	The source of the AI channel has the positive terminal connected to the onboard reference signal and the negative terminal connected to the onboard calibration PWM.
_aipos_vs_calSrcHi	The source of the AI channel has the positive terminal connected to the positive terminal of the device input connector and the negative terminal connected to the onboard calibration PWM.
_aineg_vs_calSrcHi	The source of the AI channel has the positive terminal connected to the negative terminal of the device input connector and the negative terminal connected to the onboard calibration PWM.

Internal Channels for the NI 4309

On the NI 4309, you can acquire internal channels by specifying their names for the

physical channels input of the DAQmx Create Channel VI/function.

Input Source	Description
_aignd_vs_aignd<0..7>	A differential terminal with the positive and negative terminals both connected to the ground reference; using the specified ADC <0..7>. This channel is used to measure the autozero offset of individual ADCs.
_calSrcLo_vs_aignd<0..7>	A differential terminal with the positive terminal connected to the calibration PWM and the negative terminal connected to the ground reference; using the specified ADC <0..7>. _calSrcLo is the divided down version of _calSrcHi.
_7VRef_vs_aignd<0..7>	A differential terminal with the positive terminal connected to the onboard 7V reference signal and the negative terminal connected to the ground reference; using the specified ADC <0..7>.
_calSrcHi_vs_aignd<0..7>	A differential terminal with the positive terminal connected to the onboard calibration PWM and the negative terminal connected to the ground reference; using the specified ADC <0..7>.

Input Sources for the NI 4330 and 4331

Input Source	Description
_external_channel	The source of the AI channel is the device input connector, or an accessory connected to the device connector.
_aignd_vs_aignd	The source of the AI channel is the onboard ground signal.

Input Sources for the NI 4339

Input Source	Description
_external_channel	The source of the AI channel is the device input connector, or an accessory connected to the device connector.
_paired_channel_excitation	The source of the AI channel is the paired channel's internal excitation. The channel pairs are ai0 and ai1, ai2 and ai3, ai4 and ai5, ai6 and ai7.
_aignd_vs_aignd	The source of the AI channel is the onboard signal ground.

Internal Channels for the NI 4353

On the NI 4353, you can acquire internal channels by specifying their names for the physical channels input of the DAQmx Create Channel VI/function.

Input Source	Description
_cjtemp0 through _cjtemp7	A differential terminal with the positive and negative terminals connected to a temperature sensor located on the accessory. These channels are used for cold-junction compensation. The mapping between analog input channels and cold-junction compensation channels depends on the type of installed accessory.
_aignd_vs_aignd0	A differential terminal with the positive and negative terminals both connected to the ground reference, using the ADC that is used for even-numbered channels (ai0, ai2, ai4, and so on). This channel is used to measure the autozero offset of even-numbered channels.
_aignd_vs_aignd1	A differential terminal with the positive and negative terminals both connected to the ground reference, using the ADC that is used for odd-numbered channels (ai1, ai3, ai5, and so on). This channel is used to measure the autozero offset of odd-numbered channels.

SCXI Internal Channels

Some SCXI modules also have internal channels. These are physical channels that are not accessible from an I/O connector. To measure the signals present at these internal physical channels, use them to create virtual channels. The SCXI-1100, SCXI-1102, SCXI-1120, SCXI-1121, SCXI-1122, and SCXI-1125 modules have an internal physical channel called _cjTemp channel. It is the cold-junction compensation channel for measuring the temperature at the connector for thermocouples.

The SCXI-1112 has internal channels _cjTemp0, _cjTemp1, _cjTemp2, _cjTemp3, _cjTemp4, _cjTemp5, _cjTemp6, and _cjTemp7. These are the cold-junction compensation channels for each analog input channel on the SCXI-1112.

The SCXI-1520 has eight pairs of internal channels _pPos0 and _pNeg0 through _pPos7 and _pNeg7. These channels read back the excitation on the corresponding analog input channel. The pPos half of the pair is the positive side of the excitation, and the pNeg half is the negative side of the excitation. The real excitation value is the pPos value minus the pNeg value.

The SCXI-1521/B has 24 voltage excitation internal channels _Vex0 through _Vex23. These channels read back the excitation on the corresponding analog input channel. In addition, the SCXI-1521/B has 24 pairs of internal channels _IexPos0 and _IexNeg0 through _IexPos23 and _IexNeg23. The current through a sensor connected to channel x is the _IexPosx value minus the _IexNegx value.

Internal Channels for USB DAQ Devices

The following table lists the internal channel for the USB-9211 device.

Internal Channel Name	Description
_cjtemp	A differential terminal with the positive terminal connected to the onboard temperature sensor and the negative terminal connected to the ground reference for analog input. This channel is used for cold-junction compensation.

The following table lists the internal channels for the USB-9219 device.

Internal Channel Name	Description
_cjtemp0	A differential terminal with the positive terminal connected to the onboard temperature sensor and the negative terminal connected to the ground reference for analog input. This channel is used for cold-junction compensation for analog input channel 0.
_cjtemp1	This channel is used for cold-junction compensation for analog input channel 1.
_cjtemp2	This channel is used for cold-junction compensation for analog input channel 2.
_cjtemp3	This channel is used for cold-junction compensation for analog input channel 3.

Internal Channels for X Series Multiplexed Sampling Devices

On an X Series device, you can either acquire a signal present on the I/O connector, or you can acquire a signal that is being generated from the internal calibration multiplexer. The channels available on this multiplexer are typically used for calibration purposes, but you can also sample them as you would a physical signal

present on the I/O connector. To read from one of these internal channels on a X Series multiplexed sampling device, you must use the internal channel as the device's physical channel (for example, Dev1/_aignd_vs_aignd) when creating the virtual channel.

The following table describes all internal channels for X Series multiplexed sampling devices.

Internal Channel Name	Description
_aignd_vs_aignd	A single-ended terminal with the positive and negative terminals both connected to the ground reference for analog input.
_ao0_vs_aognd	A differential terminal with the positive terminal connected to physical channel ao0 and the negative terminal connected to the ground reference for analog output.
_ao1_vs_aognd	A differential terminal with the positive terminal connected to physical channel ao1 and the negative terminal connected to the ground reference for analog output.
_ao2_vs_aognd	A differential terminal with the positive terminal connected to physical channel ao2 and the negative terminal connected to the ground reference for analog output.
_ao3_vs_aognd	A differential terminal with the positive terminal connected to physical channel ao3 and the negative terminal connected to the ground reference for analog output.
_calref_vs_aignd	A single-ended terminal with the positive terminal connected to the internal calibration reference voltage and the negative terminal connected to the ground reference for analog input.
_aignd_vs_aisense	A differential terminal with the positive terminal connected to the ground reference for analog input and the negative terminal connected to physical channel AI SENSE.
_aignd_vs_aisense2	A differential terminal with the positive terminal connected to the ground reference for analog input and the negative terminal connected to physical channel AI SENSE2.
_aignd_vs_aisense3	A differential terminal with the positive terminal connected to the ground reference for analog input and the negative terminal connected to physical channel AI SENSE3.
_aignd_vs_aisense4	A differential terminal with the positive terminal connected to the

Internal Channel Name	Description
	ground reference for analog input and the negative terminal connected to physical channel AI SENSE4.
_calSrcHi_vs_aignd	A single-ended terminal with the positive terminal connected to the calibration PWM and the negative terminal connected to the ground reference for analog input.
_calref_vs_calSrcHi	A differential terminal with the positive terminal connected to the internal calibration reference voltage and the negative terminal connected to the calibration PWM.
_calSrcHi_vs_calSrcHi	A differential terminal with the positive and negative terminals connected to the calibration PWM.
_aignd_vs_calSrcHi	A differential terminal with the positive terminal connected to the ground reference for analog input and the negative terminal connected to the calibration PWM.
_calSrcMid_vs_aignd	A single-ended terminal with the positive terminal connected to the calibration PWM and the negative terminal connected to the ground reference for analog input. _calSrcMid is the divided down version of _calSrcHi.
_calSrcLo_vs_aignd	A single-ended terminal with the positive terminal connected to the calibration PWM and the negative terminal connected to the ground reference for analog input. _calSrcLo is the divided down version of _calSrcHi.
_ai0_vs_calSrcHi	A differential terminal with the positive terminal connected to physical channel ai0 and the negative terminal connected to the calibration PWM.
_ai8_vs_calSrcHi	A differential terminal with the positive terminal connected to physical channel ai8 and the negative terminal connected to the calibration PWM.
_boardTempSensor_vs_aignd	A single-ended terminal with the positive terminal connected to the onboard temperature sensor and the negative terminal connected to the ground reference for analog input.

Related concepts:

- X Series Device Groupings

Internal Channels for X Series Simultaneous Sampling Devices

On an X Series device, you can either acquire a signal present on the I/O connector, or you can acquire a signal that is being generated from the internal calibration multiplexer. The channels available on this multiplexer are typically used for calibration purposes, but you can also sample them as you would a physical signal present on the I/O connector. To read from one of these internal channels on a X Series simultaneous sampling device, you must use one of the device's AI physical channels (for example, Dev1/ai0) when creating the virtual channel. The physical channel specifies the ADC for the internal channel. You can then set the appropriate string value on the Input Source channel attribute/property.

> ✎ **Note** All X Series simultaneous sampling devices must have the same Input Source setting on all channels.

Internal Channel Name	Description
_external_channel	The differential terminal on the I/O connector that is typically used for acquiring data.
_ai0_vs_calSrcHi	A differential terminal with the positive terminal connected to physical channel ao0 and the negative terminal connected to the calibration PWM.
_ai1_vs_calSrcHi	A differential terminal with the positive terminal connected to physical channel ao1 and the negative terminal connected to the calibration PWM
_ai0_vs_aignd	A differential terminal with the positive terminal connected to physical channel ai0 and the negative terminal connected to the ground reference for analog input
_ai8_vs_aignd	A differential terminal with the positive terminal connected to the negative terminal of ai0 and the negative terminal connected to the ground reference for analog input.
_ai8_vs_calSrcHi	A differential terminal with the positive terminal connected to the negative terminal of ai0 and the negative terminal connected to the calibration PWM.
_aignd_vs_aignd	A single-ended terminal with the positive and negative terminals both connected to the ground reference for analog input.
_aignd_vs_calSrcHi	A differential terminal with the positive terminal connected to the ground

Internal Channel Name	Description
	reference for analog input and the negative terminal connected to the calibration PWM.
_ao0_vs_aognd	A differential terminal with the positive terminal connected to physical channel ao0 and the negative terminal connected to the ground reference for analog output.
_ao0_vs_calSrcHi	A differential terminal with the positive terminal connected to physical channel ao0 and the negative terminal connected to the calibration PWM.
_ao1_vs_aognd	A differential terminal with the positive terminal connected to physical channel ao1 and the negative terminal connected to the ground reference for analog output.
_ao1_vs_calSrcHi	A differential terminal with the positive terminal connected to physical channel ao1 and the negative terminal connected to the calibration PWM.
_ao2_vs_aognd	A differential terminal with the positive terminal connected to physical channel ao2 and the negative terminal connected to the ground reference for analog output.
_ao3_vs_aognd	A differential terminal with the positive terminal connected to physical channel ao3 and the negative terminal connected to the ground reference for analog output.
_calref_vs_aignd	A single-ended terminal with the positive terminal connected to the internal calibration reference voltage and the negative terminal connected to the ground reference for analog input.
_calSrcHi_vs_aignd	A single-ended terminal with the positive terminal connected to the calibration PWM and the negative terminal connected to the ground reference for analog input.
_calSrcHi_vs_calSrcHi	A differential terminal with the positive and negative terminals connected to the calibration PWM.
_calref_vs_calSrcHi	A differential terminal with the positive terminal connected to the internal calibration reference voltage and the negative terminal connected to the calibration PWM.
_calSrcMid_vs_aignd	A single-ended terminal with the positive terminal connected to the calibration PWM and the negative terminal connected to the ground reference for analog input. _calSrcMid is the divided down version of _calSrcHi.
_calSrcMid_vs_calSrcHi	A differential terminal with the positive and negative terminals connected

Internal Channel Name	Description
	to the calibration PWM. _calSrcMid is the divided down version of _calSrcHi .
_calSrcHi_vs_ai0	A differential terminal with the positive terminal connected to the calibration PWM and the negative terminal connected to physical channel ai0.
_calSrcHi_vs_ai8	A differential terminal with the positive terminal connected to the calibration PWM and the negative terminal connected to physical channel ai8.

Related concepts:

- X Series Device Groupings

Default Input/Output Terminal Configurations

If you do not explicitly specify the input or output terminal configuration when you create a channel, NI-DAQmx automatically determines the default terminal configuration at run time. The following table lists the default terminal configurations for devices.

Device	Default Input Terminal Configuration	Default Output Terminal Configuration
AO Series	N/A	Referenced single-ended
DSA	Pseudodifferential	Differential
NI 60xx (E Series), NI 62xx (M Series), NI 63xx (X Series), NI ELVIS II, NI PCI-6010, NI USB-6001, NI USB-6002, NI USB-6003, NI USB-6008, NI USB-6009, and NI TC01	For devices with eight channels: differential for the first four channels, referenced single-ended for the next four channels. For devices with 16 channels or more: differential for eight channels followed by referenced single-ended for eight channels. For instance,	Referenced single-ended

Device	Default Input Terminal Configuration	Default Output Terminal Configuration
	channels 0-7, 16-23, and 32-39 are differential. Channels 8-15, 24-31, and 40-47 are referenced-single ended.	
myDAQ	Differential for the analog input channels, referenced single-ended for the audio input channels.	Referenced single-ended
NI ELVIS II+	Referenced single-ended for scopeCh0 and scopeCh1. For other analog input channels: differential for channels 0-7, referenced single-ended for channels 8-15.	Referenced single-ended
NI USB-6000	Referenced single-ended	N/A
NI PXI-6132, NI PXI-6133, NI PXI-6143	Differential	N/A
NI PCI-6110, NI PCI-6111, NI 6115, NI 6120	Pseudodifferential	Referenced single-ended
NI PXI-42xx	Differential	N/A
SCC	Non-referenced single-ended	Referenced single-ended
SC Express	Differential. On the NI 4353, the cold-junction compensation channels are referenced single ended by default.	N/A
SCXI	Differential	Differential
NI 9201, NI USB-9201, NI 9203, NI 9208, NI 9217, NI 9221, NI USB-9221, NI 9253, NI 9775[1]38	Referenced single-ended	N/A
NI 9207	Differential for voltage channels, referenced single-ended for current channels	N/A
NI 9205, NI 9206, NI 9209, and AI 0-15 on CompactRIO single-board controllers	For devices with 16 channels or more: differential for eight channels followed by referenced single-ended for eight	N/A

38. All listed devices have a fixed terminal configuration
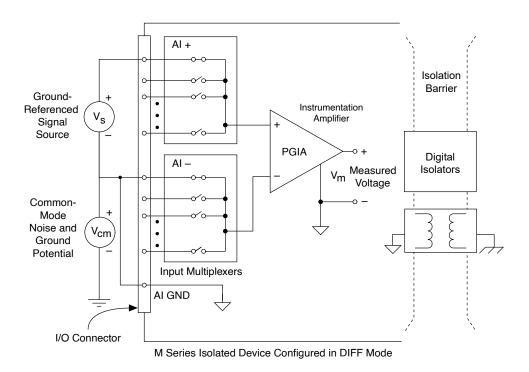
Device	Default Input Terminal Configuration	Default Output Terminal Configuration
	channels. For instance, channels 0-7, 16-23, and 32-39 are differential. Channels 8-15, 24-31, and 40-47 are referenced-single ended.	
FD-11601, FD-11603, FD-11605, FD-11613, FD-11614, FD-11637, NI 9202, NI 9210, NI 9211, NI USB-9211A, NI 9212, NI 9213, NI USB-9213, NI 9215, NI USB-9215A, NI 9216, NI 9218, NI 9219, NI USB-9219, NI 9220, NI 9224, NI 9225, NI 9226, NI 9227, NI 9229, NI USB-9229, NI 9235, NI 9236, NI 9237, NI 9228, NI USB-9237, NI 9238, NI 9239, NI USB-9239, NI 9246, NI 9247, NI 9250, NI 9251, NI 9252 [1]	Differential	N/A
FD-11634, NI 9230, NI 9231, NI 9232, NI 9234, NI USB-9234	Pseudodifferential	N/A
NI 9242, NI 9244	Nonreferenced single-ended for ai0, ai1, and ai2. Referenced single-ended for neutral.	
NI 9262, NI 9263, NI USB-9263, NI 9264, NI USB-9264, NI 9265, NI USB-9265, NI 9266, and AO 0-3 on CompactRIO single-board controllers	N/A	Referenced single-ended
NI 9269	N/A	Differential

Terminal Configurations (Analog Input Ground Reference Settings) for Isolated Devices

You can use differential, referenced single-ended (RSE), or nonreferenced single-ended (NRSE) terminal configurations (or analog input ground reference settings) for isolated devices.

The following figure shows a differential measurement system. For illustrations of

other terminal configurations, refer to your device documentation.

M Series Isolated Device Configured in DIFF Mode

RSE and NRSE measurement systems are the same for isolated devices in that the measurement is made with respect to a floating or isolated ground, AI GND. AI GND is the floating reference for all RSE and NRSE channels. AI GND is isolated from earth ground through an isolation barrier on the device.

It is important to keep within the specifications of your device to avoid hazardous conditions. It is considered improper use of the device to surpass the specifications, and the device is no longer considered to be in safe use. Isolated devices specify a continuous working isolation voltage that specifies the maximum voltage difference allowed between any of the input signals to the chassis/earth ground. For example, a product rated for 60 VDC of continuous working isolation that has a voltage difference of +51 VDC between AIGND and the chassis/earth ground cannot have a signal greater than +9 VDC when referenced to a AIGND or 60 VDC when reference to chassis/earth ground at its input terminals.

Routing

This section contains information about routing for AO Series, E Series, S Series, and

TIO devices.

Routing Considerations for AO Series Devices

The counters on these devices are very versatile and in many cases can route signals across subsystems. They can also be used to route signals to/from the I/O connector. However, when a counter is used as part of a route, you may not be able to use the counter for other applications while the route remains reserved. Most routes do not require an internal counter terminal, but many advanced routes do. For example, if you want to use the signal present at PFI 4 on Dev1 as the Start Trigger for an acquisition on Dev2, you simply need to specify `/Dev1/PFI4` as the source of the trigger. However, to make the route, the signal is internally routed from `/Dev1/PFI4` to `/Dev1/Ctr0Source` to a RTSI bus line or PXI_Trig to `Dev2/ai/StartTrigger`. These terminals need not be explicitly specified when programming the route. In this case, it is not obvious that a counter terminal is used to make the route. Subsequent attempts at using the counter while it is in use result in a routing reservation error. To see if the route you are making uses counter resources, consult the table displayed under the Device Routes tab in MAX.

Routing Considerations for E Series and S Series Devices

- **PFI 0**—When exporting a signal through task-based routing to most PFI terminals, the route is reserved and committed with the task. When the task goes back to a verified state, software resources for the route are released, but the route remains in place in hardware. It remains in place to prevent glitching on the PFI terminal and to prevent any unexpected effects on external circuitry monitoring the signal. However, PFI 0 is an exception to this rule. Because PFI 0 can accept both analog and digital signals, it tristates when the task is not in the committed or running state. This behavior is intended to prevent accidental connections of an analog signal directly to digital circuitry that could damage the device.

  Note PFI 0 accepts only digital signals on the NI 6154.

  When in use, the analog trigger circuitry takes over the PFI 0 terminal internal to the device. Because of this, you cannot use PFI 0 to route any digital signals when using the analog trigger, regardless of whether you are triggering off of PFI 0 or an

analog input channel. If you try to use PFI 0 for digital signals and the analog trigger at the same time, you receive a routing error.

- **Counters**— The counters on these devices are very versatile and in many cases can route signals across subsystems. They can also be used to route signals to/from the I/O connector. However, when a counter is used as part of a route, you may not be able to use the counter for other applications while the route remains reserved. Most routes do not require an internal counter terminal, but many advanced routes do. For example, if you want to use the signal present at PFI 4 on Dev1 as the Start Trigger for an acquisition on Dev2, you simply need to specify `/Dev1/PFI4` as the source of the trigger. However, to make the route, the signal is internally routed from `/Dev1/PFI4` to `/Dev1/Ctr0Source` to a RTSI bus line or PXI_Trig to `Dev2/ai/StartTrigger`. These terminals need not be explicitly specified when programming the route. In this case, it is not obvious that a counter terminal is used to make the route. Subsequent attempts at using the counter while it is in use result in a routing reservation error. To see if the route you are making uses counter resources, consult the table displayed under the Device Routes tab in MAX.

Routing Considerations for TIO Devices

Though TIO counters can receive signals from any of the PFI lines, NI-DAQmx uses internal resources to connect some PFIs to counter inputs. There are some PFI lines that do not use internal resources that are preferred for use with different counter signals.

- **CtrnAux** —PFI 37, PFI 33, PFI 29, PFI 25, PFI 21, PFI 17, PFI 13, PFI 9
- **CtrnGate**—PFI 38, PFI 34, PFI 30, PFI 26, PFI 22, PFI 18, PFI 14, PFI 10
- **CtrnSource**—PFI 39, PFI 35, PFI 31, PFI 27, PFI 23, PFI 19, PFI 15, PFI 11

For output, the same rules apply. Though the counter can output on any PFI line, there are a subset of preferred PFIs that do not use internal resources to make the routes.

- **CtrnInternalOutput**—PFI 36, PFI 32, PFI 28, PFI 24, PFI 20, PFI 16, PFI 12, PFI 8

To see if the route you are making uses internal resources, consult the table displayed under the Device Routes tab in MAX.

Switches

This section contains information specific to switch devices about API support and switching capacity, including switching voltage, switching current, and switching power.

API Support for Switch Modules

Switch modules can support any of four different ways to control their relays. You may use the APIs interchangeably, but NI recommends using a single API for each application.

- **Digital Output**—Create your tasks, either programmatically with the Create Channel Digital Output function/VI or interactively through the DAQ Assistant, using the digital output physical channels. Use the digital versions of the Write function/VI to control the relays. Each digital port consists of 32 digital lines, and each line represents a relay on the switch. For example, if a module contains 64 relays, the first 32 are on port 0, and the rest will be on port 1. Writing a 0 to a digital line opens the relay and writing a 1 closes it.
- **Immediate**—The immediate API, supported by all switches, provides a switch channel-based interaction recommended for nonscanning operations. Functions/ VIs such as DAQmx Switch Connect and DAQmx Switch Disconnect are considered part of the immediate API.
- **Relay**—The relay API provides a relay-based interaction. Functions/VIs like DAQmx Switch Open Relays and DAQmx Switch Close Relays are considered part of the relay API.
- **Scanning**—Scanning is a method of connecting channels and is often used when connecting instruments and devices under test (DUTs) in a specific order. In this operation mode, the switch cycles through each entry in a scan list downloaded to the switch. The triggers the switch receives initiate this cycling. Create scanning tasks using DAQmx Switch Create Scan List and control tasks using functions/VIs like DAQmx Start, DAQmx Stop, and so on.

Supported Topologies

Every switch module supports one or more topologies. Changing the topology alters the functionality of the switch and, in many cases, changes the list of supported

channel names.

Special Considerations

Some switch modules have specific behaviors that you must consider when developing applications. These are described in the following table.

Device	Supported APIs	Supported Topologies
• NI 2810A/B • Reed Matrix for NI SwitchBlock	• Immediate • Relay	2810/1-Wire 4×43 Matrix
• NI 2811A/B • Reed Matrix for NI SwitchBlock	• Immediate • Relay	2811/1-Wire 8×21 Matrix
• NI 2812A/B • Reed Matrix for NI SwitchBlock	• Immediate • Relay	2812/1-Wire 16x9 Matrix
• NI 2813A/B • Reed Matrix for NI SwitchBlock	• Immediate • Relay	2813/2-Wire 4x21 Matrix
• NI 2814A/B • Reed Matrix for NI SwitchBlock	• Immediate • Relay	2814/2-Wire 8x9 Matrix
• NI 2815A/B • Reed Matrix for NI SwitchBlock	• Immediate • Relay	2815/1-Wire 4×86 Matrix
• NI 2816A/B • Reed Matrix for NI SwitchBlock	• Immediate • Relay	2816/1-Wire 8×46 Matrix

Device	Supported APIs	Supported Topologies
• NI 2817A/B • Reed Matrix for NI SwitchBlock	• Immediate • Relay	2817/1-Wire 16x22 Matrix
• NI 2833 • Electromechanical Latching Matrix for NI SwitchBlock	• Immediate • Relay	2833/2-Wire 4x71 Matrix
• NI 2834 • Electromechanical Latching Matrix for NI SwitchBlock	• Immediate • Relay	2834/2-Wire 8x34 Matrix
• NI 2865 • Matrix with Analog Bus Protection for NI SwitchBlock	• Immediate • Relay	2865/1-Wire 4x84 Matrix
• PXI-2501 • 24-Channel FET • Multiplexer/Matrix	• Immediate • Relay • Scanning	• 2501/1-Wire 48x1 Mux • 2501/1-Wire 48x1 Amplified Mux • 2501/2-Wire 24x1 Mux • 2501/2-Wire 24x1 Amplified Mux • 2501/2-Wire Dual 12x1 Mux • 2501/2-Wire Quad 6x1 Mux • 2501/2-Wire 4x6 Matrix • 2501/4-Wire 12x1 Mux
• PXI-2503 • 24-Channel Relay • Multiplexer/Matrix	• Immediate • Relay • Scanning	• 2503/1-Wire 48x1 Mux • 2503/2-Wire 24x1 Mux • 2503/2-Wire Dual 12x1 Mux

Device	Supported APIs	Supported Topologies
		• 2503/2-Wire Quad 6x1 Mux • 2503/2-Wire 4x6 Matrix • 2503/4-Wire 12x1 Mux
• PXI-2510 • 68-Channel 2A Fault Insertion Unit	• Immediate • Relay • Scanning	Independent
• PXI-2512, PXIe-2512 • 7-Channel 10A Fault Insertion Unit	• Immediate • Relay • Scanning	Independent
• PXI-2514, PXIe-2514 • 7-Channel 40A Fault Insertion Unit	• Immediate • Relay • Scanning	Independent
• PXI-2515, PXIe-2515 • High-Speed Digital I/O Signal Insertion Switch	• Immediate • Relay • Scanning	Independent
• PXI-2520 • 80-Channel SPST Relay Module	• Immediate • Relay • Scanning	• 80-SPST Switch
• PXI-2521 • 40-Channel DPST Relay Module	• Immediate • Relay • Scanning	• 40-DPST Switch

Device	Supported APIs	Supported Topologies
• PXI-2522 • 53-Channel SPDT Relay Module	• Immediate • Relay • Scanning	53-SPDT Switch
• PXI-2523 • 26-Channel DPDT Relay Module	• Immediate • Relay • Scanning	26-DPDT Switch
• PXI-2527 • 32-Channel 300 V Multiplexer	• Immediate • Relay • Scanning	• 2527/1-Wire 64x1 Mux • 2527/1-Wire Dual 32x1 Mux • 2527/2-Wire 32x1 Mux • 2527/2-Wire Dual 16x1 Mux • 2527/4-Wire 16x1 Mux • 2527/Independent
• PXI-2529 • 128-Crosspoint Relay Matrix	• Immediate • Relay • Scanning	• 2529/2-Wire 8x16 Matrix • 2529/2-Wire 4x32 Matrix • 2529/2-Wire Dual 4x16 Matrix
• PXI-2530 • 128-Channel Reed Relay • Multiplexer/Matrix	• Immediate • Relay • Scanning	• 2530/1-Wire 128x1 Mux • 2530/1-Wire Dual 64x1 Mux • 2530/2-Wire 64x1 Mux • 2530/4-Wire 32x1 Mux • 2530/1-Wire 4x32 Matrix • 2530/1-Wire 8x16 Matrix • 2530/1-Wire Octal 16x1 Mux • 2530/1-Wire Quad 32x1 Mux

Device	Supported APIs	Supported Topologies
		• 2530/2-Wire 4x16 Matrix • 2530/2-Wire Dual 32x1 Mux • 2530/2-Wire Quad 16x1 Mux • 2530/4-Wire Dual 16x1 Mux • 2530/Independent
• PXI-2531, PXIe-2531 • 512-Crosspoint Matrix	• Immediate • Relay • Scanning	• 2531/1-Wire 4x128 Matrix • 2531/1-Wire 8x64 Matrix • 2531/1-Wire Dual 4x64 Matrix • 2531/1-Wire Dual 8x32 Matrix
• PXI-2532 • 512-Crosspoint Matrix	• Immediate • Relay • Scanning	• 2532/1-Wire 16x32 Matrix • 2532/1-Wire 4x128 Matrix • 2532/1-Wire 8x64 Matrix • 2532/1-Wire Dual 16x16 Matrix • 2532/1-Wire Dual 4x64 Matrix • 2532/1-Wire Dual 8x32 Matrix • 2532/1-Wire Sixteen 2x16 Matrix • 2532/2-Wire 16x16 Matrix • 2532/2-Wire 4x64 Matrix • 2532/2-Wire 8x32 Matrix
• PXI-2533 • 256-Crosspoint SSR Matrix	• Immediate • Relay • Scanning	2533/1-Wire 4x64 Matrix

Device	Supported APIs	Supported Topologies
• PXI-2534 • 256-Crosspoint SSR Matrix	• Immediate • Relay • Scanning	2534/1-Wire 8x32 Matrix
• PXI-2535 • 544-Crosspoint FET Matrix	• Immediate • Relay • Scanning	2535/1-Wire 4x136 Matrix
• PXI-2536 • 544-Crosspoint FET Matrix	• Immediate • Relay • Scanning	2536/1-Wire 8x68 Matrix
• PXI-2542, PXIe-2542 • Quad Terminated 2x1 Multiplexer	• Immediate • Relay • Scanning	2542/Quad 2x1 Terminated Mux
• PXI-2543, PXIe-2543 • Dual Terminated 4x1 Multiplexer	• Immediate • Relay • Scanning	2543/Dual 4x1 Terminated Mux
• NI PXI-2544, PXIe-2544 • Terminated 8x1 Multiplexer	• Immediate • Relay • Scanning	2544/8x1 Terminated Mux
• PXI-2545 • 2.7 GHz 4x1 Terminated 50 Ohm Multiplexer	• Immediate • Relay • Scanning	2545/4x1 Terminated Mux
• PXI-2546	• Immediate	2546/Dual 4x1 Mux

Device	Supported APIs	Supported Topologies
• 2.7 GHz Dual 4x1 50 Ohm Multiplexer	• Relay • Scanning	
• PXI-2547 • 2.7 GHz 8x1 50 Ohm Multiplexer	• Immediate • Relay • Scanning	2547/8x1 Mux
• PXI-2548 • 2.7 GHz 4-SPDT 50 Ohm Relay Module	• Digital Output • Immediate • Relay • Scanning	2548/4-SPDT
• PXI-2549 • 2.7 GHz Terminated 2-SPDT 50 Ohm Relay Module	• Digital Output • Immediate • Relay • Scanning	2549/Terminated 2-SPDT
• PXI-2554 • 2.5 GHz 4x1 75 Ohm Multiplexer	• Immediate • Relay • Scanning	2554/4x1 Mux
• PXI-2555 • 2.5 GHz 4x1 Terminated 75 Ohm Multiplexer	• Immediate • Relay • Scanning	2555/4x1 Terminated Mux
• PXI-2556 • 2.5 GHz Dual 4x1 75 Ohm Multiplexer	• Immediate • Relay • Scanning	2556/Dual 4x1 Mux

Device	Supported APIs	Supported Topologies
• PXI-2557 • 2.5 GHz 8x1 75 Ohm Multiplexer	• Immediate • Relay • Scanning	2557/8x1 Mux
• PXI-2558 • 2.5 GHz 4-SPDT 75 Ohm Relay Module	• Digital Output • Immediate • Relay • Scanning	2558/4-SPDT
• PXI-2559 • 2.5 GHz Terminated 2-SPDT 75 Ohm Relay Module	• Digital Output • Immediate • Relay • Scanning	2559/Terminated 2-SPDT
• PXI-2564 • 16-SPST Relay Module	• Digital Output • Immediate • Relay • Scanning	• 2564/8-DPST • 2564/16-SPST
• PXI-2565 • 16-SPST Power Relay Module	• Digital Output • Immediate • Relay • Scanning	2565/16-SPST
• PXI-2566 • 16-SPDT Relay Module	• Digital Output • Immediate • Relay	• 2566/8-DPDT • 2566/16-SPDT

Device	Supported APIs	Supported Topologies
	• Scanning	
• PXI-2567 • 64-Channel Relay Driver Module	• Digital Output • Immediate • Relay • Scanning	2567/Independent
• PXI-2568 • 31-Channel SPST Relay Module	• Digital Output • Immediate • Relay • Scanning	• 2568/15-DPST • 2568/31-SPST
• PXI-2569 • 100-Channel SPST Relay Module	• Digital Output • Immediate • Relay • Scanning	• 2569/50-DPST • 2569/100-SPST
• PXI-2570 • 40-Channel SPDT Relay Module	• Digital Output • Immediate • Relay • Scanning	• 2570/20-DPDT • 2570/40-SPDT
• PXI-2571 • 66-Channel SPDT Relay Module	• Digital Output • Immediate • Relay • Scanning	2571/66-SPDT

Device	Supported APIs	Supported Topologies
• PXI-2575 • 196x1 Relay Multiplexer	• Immediate • Relay • Scanning	• 2575/1-Wire 196x1 Mux • 2575/2-Wire 98x1 Mux • 2575/2-Wire 95x1 Mux
• PXI-2576 • Multi-Bank Multiplexer	• Immediate • Relay • Scanning	• 2576/2-Wire Octal 8x1 Mux • 2576/2-Wire Sixteen 4x1 Mux
• PXI-2584 • High-Voltage Multiplexer	• Immediate • Relay • Scanning	• 2584/Independent • 2584/1-Wire 12x1 Mux • 2584/1-Wire Dual 6x1 Mux • 2584/2-Wire 6x1 Mux
• PXI-2585 • 10-Channel Multiplexer	• Immediate • Relay • Scanning	2585/1-Wire 10x1 Mux
• PXI-2586 • 10-Channel SPST Relay Module	• Digital Output • Immediate • Relay • Scanning	• 2586/5-DPST • 2586/10-SPST
• PXI-2590 • 1.3 GHz 4x1 50 Ohm Multiplexer	• Immediate • Relay • Scanning	2590/4x1 Mux
• PXI-2591 • 4 GHz 4x1 50 Ohm Multiplexer	• Immediate • Scanning	2591/4x1 Mux

Device	Supported APIs	Supported Topologies
• PXI-2593 • 500 MHz Dual 8x1 50 Ohm Multiplexer	• Immediate • Relay • Scanning	• 2593/16x1 Mux • 2593/Dual 8x1 Mux • 2593/8x1 Terminated Mux • 2593/Dual 4x1 Terminated Mux • 2593/Independent
• PXI-2594 • 1x4 2.5 GHz Multiplexer	• Immediate • Relay • Scanning	2594/4x1 Mux
• PXI-2595 • 1x4 5.5 GHz Multiplexer	• Immediate • Relay • Scanning	2595/4x1 Mux
• PXI-2596 • Dual 1x6 26.5 GHz Multiplexer	• Immediate • Relay • Scanning	2596/Dual 6x1 Mux
• PXI-2597 • 1x6 26.5 GHz Terminated Multiplexer	• Immediate • Relay • Scanning	2597/6x1 Terminated Mux
• PXI-2598 • Dual 26.5 GHz Transfer Switch	• Immediate • Relay • Scanning	2598/Dual Transfer
• PXI-2599 • Dual 26.5 GHz SPDT	• Immediate • Relay • Scanning	2599/2-SPDT

Device	Supported APIs	Supported Topologies
• PXIe-2720 • Ten 8-Bit Channel Resistor Module	• Immediate • Relay • Scanning	Independent
• PXIe-2722 • Five 16-Bit Channel Resistor Module	• Immediate • Relay • Scanning	Independent
• PXIe-2725 • Eighteen 8-Bit Channel Resistor Module	• Immediate • Relay • Scanning	Independent
• PXIe-2727 • Nine 16-bit Channel Resistor Module	• Immediate • Relay • Scanning	Independent
• PXIe-2790 • RF Power Combiner and Switch	• Immediate • Relay • Scanning	Independent
• PXIe-2796 • Dual 6x1 40 GHz Multiplexer	• Immediate • Relay • Scanning	2796/Dual 6x1 Mux
• PXIe-2797 • 6x1 40 GHz Terminated Multiplexer	• Immediate • Relay • Scanning	2797/6x1 Terminated Mux
• PXIe-2798	• Immediate	2798/Dual Transfer

Device	Supported APIs	Supported Topologies
• Dual 40 GHz Transfer Switch	• Relay • Scanning	
• PXIe-2799 • Dual 40 GHz SPDT	• Immediate • Relay • Scanning	2799/2-SPDT
• SCXI-1127 • 32-Channel Relay Multiplexer/Matrix • See SCXI-1127 Considerations	• Immediate • Relay • Scanning	• 1127/1-Wire 64x1 Mux • 1127/2-Wire 32x1 Mux • 1127/4-Wire 16x1 Mux • 1127/2-Wire 4x8 Matrix
• SCXI-1128 • 32-Channel Solid-State Relay (SSR) • Multiplexer/Matrix • See SCXI-1128 Considerations	• Immediate • Relay • Scanning	• 1128/1-Wire 64x1 Mux • 1128/2-Wire 32x1 Mux • 1128/4-Wire 16x1 Mux • 1128/2-Wire 4x8 Matrix • 1128/Independent
• SCXI-1129 • 256-Crosspoint Relay Matrix	• Immediate • Relay • Scanning	• 1129/2-Wire 16x16 Matrix • 1129/2-Wire 8x32 Matrix • 1129/2-Wire 4x64 Matrix • 1129/2-Wire Dual 8x16 Matrix • 1129/2-Wire Dual 4x32 Matrix • 1129/2-Wire Quad 4x16 Matrix
• SCXI-1130 • 256-Channel Reed Relay • Multiplexer/Matrix	• Immediate • Relay • Scanning	• 1130/1-Wire 256x1 Mux • 1130/1-Wire Dual 128x1 Mux • 1130/2-Wire 128x1 Mux

Device	Supported APIs	Supported Topologies
		• 1130/4-Wire 64x1 Mux • 1130/1-Wire 4x64 Matrix • 1130/1-Wire 8x32 Matrix • 1130/1-Wire Octal 32x1 Mux • 1130/1-Wire Quad 64x1 Mux • 1130/1-Wire Sixteen 16x1 Mux • 1130/2-Wire 4x32 Matrix • 1130/2-Wire Octal 16x1 Mux • 1130/2-Wire Quad 32x1 Mux • 1130/4-Wire Quad 16x1 Mux • 1130/Independent
• SCXI-1160 • 16-SPDT General-Purpose Relay Module	• Digital Output • Immediate • Relay	1160/16-SPDT
• SCXI-1161 • 8-SPDT Power Relay Module	• Digital Output • Immediate • Relay	1161/8-SPDT
• SCXI-1163R • 32-Channel SSR	• Digital Output • Immediate • Relay	1163R/Octal 4x1 Mux

Device	Supported APIs	Supported Topologies
• SCXI-1166 • 32-SPDT Relay Module	• Digital Output • Immediate • Relay • Scanning	• 1166/16-DPDT • 1166/32-SPDT
• SCXI-1167 • 64-Channel Relay Driver Module	• Digital Output • Immediate • Relay • Scanning	1167/Independent
• SCXI-1169 • 100-Channel SPST Relay Module	• Digital Output • Immediate • Relay • Scanning	• 1169/50-DPST • 1169/100-SPST
• SCXI-1175 • 196x1 Relay Multiplexer	• Immediate • Relay • Scanning	• 1175/1-Wire 196x1 Mux • 1175/2-Wire 98x1 Mux • 1175/2-Wire 95x1 Mux
• SCXI-1190 • 1.3 GHz Quad 4x1 50 Ohm Multiplexer	• Immediate	1190/Quad 4x1 Mux
• SCXI-1191 • 4 GHz Quad 4x1 50 Ohm Multiplexer	• Immediate	1191/Quad 4x1 Mux
• SCXI-1192 • 18 GHz 8-SPDT 50 Ohm Relay Module	• Digital Output • Immediate	1192/8-SPDT

Device	Supported APIs	Supported Topologies
	• Relay	
• SCXI-1193 • 500 MHz Quad 8x1 50 Ohm Multiplexer	• Immediate • Relay • Scanning	• 1193/32x1 Mux • 1193/Dual 16x1 Mux • 1193/Quad 8x1 Mux • 1193/16x1 Terminated Mux • 1193/Dual 8x1 Terminated Mux • 1193/Quad 4x1 Terminated Mux • 1193/Independent
• SCXI-1194 • Quad 1x4 2.5 GHz Multiplexer	• Immediate • Relay • Scanning	1194/Quad 4x1 Mux
• SCXI-1195 • Quad 1x4 5.5 GHz Multiplexer	• Immediate • Relay • Scanning	1195/Quad 4x1 Mux

SCXI-1127 Considerations

To route signals to the analog bus backplane, you must enable the switch device property AutoConnAnlgBus. As a result, if you connect a channel (ch1) to the common channel (com0), the signal is automatically routed from com0 to the analog bus (ab0).

The device supports only continuous scanning.

If you have used immediate or relay operations to change relay states before starting a scan, all of those relays are opened when the scan starts. After the scan completes, the relays are returned to their previous state prior to the scan.

Early revisions of this hardware reserve the SCXI_TRIG1 line. If you place an older revision of this hardware (earlier than revision E) into an SCXI chassis that also contains an SCXI analog input module that performs track and hold (such as the SCXI-1140 or SCXI-1520), you may get reservation errors when trying to use the SCXI analog input module.

SCXI-1128 Considerations

To route signals to the analog bus backplane, you must enable the switch device property AutoConnAnlgBus. As a result, if you connect a channel (ch1) to the common channel (com0), the signal is automatically routed from com0 to the analog bus (ab0).

The device supports only continuous scanning.
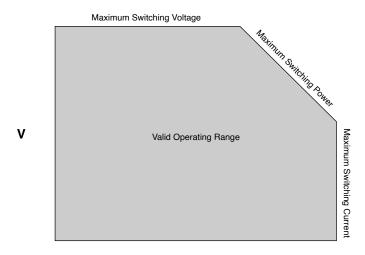
If you have used immediate or relay operations to change relay states before starting a scan, all of those relays are opened when the scan starts. After the scan completes, the relays are returned to their previous state prior to the scan.

Switching Capacity

Signal levels through a switch must account for the following specifications:

- Switching voltage
- Switching current
- Switching power

The following figure shows the valid operating range defined by these limits.

Related concepts:

- [Switching Voltage](#)
- [Switching Current](#)
- [Switching Power](#)

Switching Current

Switching current is the maximum rated current that can flow through the switch as it makes or breaks a contact. Switching active currents higher than those specified in the switch documentation can result in arcing that can damage the contacts of electromechanical relays. A minimum current specification indicates the smallest current that can reliably flow through the switch.

Switches can also have carry current. **Carry current** is the maximum rated current that can pass through a closed switch, or the current that a switch can carry. Carry current can be larger than switching current. However, since carry current can only flow while the switch is closed, the current needs to be stopped while the switch operates.

Switching Power

Switching power is the limit on the combined open-contact voltage and closed-contact current of a signal in the switch.

Switching Power = Switching Voltage * Switching Current

Switching high-power signals causes high-energy arcing at the electromechanical contacts during actuation, reducing the useful life of the switch.

Switching Voltage

Switching voltage refers to the maximum signal voltage that the switch module can safely maintain. Switching voltage is defined from channel-to-ground and from channel-to-channel. Channel-to-ground is the voltage potential between the signal line and the grounded chassis. Channel-to-channel is the voltage potential between any pair of signal lines within the module. This voltage includes voltages across open relay contacts, as well as voltages between adjacent connection terminals.

> **Note** CE marking for measurement and control devices requires compliance to the IEC 61010-1 standard. Switch modules intended for high-voltage signals (> 60 VDC / 30 V_{rms}) are rated for Measurement Categories as defined in this standard. Measurement Categories describe the acceptable transient overvoltages and fault protection necessary for safe operation. Refer to the **NI Switches Getting Started Guide** for more information on Measurement Categories.

Synchronization

This section contains information on synchronizing multiple devices.

Synchronizing cDAQ Chassis and FieldDAQ Devices

If you have multiple devices that support the IEEE 802.1AS standard, you can use Start Trigger synchronization to trigger at a specific time. These devices must either be connected through a daisy chain or a switch that supports IEEE 802.1AS.

> **Note** If the devices are connected to the network through a real-time host, you must discover these devices in MAX through the real-time host. Refer to the MAX Help for NI-DAQmx for more information on how to discover devices

remotely.

For a detailed tutorial on synchronizing FieldDAQ devices, refer to **Synchronizing Analog Input FieldDAQ Devices with NI-DAQmx in LabVIEW** on ni.com.

Related information:

- [Synchronizing Analog Input FieldDAQ Devices with NI-DAQmx in LabVIEW](#)

Synchronizing DSA Devices

You can synchronize the analog input and output operations on two or more DSA devices to extend the channel count of DSA measurements. Refer to the **NI Dynamic Signal Acquisition User Manual**, which you can find at ni.com/manuals, for additional DSA synchronization information.

Synchronizing DSA Devices with Multifunction DAQ Devices

You can use Sample Clock synchronization to synchronize DSA devices with Multifunction DAQ devices. However, synchronization creates phase delay between the devices.

Related concepts:

- [Sample Clock Synchronization](#)

Synchronizing X Series, M Series, and SC Express Devices

Note To synchronize analog input tasks on multiple X Series and SC Express devices at the same sampling rate, you can use channels from those devices within the same task.

X Series, M Series, and SC Express devices support the following synchronization methods:

- Start Trigger synchronization
- Reference Clock synchronization
- Sample Clock synchronization

Refer to the Device Routes tab in MAX to determine which PXI and RTSI lines can route synchronization signals.

> ✎ **Note** You cannot use a RTSI cable to connect multiple USB X Series devices.

> ✎ **Note** The NI 4302/4303/4304/4305, NI 4330/4331, and NI 4339 only support exporting their sample clock. They do not support importing a sample clock.

> ✎ **Note** The NI 4340 supports importing a sample clock only in hardware-timed single-point sample mode.

Related concepts:

- [Multidevice Tasks](#)
- [Start Trigger Synchronization](#)
- [Reference Clock Synchronization](#)
- [Sample Clock Synchronization](#)

Synchronizing E Series, S Series, and AO Series Devices

E Series, S Series, and AO Series devices support the following synchronization methods:

- Start Trigger synchronization
- Master Timebase synchronization
- Sample Clock synchronization

NI PCI-6154 and PXIe-6124 devices support the following synchronization methods:

- Start Trigger synchronization
- Reference Clock synchronization
- Sample Clock synchronization

Refer to the Device Routes tab in MAX to determine which PXI and RTSI lines can route synchronization signals.

Related concepts:

- Start Trigger Synchronization
- Master Timebase Synchronization
- Sample Clock Synchronization
- Reference Clock Synchronization

PXI_CLK 10 with the NI 6608 and the NI 6614

When NI-DAQmx detects that an NI 6608 is installed in slot 2 (or when an NI 6614 is installed in the system timing slot), NI-DAQmx automatically overrides the chassis PXI_CLX10 with the more stable oven controlled oscillator (OCXO) from the NI 6608 or NI 6614. This allows all devices in the chassis to inherit the OCXO accuracy and stability via the PXI_CLK10 signal. In a PXIe chassis, the PXIe_Clk100 signal is phase-locked to the PXI_CLK10, so PXIe_Clk100 will also inherit the OCXO accuracy and stability.

Refer to the NI 6608 documentation for more information about the stability of the OCXO.

> ✎ **Note** For the automatic override to occur in NI-DAQmx, you must put the NI 6608 in slot 2 or the NI 6614 in the system timing slot. For more information about configuring your PXI chassis, refer to the ***Measurement & Automation Explorer Help for PXI***.

Synchronization with M Series USB and NI ELVIS II Family Devices

M Series USB and NI ELVIS II Family devices do not support reference clock synchronization.

Supported Devices for Trigger Skew Correction

The following devices support locking synchronized triggers to a clock to compensate

for skew.

- SC Express devices
- X Series devices

Related concepts:

- [Trigger Skew Correction](#)

Timing

This section contains information about timing for AO Series, C Series, DSA, E Series, FieldDAQ, M Series USB, NI ELVIS II Family, S Series, and SC Express devices.

Timing Considerations for AO Series Devices

When using an external ao/SampleClock for finite generations, you need to provide one more sample clock pulse than the number of samples in the generation. The Wait Until Done function/VI uses the extra sample clock to indicate the task is complete. For example, if you want to generate 1000 samples using an external sample clock, the first 1000 samples clocks you provide generates all of the samples, but you need to provide 1001 sample clock pulses for the Wait Until Done function/VI to indicate the task is done. If you are trying to synchronize an analog output generation with another acquisition or generation by sharing a common clock, use the ao/SampleClock as the master clock, or key off of the generation or acquisition providing the master clock to determine when the generation is complete.

Static AO devices, such as the NI 6703 and NI 6704, do not have hardware timing and have multiplexed output. Refer to your device documentation for specifics concerning your device.

Timing Considerations for C Series Devices

> **Note** C Series devices do not support hardware-timed single-point sample mode or Wait for Next Sample Clock.

Related concepts:

- C Series Delta-Sigma Devices
- C Series Scanned Devices
- C Series Slow Sample Devices
- Digital I/O Considerations for C Series and TestScale Devices
- Digital I/O Considerations for C Series Devices

Analog Input Timing Considerations for C Series Devices

You can use multiple analog input devices of different types in the same task, and NI-DAQmx automatically synchronizes them.

NI-DAQmx supports:

- Only one analog input task at a time per NI cDAQ-9171, 9181, and 9191 chassis.
- Up to three analog input tasks at a time per cDAQ-9132, 9133, 9134, 9135, 9136, 9137, 9174, 9178, 9179, 9184, 9185, 9188, 9188XT, and 9189 chassis.
- Up to eight analog input tasks at a time per supported CompactRIO[39] controller.
- One analog input task at a time per slot or onboard IO module, per CompactRIO Single-Board[40] controller.

Related concepts:

- CompactRIO Timing Engines

AI Convert Clock Considerations

C Series Scanned devices, such as the NI 9201, NI 9205, and NI 9221, use multiplexed sampling controlled by a per-slot AI Convert Clock.

If you have multiple devices in one task, their AI Convert Clocks run in parallel, which may cause channels on multiple devices to be sampled at the same time. You can set the AI Convert Rate and the Delay From Sample Clock differently on each device. When setting AI Convert instances of the DAQmx Timing attribute/property, you must use the ActiveDev attribute/property to specify the device to which you are referring. External

---

39. cRIO-9040, 9041, 9042, 9043, 9045, 9046, 9047, 9048, 9049, 9053, 9054, 9055, 9056, 9057, and 9058.
40. sbRIO-9603, 9608, 9609, 9628, 9629, and 9638

clocking of the AI Convert Clock is not supported.

Reference Clock Considerations

Devices with a reference clock, such as the NI 9775, do not support on-demand timing.

These devices require hardware timing from a continuous clock. This clock synchronizes the onboard oscillators using a phase-locked loop and serves as the Sample Clock Timebase. When devices with reference clocks are in a task, you cannot set an external SampClk.Src or SampClkTimebase. Similar to DSA devices, devices with a reference clock do not support external sample clocks from arbitrary signal sources.

For the NI 9775, the reference clock must be 12.8 MHz.

Sampling Rate Considerations

The default AI Convert Clock rate for the C Series Scanned devices uses 10 µs of additional settling time between channels, compared to the fastest AI Convert Clock rate for the device. When the Sample Clock rate is too high to allow for 10 µs of additional settling time, the default AI Convert Clock rate uses as much settling time as is allowed by the Sample Clock rate. If there are multiple C Series Scanned devices in the same task, the same amount of additional settling time is used for all devices in the task, even if their maximum AI Convert Clock rates differ.

In a CompactDAQ chassis with C Series Slow Sample devices, such as the NI 9211, if the sampling rate of a hardware-timed acquisition exceeds the maximum sampling rate of the module, the most recently acquired sample may be read multiple times and no warning or error is generated. Exceeding the maximum sampling rate of other devices in the same task generates warnings or errors. The first sample of a hardware-timed acquisition with C Series Slow Sample devices is sampled when the task is committed. Software-timed acquisitions with C Series Slow Sample devices always wait for a new sample to be acquired.

In a CompactRIO or Single-Board RIO controller with C Series Slow Sample devices, such as the NI 9211, if the sampling rate of a hardware-timed acquisition exceeds the maximum sampling rate of the module, DAQmx generates warning or errors. When a Slow Sample device is in the same task as a non-Slow Sample device, exceeding the maximum sampling rate of the Slow Sample device results in the most recently

acquired sample being read multiple times. In this scenario, the first sample of a hardware-timed acquisition with C Series Slow Sample devices is sampled when the task is committed.

The NI 9213/9214 maximum sample rate of 100 S/s applies to tasks with 13 or less analog input channels per module. To determine the maximum sample rate for an NI 9213/9214, query the AI.MaxMultiChanRate attribute/property and divide that returned value by the number of channels. For instance, if your analog input task uses 14 channels and AI.MaxMultiChanRate returns 1360.54, the maximum sample rate would be 97.18 S/s. You can also query the maximum sample rate used in the task with the SampClk.MaxRate attribute/property.

The maximum sampling rate of the NI 9215 depends on which channel(s) you are acquiring from. For instance, a task acquiring from any combination of ai0, ai1, and ai2 can sample at faster rates than a task that includes ai3. The maximum sampling rates are attainable only when sampling from ai0. If you have multiple NI 9215 devices in the same task, they sample in parallel. For instance, multiple NI 9215 devices acquiring from ai0 may be able to achieve a faster sampling rate than a single NI 9215 acquiring from ai3.

The C Series Delta-Sigma devices, such as the NI 9225, NI 9227, NI 9235, and NI 9239, have both a maximum and a minimum sampling rate. Refer to the specifications for your device to determine the sampling rate range.

When you set a sampling rate for the NI 9230, NI 9231, NI 9232, or NI 9234, NI-DAQmx selects the highest possible decimation rate. Refer to the operating instructions for your device for more information about available sampling rates.

When C Series Delta Sigma devices, such as the NI 9230, NI 9231, NI 9232, NI 9234, NI 9250, or NI 9251 are in a task with a C Series device that has a different sample clock timebase, NI-DAQmx always chooses the sample clock timebase with the highest frequency. To override this selection, you can set the sample clock timebase in the SampClk.Timebase.Src attribute/property.

Hardware and On-Demand Timing for C Series Delta-Sigma Devices

C Series Delta-Sigma devices do not support the on-demand timing type.

All acquisitions and generations for C Series Delta-Sigma devices require hardware timing from a steady clock. You cannot set the SampClk.Src attribute/property to an external source when a C Series Delta-Sigma device is in the task. With C Series Delta-Sigma devices, external clocking from arbitrary external signal sources such as encoders and tachometers is not supported.

Analog Output Timing Considerations for C Series Devices

Only one hardware-timed analog output task per CompactDAQ chassis at a given time is supported, but the number of concurrent software-timed analog output tasks is limited only by the available channels. A single C Series analog output device cannot be used for hardware-timed and software-timed tasks at the same time.

CompactRIO and Single-Board RIO controllers can support more than one hardware-timed AO task per controller. Refer to **CompactRIO Timing Engines**.

Related concepts:

- CompactRIO Timing Engines

Digital Input/Output Timing Considerations for C Series Devices

Only one hardware-timed digital input task and one hardware-timed digital output task per CompactDAQ chassis at a given time is supported, but the number of concurrent software-timed digital I/O tasks is limited only by the available lines.

Refer to **Digital I/O Considerations for C Series** for more information about hardware-timed digital input/output.

CompactRIO and Single-Board RIO controllers can support more than one hardware-timed digital input task and one hardware-timed digital output task per controller. Refer to **CompactRIO Timing Engines**.

Related concepts:

- CompactRIO Timing Engines
- Digital I/O Considerations for C Series and TestScale Devices
- Digital I/O Considerations for C Series Devices

Configurable Timing for C Series Devices

On the NI 9207, NI 9208, NI 9209, NI 9212, NI 9213, NI USB-9213, NI 9214, NI 9216, NI 9217, NI 9219, NI 9224, NI 9226, NI 9228, and NI 9775, you can configure high-speed or high-resolution measurements using the AI.ADCTimingMode attribute/property.

On the NI 9212, NI 9219, and NI USB-9219 you can also configure low-noise measurements using the AI.ADCTimingMode attribute/property.

Default Settings for the AI.ADCTimingMode Attribute/Property

For all modules in the CompactRIO and Single-Board RIO controllers, the default value in hardware-timed mode is automatically determined based on Sample Clock Rate.

For the NI 9207, NI 9208, NI 9209, NI 9212, NI 9213, NI USB-9213, NI 9214, NI 9216, NI 9217, NI 9219, and NI 9226 in CompactDAQ, the AI.ADCTimingMode attribute/property is set to High Resolution by default. To increase the conversion rate, set this attribute/property to High Speed.

For the NI USB-9219, this attribute/property is set to High Resolution by default in on-demand mode, and the default value in hardware-timed mode is automatically determined based on Sample Clock Rate. To increase power line noise rejection on the NI 9212 and NI 9219, set this attribute/property to Best 60 Hz Rejection or Best 50 Hz Rejection.

For the NI 9224 and NI 9228, this attribute/property is set by default to Automatic, which causes the module to sample with the highest resolution timing mode that is faster than the rate you specified.

For the NI 9775, this attribute/property has a default timing mode of Automatic. The Automatic timing mode automatically configures the module to sample with the highest resolution timing mode that is compatible with the user-specified rate. If the AI.Lowpass.Enable attribute/property is set to false, the timing mode will remain in High Speed regardless of the user-specified sample rate.

Slow sample modules are able to sample just fast enough to avoid returning repeated data. If the user-specified rate is above the maximum rate, the module will enter background convert and return repeated data. Slow sample modules on the

cDAQ-9171 and cDAQ-9191 chassis have a default timing mode of Automatic.

The AI.ADCTimingMode attribute/property affects both the maximum and default values for AIConv.Rate attribute/property in the DAQmx Timing property node. For instance, if the ADC timing mode corresponds to a conversion time of 200 ms, the maximum conversion rate is 5 Hz.

Counter Input Timing Considerations for C Series Devices

For the NI 9361, you can use multiple counter input devices in the same task, and NI-DAQmx automatically synchronizes them. A single NI 9361 cannot be used for hardware-timed and software-timed tasks at the same time. The CompactDAQ chassis and CompactRIO controllers support up to four of the onboard chassis counters at a time.

Implicit timing is not supported on the NI 9361.

Related concepts:

- [CompactRIO Timing Engines](#)

Timing Considerations for DSA Devices

- Supported Sampling Rates

  Unlike some other DAQmx devices, DSA devices have both a maximum and a minimum sampling rate. Refer to the specifications for your device to determine the sampling rate range.

- Other DSA Timing Considerations

  DSA devices do not support the on-demand timing type. All DSA acquisitions and generations require hardware timing from a steady clock. The NI 4464, 4480, and 4481 are an exception. The NI 4464, 4480, and 4481 support on-demand timing type.

  DSA devices do not support external clocking from arbitrary external signal sources such as encoders and tachometers. The PFI lines on DSA devices cannot

accept external clocks. You can program a DSA device to use an external clock only when it is a slave in multi-device synchronized system. Refer to Synchronizing DSA Devices for more details.

Related concepts:

- [Synchronizing DSA Devices](#)

Timing Considerations for E Series Devices

Special timing considerations when using E Series devices:

- ai/ConvertClock

  When using the ai/ConvertClock as the source of a route, one extra convert pulse is generated than you might expect. For example, if you perform a finite acquisition of 100 samples with four channels, you see 401 convert pulses instead of 400. This extra convert pulse is necessary to set up the configuration memory in hardware and occurs as the task transitions to the committed state.

- ao/SampleClock

  When using an external ao/SampleClock for finite generations, you need to provide one extra sample clock pulse than the number of samples in the generation for the Wait Until Done function/VI to indicate the task is complete. For example, if you want to generate 1000 samples using an external sample clock, you need to provide 1001 sample clock pulses or the Wait Until Done function/VI never indicates the task is done. All of the samples are generated, but the analog output timing engine needs one additional clock pulse to indicate the generation is complete. If you are trying to synchronize an analog output generation with another acquisition or generation by sharing a common clock, use the ao/SampleClock as the master clock or key off of the generation or acquisition providing the master clock to determine when the generation is complete.

Timing Considerations for FieldDAQ

Sampling Rate Considerations

For the FD-11613 and FD-11614, if the sampling rate of a hardware-timed acquisition exceeds the maximum sampling rate of the module, the most recently acquired sample may be read multiple times and no warning or error is generated. Exceeding the maximum sampling rate of other devices in the same task generates warnings or errors. The first sample of a hardware-timed acquisition with these devices is sampled when the task is committed. Software-timed acquisitions with these devices always wait for a new sample to be acquired.

The FD-11601, FD-11603, FD-11605, FD-11634, and FD-11637 have a user-selectable sample clock timebase. You can set the sample clock timebase with the SampClk.Timebase.Src attribute/property. The FD-11601, FD-11603, FD-11605, FD-11634, and FD-11637 also have both a maximum and a minimum sampling rate. Refer to your device documentation to determine the sampling rate range and the user-selectable timebase for your FieldDAQ device.

If you do not specify a sample clock timebase source or rate, NI-DAQmx auto-selects the sample clock timebase with the closest sampling rate greater than or equal to your requested sampling rate. If you do specify the sample clock timebase source or rate, NI-DAQmx respects the setting and only chooses a sampling rate supported with that sample clock timebase.

On-Demand Timing

The FD-11601, FD-11603, FD-11605, FD-11634, and FD-11637 do not support the on-demand timing type.

Configurable Timing for FD-11613 and FD-11614

The FD-11613 and FD-11614 support the following timing modes:

High Resolution	Optimizes accuracy and noise and rejects power line frequencies.
Best 50 Hz Rejection	Optimizes 50 Hz noise rejection.
Best 60 Hz Rejection	Optimizes 60 Hz noise rejection.

High Speed	Optimizes sample rate and signal bandwidth.
Automatic	Uses the most appropriate supported timing mode based on the Sample Clock Rate.

Use the AI.ADCTimingMode attribute/property to configure timing on the FD-11613 and FD-11614. This attribute/property is set by default to Automatic, which causes the module to sample with the highest resolution timing mode that is faster than the rate you specified.

Hardware-Timed Non-Buffered Sample Mode

In hardware-timed non-buffered sample mode, samples are acquired or generated continuously using hardware timing and no buffer. You enable this mode by using sample clock or change detection timing types and specifying a buffer size of 0. This mode is similar to hardware-timed single point sample mode but with slower performance and no error checking. Because there is no buffer if you use hardware-timed non-buffered sample mode, you should ensure that reads or writes execute fast enough to keep up with hardware timing.

> **Note** NI USB devices and NI CompactDAQ systems do not support hardware-timed non-buffered sample mode when sample clock timing is used.

Related concepts:

- Sample Timing Types
- Hardware-Timed Single Point Sample Mode

Timing Considerations with NI ELVIS II Family and M Series USB Devices

NI ELVIS II and M Series USB devices do not support hardware-timed single-point sample mode or Wait for Next Sample Clock. The oscilloscope channels on NI ELVIS II+, scopeCh0 and scopeCh1, support only finite samples mode, not hardware-timed single-point mode or continuous samples mode. The NI ELVIS II+ oscilloscope channels also only support the sample clock sample timing type.

Non-Buffered Change Detection

On X Series, M Series, and C Series devices, you can detect changes on digital lines or ports without using a buffer by specifying change detection timing with a buffer size of 0. To use change detection timing on port 1 or port 2 for X Series devices, you must use this non-buffered approach (i.e. specifying a buffer size of 0). On M Series devices, you can read from port 1 and port 2 using non-buffered change detection timing, but changes must be detected on port 0.

Timing Considerations for S Series

Analog Input with Pipelined ADCs

> **Note** Not all S Series devices have pipelined ADCs. Refer to the specifications for your device to determine if your device contains pipelined ADCs.

Many S Series devices have pipelined ADCs with an intrinsic pipeline depth. This pipelining allows the device to sample at higher rates, but it also has other consequences on the timing requirements for the device. S Series devices, except for the NI 6143, do not support AI hardware-timed single-point sample mode. Since the data needs to travel through the pipeline before it can be read, the data being read is always pipeline-depth points old. For instance, if the pipeline depth for a device is three, the first sample is acquired on clock tick 1, but it is not available for reading until clock tick 4. Following this logic, you must supply pipeline-depth extra clock pulses for a finite acquisition to flush the pipeline. Continuing with the previous example, if the pipeline depth is three and you want to acquire 1000 samples, you need to generate 1003 sample clock pulses. If you are using the onboard sample clock, NI-DAQmx automatically generates the appropriate number of sample clock pulses. However, when using an external sample clock or when synchronizing devices, you need to ensure you supply the appropriate number of sample clock pulses.

There is also a finite amount of time a sample can be held in the pipeline before it starts to degrade and lose measurement accuracy. This time limit imposes a minimum sampling rate that must be met to achieve the measurement accuracy specified for the device. Although you can sample slower than this minimum recommend sampling rate, the accuracy specifications for the device are not guaranteed. Refer to the

specifications for your device to determine the recommended minimum sampling rate.

This degradation of samples in the pipeline also affects on-demand single-point acquisitions and acquisitions that use a Pause Trigger. For on-demand single-point acquisitions, NI-DAQmx generates multiple sample clocks at the maximum sample rate of the device for each sample that is read. For S Series devices with a pipelined ADC, the number of sample clocks generated is equal to the pipeline depth plus one. For S Series devices that do not have a pipelined ADC, two sample clock pulses are generated for each point. This means that if you export the sample clock while doing an on-demand single-point acquisition, you get more sample clock pulses than data points. NI-DAQmx then throws away all points except the data point that corresponds to the first sample clock pulse. This ensures the data returned is always valid data. For acquisitions that use a Pause Trigger, the trigger could invalidate the samples in the pipeline if the trigger is asserted longer than the pipeline depth divided by the minimum sampling rate. For instance, if the device has a pipeline depth of three and a minimum sampling rate of 1000 samples per second, data should not sit in the pipeline for more than 3 ms. This gives up to a maximum of 3 ms for the Pause Trigger to remain asserted and three sample clocks to be detected before the data in the pipeline deteriorates past specifications. In the case of a Pause Trigger, NI-DAQmx does not detect or throw out any invalid samples. You must detect this situation and deal with any invalid samples as appropriate.

Analog Output

When using an external ao/SampleClock for finite generations, you need to provide one more sample clock pulse than the number of samples in the generation for the Wait Until Done function/VI to indicate the task is complete. For example, if you want to generate 1000 samples using an external sample clock, you need to provide 1001 sample clock pulses, or the Wait Until Done function/VI never indicates the task is done. All of the samples are generated, but the analog output timing engine needs one additional clock pulse to indicate the generation is complete. If you are trying to synchronize an analog output generation with another acquisition or generation by sharing a common clock, use the ao/SampleClock as the master clock, or key off of the generation or acquisition providing the master clock to determine when the generation is complete.

> ✎ **Note** For S Series devices that use the STC2 timing chip, such as the NI 6154, you do not need to apply an extra sample clock pulse to complete the task.

Timing Considerations for SC Express Devices

Related concepts:

- [Synchronizing X Series, M Series, and SC Express Devices](#)

Sampling Rate Considerations

NI 433x devices and the NI 4340 have both a maximum and a minimum sampling rate. Refer to the specifications for your device to determine the sampling rate range.

The NI 4309, NI 4353, and NI 4357 use multiplexed sampling with multiple ADCs. The maximum sampling rate of a NI 4309, NI 4353, or NI 4357 depends on how many channels you are acquiring, and which channels you are acquiring from. Setting the AI.ADCTimingMode and AI.ADCCustomTimingMode properties/attributes for the NI 4353 or NI 4357 also affects the maximum sampling rate.

Other Timing Considerations

NI 433x devices do not support external clocking from arbitrary external signal sources such as encoders and tachometers. You can program a NI 433x device to use an external timebase only when it is a slave in multi-device synchronized system. Refer to Synchronizing X Series, M Series, and SC Express Devices for more details.

The NI 4340 supports external clocking only in hardware-timed single-point sample mode.

Timing Considerations with Standalone NI CompactDAQ Systems

While Standalone NI CompactDAQ systems, such as the NI cDAQ-9132, 9133, 9134, 9135, 9136, and 9137 can be programmed with and run the LabVIEW Real-Time Module, the hardware and software architecture is optimized for data logging applications rather than for real-time control applications. This means that Standalone

NI CompactDAQ systems are ideal for streaming many channels to or from disk for extended periods of time, but should not be used for applications that require hard determinism. The Standalone NI CompactDAQ hardware and software architectures are not designed for deterministic operation.

> **Note**
>
> - Standalone NI CompactDAQ systems do not support hardware-timed single-point sample mode or hardware-timed non-buffered sample mode.
> - Standalone NI CompactDAQ systems do not support DAQ events as timing sources for time loops.

Sample Rate Considerations

The maximum sample rate depends on several factors, including the settling time for your device, the number of channels in the task, and whether your device uses simultaneous sampling (one ADC per channel) or multiplexed sampling (one ADC for all analog input channels). To find the maximum sample rate for your device, use the Sample Clock:Maximum Sample Rate attribute/property.

Sample Clock-Timed Pulse Train Generation

X Series and 661x devices support buffered, sample clock-timed pulse train generation. M Series, S Series, TIO, and E Series devices do not. Refer to your device documentation for additional information on supported counter functionality.

Device-Specific Sampling Methods
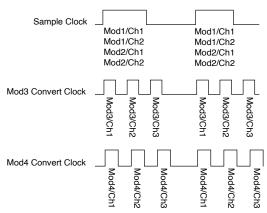
S Series and X Series simultaneous sampling devices use simultaneous sampling. M Series, E Series, all other X Series devices use multiplexed sampling.

C Series devices in CompactDAQ chassis[41], TestScale modules in TestScale chassis[42],

41. cDAQ-9171, 9174, 9178, 9179, 9181, 9184, 9185, 9188, 9188XT, 9189, and 9191

CompactDAQ controllers[43], CompactRIO controllers[44] and CompactRIO Single-Board controllers[45] use both simultaneous and multiplexed sampling, where all devices in the chassis share the same sample clock. Devices, such as the NI 9215, with an ADC for each analog channel use simultaneous sampling. Devices, such as the the NI 9205, with a single ADC sample in sequence use multiplexed sampling.

Each multiplexed C Series device has a separate convert clock. The convert clock timing is based on the number of channels for that device in the task, not the total number of channels in the task. You can set the convert clock rate on a per-device basis using the Active Devices and AI Convert Rate attributes/properties on the DAQmx Timing property node. The following figure depicts a ten-channel analog input task on two simultaneous sampling C Series devices and two multiplexed sampling C Series devices with different AI convert rates:

Related concepts:

- [Multiplexed Versus Simultaneous Sampling](#)

Timing Considerations for X Series Devices

X Series USB devices do not support hardware-timed single-point sample mode or Wait for Next Sample Clock.

---

42. TS-15000 and TS-15010

43. cDAQ-9132, 9133, 9134, 9135, 9136, and 9137

44. cRIO-9040, 9041, 9042, 9043, 9045, 9046, 9047, 9048, 9049, 9053, 9054, 9055, 9056, 9057, and 9058

45. sbRIO-9603, 9608, 9609, 9628, 9629, and 9638

Timestamps

Network-synchronized devices support four timestamp resources that allow for timestamping the first sample clock, start trigger, arm start trigger (counters only) or reference trigger.

The timestamp resources will be used to provide an accurate t0 for waveform reads with timed acquisitions that will correspond to the first sample clock for Analog Input and Digital Input tasks. Therefore, the first sample timestamp is enabled by default for these types of tasks.

The timestamp timescale can be configured using the Timestamp.Timescale attribute/property. Time triggers and timestamps can be specified in I/O Device Time or Host Time, depending on the needs of your application.

- I/O Device Time

  Shared by all network-synchronized devices on your 802.1AS subnet. I/O Device Time is most useful for synchronizing events across multiple chassis or correlating timestamps from multiple chassis, because even though it may be in an obscure time scale (for example, related to a point in the distant past, such as the Linux 1970 epoch), it removes other sources of skew related to Windows system time or other systems that are not network-synchronized to the same 802.1AS subnet. In that way, it provides best time trigger and timestamp accuracy but may reduce usability if it is not correlated to a recognizable global time. I/O Device Time also has the advantage of being monotonically increasing, so time triggers and timestamps spread across multiple devices or tasks accurately maintain their offsets from each other.

- Host Time

  The timescale your PC or NI Linux Real-Time controller uses. In cases where the NI Linux Real-Time controller is the Grand Master of your 802.1AS subnet, Host Time and I/O Device Time are the same. However, Host Time is typically synchronized to a local Real Time Clock or a Network Time Protocol server, and it is usually traceable to global time. Using Host Time is more intuitive because triggers and timestamps on the chassis are specified in times that are easily correlated to your local system time. However, this usability comes at the cost of reduced relative

accuracy between time triggers and timestamps that are spread across multiple devices or tasks, because using the calculated offset between the two timescales is not as accurate as using I/O Device Time directly. To help account for this loss of accuracy in a specific and common use-case, NI-DAQmx guarantees that two events that are scheduled for the same Host Time are guaranteed to start at the same I/O Device Time, preserving precise synchronization between chassis.

The number of available timestamp resources can be queried using the NumTimestampEngines attribute/property.

Network-synchronized devices include the following devices:

- cDAQ-9185, and 9189.
- FD-11601, FD-11603, FD-11605, FD-11613, FD-11614, FD-11634, and FD-11637.
- cRIO-9040, 9041, 9042, 9043, 9045, 9046, 9047, 9048, 9049, 9053, 9054, 9055, 9056, 9057, and 9058.
- sbRIO-9603, 9608, 9609, 9628, 9629, and 9638.

Related concepts:

- First Sample Timestamp
- Using Wait for Valid Timestamp VI
- Sync Lock Lost Detection
- Triggering
- Time-Based Features for Network-Synchronized Devices

Using Wait for Valid Timestamp VI

The Wait for Valid Timestamp VI waits until the selected timestamp is available and a value is returned. Possible timestamps are Start Trigger, Reference Trigger, Arm Start Trigger and First Sample.

First Sample Timestamp

The first sample timestamp corresponds to the time of the first sample clock pulse in a task.

The First Sample Timestamp attribute/property can only be enabled for hardware

timed tasks, and is enabled by default for Analog Input and Digital Input tasks. For Digital Input tasks as well as C Series Slow Sample and Delta-Sigma Devices, the first sample timestamp will be the t0 for waveform reads. For C Series Scanned Devices, the t0 for waveform reads will be the first sample timestamp plus an additional delay to more accurately reflect the time at which the sample was taken.

Related concepts:

- C Series Device Groupings

Configurable ADC Timing

On some devices, you can configure high-speed, high-resolution, or low-noise measurements using the AI.ADCTimingMode attribute/property. The following devices support configurable ADC Timing.

- FD-11613 and FD-11614
- NI 9207, NI 9208, NI 9212, NI 9213, NI 9217, and NI 9219
- NI 4353 and NI 4357

Related concepts:

- Configurable Timing for SC Express Devices
- Timing Considerations for C Series Devices

Configurable Timing for SC Express Devices

On the NI 4353 and NI 4357, you can configure high-speed or high-resolution measurements using the AI.ADCTimingMode attribute/property. You can also configure a custom value. To configure a custom value, do the following:

1. Set the AI.ADCTimingMode attribute/property to Custom.
2. Specify a custom value with the AI.ADCCustomTimingMode attribute/property. A smaller value is the lowest noise option while a larger value is the highest speed option. Refer to your device documentation for additional information on custom values.

Multiple Timing Engines

The following devices have multiple timing engines on some subsystems.

- NI 4302, 4303, 4304, 4305, 4339, 4340, 4463, 4464, 4480, and 4481
- NI 6533 and 6534
- cDAQ-9132, 9133, 9134, 9135, 9136, 9137, 9174, 9178, 9179, 9184, 9185, 9188, 9188XT, and 9189
- TS-15000 and TS-15010
- cRIO-9040, 9041, 9042, 9043, 9045, 9046, 9047, 9048, 9049, 9053, 9054, 9055, 9056, 9057, and 9058
- sbRIO-9603, 9608, 9609, 9628, 9629, and 9638

Related concepts:

- Timing Engines
- NI 4302, 4303, 4304, 4305 Timing Engines
- NI 4339, 4463, 4464 Timing Engines
- NI 4340 Timing Engines
- NI 4480, 4481 Timing Engines
- NI 6533, 6534 Timing Engines
- cDAQ-91xx and TestScale Chassis Timing Engines
- CompactRIO Timing Engines

NI 4302, 4303, 4304, 4305 Timing Engines

The 4302, 4303, 4304, and 4305 have four timing engines. Each of these timing engines can have its own configuration settings for timing, triggering, and the sample mode (either buffered or hardware timed single point).

Each timing engine can use any AI channel and can control up to eight channels. Each channel can only be used once across all timing engines.

By default, NI-DAQmx automatically selects an available timing engine when reserving the task. Use the DAQmx Timing attribute/property SampTimingEngine to specify the timing engine to use or to determine which timing engine NI-DAQmx automatically selected.

> **Note** You must reserve the task before querying the timing engine unless you explicitly specified the timing engine.

The SampTimingEngine attribute/property is an integer value corresponding to one of the four analog input timing engines available on the device:

SampTimingEngine Value	Timing Engine Used
0	te0
1	te1
2	te2
3	te3

If a task has more than eight channels, NI-DAQmx automatically uses multiple timing engines, synchronizing and sharing settings across all timing engines. All timing engines in a single task must share the same settings.

For example, if a task has 10 channels, NI-DAQmx uses two timing engines, which leaves two timing engines for other tasks. If the task has 32 channels, all four timing engines are used.

Each channel's digital filter settings can be configured independently if the channel is used in a task in buffered sample mode. For tasks using HWTSP sample mode, all channels in the task must have the same digital filter configuration.

Related concepts:

- Digital AI Filtering

NI 4339, 4463, 4464 Timing Engines

The NI 4339 and 4464 provide timing engines for analog input. The NI 4463 provides timing engines for analog output. Multiple timing engines allow those devices to run up to two tasks simultaneously, each using independent timing and triggering configurations.

By default, NI-DAQmx automatically selects an available timing engine when reserving the task. Use the DAQmx Timing attribute/property SampTimingEngine to specify the timing engine to use or to determine which timing engine NI-DAQmx automatically selected.

> **Note** You must reserve the task before querying the timing engine unless you explicitly specified the timing engine.

The SampTimingEngine attribute/property is an integer value corresponding to one of the two analog input timing or analog output engines available on the device:

SampTimingEngine Value	Timing Engine Used
0	te0
1	te1

On the NI 4339, 4463, and 4464, the Sample Clock and Sample Clock Timebase, as well as the Start and Reference triggers, exist on each timing engine. Therefore, the names of the output terminals for those signals include the associated timing engine. If NI-DAQmx automatically selects the timing engine for a task, the timing engine, thus the output terminals for those signals, are undefined until you reserve the task. To reference one of those terminals, such as to share a Start Trigger across multiple tasks, use the Terminal attribute/property associated with each signal to determine the terminal name.

> **Note** You must reserve the task before querying the terminal name unless you explicitly specify the timing engine.

> **Note** The NI 4339 only supports running a single buffered task and a single hardware-timed single-point task simultaneously.

Related concepts:

- Timing Engines
- Reserved State

- [Syntax for Terminal Names](#)
- [Terminals](#)

NI 4340 Timing Engines

The 4340 has four timing engines. Each of these timing engines can have its own configuration settings for timing, triggering, and the sample mode (either buffered or hardware timed single point).

Each timing engine can use any AI channel and can control up to four channels. Each channel can only be used once across all timing engines.

By default, NI-DAQmx automatically selects an available timing engine when reserving the task. Use the DAQmx Timing attribute/property SampTimingEngine to specify the timing engine to use or to determine which timing engine NI-DAQmx automatically selected.

> **Note** You must reserve the task before querying the timing engine unless you explicitly specified the timing engine.

The SampTimingEngine attribute/property is an integer value corresponding to one of the four analog input timing engines available on the device:

SampTimingEngine Value	Timing Engine Used
0	te0
1	te1
2	te2
3	te3

NI 4466 and 4467 Timing Engines

The NI 4466 and 4467 provide four timing engines, two for analog input and two for analog output. Multiple timing engines allow those devices to run up to two analog input and two analog output tasks simultaneously, each using independent timing

and triggering configurations.

By default, NI-DAQmx automatically selects an available timing engine when reserving the task. Use the DAQmx Timing attribute/property SampTimingEngine to specify the timing engine to use or to determine which timing engine NI-DAQmx automatically selected.

> **Note** You must reserve the task before querying the timing engine unless you explicitly specified the timing engine.

The SampTimingEngine attribute/property is an integer value corresponding to one of the four analog input timing or analog output engines available on the device:

SampTimingEngine Value	Timing Engine Used
0	te0
1	te1
2	te2
3	te3

The Sample Clock and Sample Clock Timebase, as well as the Start and Reference triggers, exist on each timing engine. Therefore, the names of the output terminals for those signals include the associated timing engine. If NI-DAQmx automatically selects the timing engine for a task, the timing engine, thus the output terminals for those signals, are undefined until you reserve the task. To reference one of those terminals, such as to share a Start Trigger across multiple tasks, use the Terminal attribute/property associated with each signal to determine the terminal name.

> **Note** You must reserve the task before querying the terminal name unless you explicitly specify the timing engine.

NI 4480, 4481 Timing Engines

The NI 4480 and 4481 provide timing engines for analog input. Multiple timing engines

allow those devices to run up to three tasks simultaneously, each using independent timing and triggering configurations.

By default, NI-DAQmx automatically selects an available timing engine when reserving the task. Use the DAQmx Timing attribute/property SampTimingEngine to specify the timing engine to use or to determine which timing engine NI-DAQmx automatically selected.

> ✏️ **Note** You must reserve the task before querying the timing engine unless you explicitly specified the timing engine.

The SampTimingEngine attribute/property is an integer value corresponding to one of the two analog input timing or analog output engines available on the device:

SampTimingEngine Value	Timing Engine Used
0	te0
1	te1
2	te2

On the NI 4480 and 4481, the Sample Clock and Sample Clock Timebase, as well as the Start and Reference triggers, exist on each timing engine. Therefore, the names of the output terminals for those signals include the associated timing engine. If NI-DAQmx automatically selects the timing engine for a task, the timing engine, thus the output terminals for those signals, are undefined until you reserve the task. To reference one of those terminals, such as to share a Start Trigger across multiple tasks, use the Terminal attribute/property associated with each signal to determine the terminal name.

> ✏️ **Note** You must reserve the task before querying the terminal name unless you explicitly specify the timing engine.

NI 6533, 6534 Timing Engines

NI 6533 and 6534 devices provide two timing engines for digital I/O. Multiple timing engines allow those devices to run up to two tasks simultaneously, each using independent timing and triggering configurations. Also, each timing engine uses different lines for handshaking and burst handshaking.

By default, NI-DAQmx assigns a timing engine based on the ports used in the task. Refer to device documentation to determine which ports use which timing engine. Use the DAQmx Timing property/attribute SampTimingEngine to specify the timing engine to use or to determine which timing engine NI-DAQmx automatically selected.

The SampTimingEngine property/attribute is an integer value corresponding to one of the two timing engines available on the device:

SampTimingEngine Value	Timing Engine Used
0	dig0
1	dig1

On NI 6533 and 6534 devices, the Sample Clock and Sample Clock Timebase, as well as the Start and Reference triggers, exist on each timing engine. Therefore, the names of the output terminals for those signals include the associated timing engine.

Related concepts:

- [Timing Engines](#)
- [Handshake Timing Defaults](#)
- [Burst Handshaking Timing Defaults for NI 653x Devices](#)
- [Syntax for Terminal Names](#)
- [Terminals](#)

cDAQ-91*xx* and TestScale Chassis Timing Engines

The cDAQ-91xx[46][1] or TestScale[47][2] chassis provide three timing engines for analog input. Multiple timing engines allow those chassis to run up to three analog input

46. cDAQ-9132, 9133, 9134, 9135, 9136, 9137, 9174, 9178, 9179, 9184, 9185, 9188, 9188XT, and 9189
47. TS-15000 and TS-15010

tasks simultaneously, each using independent timing and triggering configurations.

By default, NI-DAQmx automatically selects an available timing engine when reserving the task. Use the DAQmx Timing attribute/property SampTimingEngine to specify the timing engine to use or to determine which timing engine NI-DAQmx automatically selected.

> ✏️ **Note** You must reserve the task before querying the timing engine unless you explicitly specified the timing engine.

The SampTimingEngine attribute/property is an integer value corresponding to one of the three analog input timing engines available on the chassis:

SampTimingEngine Value	Timing Engine Used
0	te0
1	te1
2	ai

On cDAQ-91xx[1] or TestScale[2] chassis, the Sample Clock and Sample Clock Timebase, as well as the Start, Reference, and Pause triggers, exist on each timing engine. Therefore, the names of the output terminals for those signals include the associated timing engine. If NI-DAQmx automatically selects the timing engine for a task, the timing engine, thus the output terminals for those signals, are undefined until you reserve the task. To reference one of those terminals, such as to share a Start Trigger across multiple tasks, use the Terminal attribute/property associated with each signal to determine the terminal name.

> ✏️ **Note** You must reserve the task before querying the terminal name unless you explicitly specify the timing engine.

Counter Input Modules

The NI 9361 can also use te0 and te1.

Related concepts:

CompactRIO Timing Engines

CompactRIO[48] and CompactRIO Single-Board[49] controllers provide eight timing engines for input and a separate eight timing engines for output. The eight input timing engines are shared between analog input, digital input, and NI 9361 tasks. Multiple input timing engines allow those chassis to run up to eight analog input, digital input, or NI 9361 tasks simultaneously, each using independent timing and triggering configurations.

The eight output timing engines are shared between analog output and digital output. Multiple output timing engines allow those chassis to run up to eight analog output or digital output tasks simultaneously, each using independent timing and triggering configurations.

By default, NI-DAQmx automatically selects an available timing engine when reserving the task. Use the DAQmx Timing attribute/property SampTimingEngine to specify the timing engine to use or to determine which timing engine NI-DAQmx automatically selected.

> 🖉 **Note** You must reserve the task before querying the timing engine unless you explicitly specified the timing engine.

The SampTimingEngine attribute/property is an integer value corresponding to one of the eight timing engines available on the chassis:

SampTimingEngine Value	Analog Input, Digital Input, and NI 9361 Timing Engine Used	Analog Output and Digital Output Timing Engine Used
0	it0	ot0

48. cRIO-9040, 9041, 9042, 9043, 9045, 9046, 9047, 9048, 9049, 9053, 9054, 9055, 9056, 9057, and 9058.
49. sbRIO-9603, 9608, 9609, 9628, 9629, and 9638.

SampTimingEngine Value	Analog Input, Digital Input, and NI 9361 Timing Engine Used	Analog Output and Digital Output Timing Engine Used
1	it1	ot1
2	it2	ot2
3	it3	ot3
4	it4	ot4
5	it5	ot5
6	it6	ot6
7	it7	ot7

On the CompactRIO and CompactRIO Single-Board controllers, the Sample Clock and Sample Clock Timebase, as well as the Start, Reference (for analog input, digital input, and NI 9361 tasks only), and Pause triggers, exist on each timing engine. Therefore, the names of the output terminals for those signals include the associated timing engine. If NI-DAQmx automatically selects the timing engine for a task, the timing engine, thus the output terminals for those signals, are undefined until you reserve the task. To reference one of those terminals, such as to share a Start Trigger across multiple tasks, use the Terminal attribute/property associated with each signal to determine the terminal name.

Note You must reserve the task before querying the terminal name unless you explicitly specify the timing engine.

DSA

This section contains information specific to DSA devices.

Alias Rejection (DSA, C Series, and NI 433*x*)

DSA devices, C Series Delta-Sigma devices, and NI 433x devices employ a class of ADCs and DACs known as delta-sigma converters. Delta-sigma ADCs include built-in digital

filters to provide alias protection from out-of-band signal components. The digital filters always impart a delay of several samples between the time when a given analog voltage level becomes present at the ADC input and when the converter returns the corresponding digitized value. The length of this delay is always deterministic for a particular device running at a given sampling rate.

Likewise, interpolators and delta-sigma DACs provide digital filtering on analog output signals to eliminate out-of-band imaging and quantization noise. As with analog input, the digital output filtering results in a deterministic delay through the DAC.

You can safely ignore the effects of the digital filter delay for most input-only or output-only applications. The filter delay can become significant for applications requiring input and output synchronization such as stimulus-response testing and tight loop control. For DSA and C Series devices, if your application employs external digital triggering, the acquisition returns data that occurred in time before the trigger event. The number of samples preceding the trigger matches the ADC filter delay. Refer to your device documentation for more details on the ADC and DAC digital filter delays.

Alias Rejection at Low Sample Rates (DSA, C Series, and NI 4330/1)

At very low sample rates, the anti-aliasing filters for AI channels on DSA devices, C Series Delta-Sigma devices, NI 4330, and NI 4331 devices may not completely reject all out-of-band signals. This is primarily due to the internal digital filter of the delta-sigma ADC, which cannot suppress signals with frequencies near the multiples of the oversample rate (sampling rate multiplied by oversample factor). These devices also employ fixed cutoff analog lowpass anti-aliasing filters, but at low sample rates, some multiples of the oversample rate can fall below the cutoff frequency of the analog anti-aliasing filter.

For example, for a device using ADCs with an oversample factor of 128X and sampling at a rate of 1 kS/s, the oversample rate is 128 kHz. Some multiples of that oversample rate fall below the cutoff of the analog anti-aliasing filter. If the signal to be digitized contains energy near these frequencies, aliasing can result.

One way to prevent aliasing is to raise the sample rate so that the first 128X multiple of the sample rate falls above the cutoff of the analog anti-aliasing filter. For example, a sample rate of 25.6 kS/s is not subject to aliasing because the first 128X multiple (3.2 MHz) is well above the cutoff frequency of the analog anti-aliasing filter. Some DSA

devices support enhanced alias rejection, which automatically handles alias rejection at low sample rates. Refer to the device documentation for the specifics of your device.

Related concepts:

- [C Series Device Groupings](#)
- [Enhanced Alias Rejection](#)
- [Alias Rejection at Low Sample Rates (DSA, C Series, and NI 4330/1)](#)

Enhanced Alias Rejection

To avoid aliasing at low sample rates, some DSA devices support enhanced alias rejection. With enhanced alias rejection enabled, the device clocks the ADCs at a multiple of the user-specified sample rate. This results in an improvement in low-frequency alias rejection. With enhanced alias rejection enabled, you do not need to scale the desired sample rate, and you do not need to programmatically decimate data the device returns.

Enhanced alias rejection is controlled with the AI.EnhancedAliasRejectionEnable attribute/property. Enhanced alias rejection is enabled by default on NI 4461/4462 devices and disabled by default on NI 447x and NI 449x devices.

> **Note** The original versions of NI 447x devices do not support Enhanced Alias Rejection. Refer to *Dynamic Signal Acquisition Help* for more information.

Related concepts:

- [Alias Rejection at Low Sample Rates (DSA, C Series, and NI 4330/1)](#)

Synchronization Issues

When synchronizing multiple devices, set the AI.EnhancedAliasRejectionEnable attribute/property to the same value on all devices. If any of the synchronized devices do not support enhanced alias rejection, set AI.EnhancedAliasRejectionEnable to FALSE on all devices. When synchronizing devices from different categories, NI 4461/4462 with NI 447x for example, set AI.EnhancedAliasRejectionEnable to FALSE on all

devices.

Channel Order

On DSA devices, you must list channels in a task in ascending order. For example, if your task includes ai0 and ai1, you must arrange the channel list such that ai0 precedes ai1.

This constraint applies to virtual channels as well as physical channels. For example, if you include a virtual channel for ai0 named vibration and a virtual channel for ai1 named proxProbe, vibration must precede proxProbe in the channel list.

DSA, C Series, and the DAQmx I/O Server

DSA devices and C Series Delta-Sigma devices do not support the DAQmx I/O Server.

Related concepts:

- [C Series Delta-Sigma Devices](#)

Filter Delay Removal

NI 4302, 4303, 4304, 4305, 433x, 4340, 4464, 4466, 4467, 4480, 4481, and 449x devices support filter delay removal, which automatically discards filter delay samples. The AI.RemoveFilterDelay attribute/property controls the filter delay removal.

On NI 449x devices, the AI.RemoveFilterDelay attribute/property is disabled by default.

On NI 4302, 4303, 4304, 4305, 433x, 4340, 4464, 4466, 4467, 4480, and 4481 devices, the AI.RemoveFilterDelay attribute/property is enabled by default. On the NI 4330/4331, filter delay removal is always enabled.

On NI 4302, 4303, 4304, 4305, 4339, 4340, 4464, 4466, 4467, 4480, and 4481 devices, you can use the AI.FilterDelayAdjustment attribute/property to further adjust which samples to discard. This adjustment is relative to the amount of filter delay in the device, which is indicated by the AI.FilterDelay attribute/property.

If you disable the AI.RemoveFilterDelay attribute/property, you must compensate for filter delay samples. Refer to your device documentation for information on how many filter delay samples to compensate for on your device.

Filter Delay (DSA, C Series, and NI 433*x*)

The delta-sigma ADCs and DACs on DSA devices, C Series Delta-Sigma devices, and NI 433x devices employ digital filtering that imparts a delay of several sample intervals. The filter delays are equal in a homogeneous system, so these delays cancel out when performing phase measurements between channels. However, the filter delays differ in a heterogeneous system. This can introduce errors in phase comparisons between channels on different devices or between channels on similar devices running at different rates. Such phase errors are always deterministic, and you can account for them in software.

Related concepts:

- C Series Device Groupings

Gain for DSA Devices

On DSA devices, each gain setting corresponds to a particular range centered on 0 V. The gain settings are specified in decibels (dB), where the 0 dB reference is the default range of ±10 V.

For analog input operations, a negative gain value implies attenuation of the signal before the ADC, increasing the range beyond ±10 V. Thus, an input gain setting of –10 dB corresponds to an input range of ±31.6 V. On analog output, a negative gain value implies attenuation following the DAC. This decreases the output range. For instance, an output gain setting of –20 dB corresponds to an output range of ±1 V.

NI-DAQmx has three separate attribute/property sets you can use to control the hardware gain setting. Each has a different priority. If you write values to two or more of these attributes/properties that correspond to different hardware gain settings, the one with the highest priority will determine the hardware behavior.

- **Gain Attributes/Properties**—The gain attributes/properties AI.Gain and AO.Gain set the amount of gain to apply to the signal. These properties are set in decibels referenced to 10 V. These properties have the highest priority in NI-DAQmx.
- **Range Attributes/Properties**—The range attributes/properties AI.Rng.High, AI.Rng.Low, AO.DAC.Rng.High, and AO.DAC.Rng.Low define the maximum and minimum voltages you can acquire or generate. The range attributes/properties have a lower priority than the gain attributes/properties, but a higher priority than the maximum and minimum attributes/properties.
- **Maximum and Minimum Attributes/Properties**—The maximum and minimum attributes/properties AI.Max, AI.Min, AO.Max, and AO.Min specify values in engineering units that define the range. These attributes/properties have the lowest priority in NI-DAQmx. They are also the most commonly used since you can set them immediately when an NI-DAQmx task is created.

Hardware Data Compression (DSA and NI 433*x*)

If the raw data compression type is set to lossless packing and all channels in a task support hardware compression, hardware data compression is enabled by default. However, if any channels do not support hardware data compression, software data compression is selected by default.

Integrated Electronic Piezoelectric Excitation (IEPE)

If you attach an IEPE accelerometer or microphone to an AI channel that requires excitation from a device such as a DSA device, NI 9230, NI 9231, NI 9232, NI 9234 or an NI 9250[50], you must enable the IEPE excitation circuitry for that channel to generate the required current. IEPE signal conditioning can be independently configured on a per-channel basis.

To enable the IEPE current source on your device, use the Channel attribute/property AI.Excit.Val to specify a current in amperes. Some devices allow multiple excitation current values such as 0.004 A and 0.01 A. Other devices allow only a single value such as 0.004 A. A value of 0 A disables the IEPE excitation. Refer to the device documentation for details on your device.

50. Devices are listed as examples and are not intended to be a comprehensive list. Refer to the device documentation for the IEPE specifications of your device.

> **✏️ Note**
>
> - You cannot enable IEPE excitation on DSA devices when the terminal configuration is differential. The exception is NI 9218 which has optional IEPE excitation and supports a differential terminal configuration.
> - Changing the IEPE excitation level may cause transient voltages to appear in the signal. NI-DAQmx does not implement a delay to allow the signal to settle. Therefore, after changing the IEPE level and committing this change to the hardware with the Start function/VI or the Control Task function/VI, you might add a software delay to allow the signal to settle before proceeding with your application.

A DC offset is generated equal to the product of the excitation current and sensor impedance when IEPE signal conditioning is enabled. To remove the unwanted offset, you should enable AC coupling. Using DC coupling with IEPE excitation enabled is appropriate only if the offset does not exceed the voltage range of the channel.

Input Coupling

You can configure each AI channel of your DSA device to be either AC or DC coupled, with the exception of the NI 4495, the NI 4496, and the NI 4498 devices, which are AC coupled only. If you select DC coupling, any DC offset present in the source signal is passed to the ADC. The DC-coupled configuration is usually best if the signal source has only small amounts of offset voltage, less than ±100 mV, or if the DC content of the acquired signal is important.

If the source has a significant amount of unwanted DC offset (bias voltage), you should select AC coupling to take full advantage of the input dynamic range. Selecting AC coupling enables a single-pole, high-pass resistor-capacitor (RC) filter into the positive and negative signal path. Refer to your device documentation for additional information on the filter circuitry.

Use the NI-DAQmx Channel attribute/property AI.Coupling to set the input coupling mode on your DSA device.

If you create a virtual channel for acceleration or sound pressure measurements with your DSA device, the input coupling for the channel defaults to AC. For other types of virtual channels, the input coupling defaults to DC.

> ✎ **Note** NI-DAQmx does not compensate for the settling time or delay introduced by the RC filter.

Using AC coupling results in a drop in the low-frequency response of the AI circuitry. The AC coupling circuitry is usually characterized by a 3 dB cut-off frequency. However, the roll off from the high-pass filter can have a measurable effect even at frequencies several times greater than the 3 dB point.

Overload Detection

DSA devices support overload detection in both the analog domain (pre-digitization) and digital domain (post-digitization). An analog overrange can occur independently from a digital overrange, and vice versa. For example, an IEPE accelerometer might have a resonant frequency that, when stimulated, can produce an overrange in the analog signal. However, because the delta-sigma technology of the ADC uses very sharp anti-aliasing filters, the overrange is not passed into the digitized signal. Conversely, a sharp transient on the analog side might not overrange, but the step response of the delta-sigma anti-aliasing filters might result in clipping in the digital data.

Some DSA devices support both analog and digital overload detection, while others support only digital overload detection. Consult your device documentation for more information on the overload detection capabilities for your device.

Two NI-DAQmx Read attributes/properties allow you to check for overloaded channels. The first is OverloadedChansExist. This attribute/property returns a Boolean true if one or more channels experience an overload condition. The second is OverloadedChans. This attribute/property returns an array of strings indicating which channels (if any) experienced an overload condition. You must query the OverloadedChansExist attribute/property before querying the OverloadedChans attribute/property.

OverloadedChansExist reads the overload condition from the device and caches it in

the driver. It also resets the overload condition of the device after it is read. Subsequent reads of OverloadedChans attribute/property will read the overloaded channel information cached in the driver from the previous OverloadedChansExist query.

> **Note** NI-DAQmx returns all data whether or not an overload occurs. If your application requires overload checking, it is recommended that you read the overload attributes/properties after each call to Read. Your program should discard questionable data or return a flag when the driver reports an overload.

Simultaneous Tasks

The following devices can run multiple tasks simultaneously on some subsystems.

- NI 4302, 4303, 4304, 4305, 4339, 4340, 4463, 4464, 4466, and 4467
- NI 6533 and 6534
- cDAQ-9132, 9133, 9134, 9135, 9136, 9137, 9174, 9178, 9179, 9184, 9185, 9188, 9188XT, and 9189
- TS-15000 and TS-15010
- cRIO-9040, 9041, 9042, 9043, 9045, 9046, 9047, 9048, 9049, 9053, 9054, 9055, 9056, 9057, and 9058
- sbRIO-9603, 9608, 9609, 9628, 9629, and 9638

Related concepts:

- NI 4302, 4303, 4304, 4305 Simultaneous Tasks
- NI 4339, 4463, and 4464 Simultaneous Tasks
- NI 4480 and 4481 Simultaneous Tasks
- NI 6533/6534 Simultaneous Tasks
- CompactDAQ, CompactRIO, and TestScale Simultaneous Tasks

NI 4302, 4303, 4304, 4305 Simultaneous Tasks

The 4302, 4303, 4304, and 4305 can run up to four analog input tasks simultaneously,

each using independent timing and triggering configurations. Each task uses at least one of the four timing engines on the device.

Each timing engine can control up to eight channels. If a task has more than eight channels, NI-DAQmx automatically uses multiple timing engines for the task, synchronizing and sharing settings across all timing engines. For example, if a task has 10 channels, NI-DAQmx uses two timing engines for the task, which leaves two timing engines for other tasks. If the task has 32 channels, all four timing engines are used for one task.

Each bank of 8 ADCs (ai0:7, ai8:15, ai16:23, ai23:31) must all be in the same sample mode (buffered or hardware timed single point). For example, if ai0 is used in a task configured for hardware timed single point, any task using ai1:7 must also use the hardware timed single point sample mode.

Each channel's gain can be independently configured regardless of what task it's in or what sample mode it's used with.

Related concepts:

- [NI 4302, 4303, 4304, 4305 Timing Engines](#)

NI 4339, 4463, and 4464 Simultaneous Tasks

NI 4339, 4463, and 4464 devices can run up to two tasks simultaneously, each using independent timing and triggering configurations. Each task uses one of the two timing engines on the device.

The NI 4339 only supports running a single buffered task and a single hardware-timed single-point task simultaneously.

NI 4340 Simultaneous Tasks

NI 4340 devices can run up to four tasks simultaneously, each using independent timing and triggering configurations. Each task uses one of the four timing engines on the device.

NI 4480 and 4481 Simultaneous Tasks

NI 4480 and 4481 devices can run up to three tasks simultaneously, each using independent timing and triggering configurations. Each task uses one of the three timing engines on the device.

Each timing engine can be individually configured for frequency-domain mode (Sample Rate ≤ 1.25 MS/s) or time-domain mode (Sample Rate > 1.25 MS/s). For information on the differences between frequency-domain and time-domain mode refer to the **PXIe-4480/4481 User Manual**.

NI 6533/6534 Simultaneous Tasks

NI 6533 and 6534 devices can run up to two tasks simultaneously, each using independent timing and triggering configurations. Each task uses one of the two timing engines on the device.

Related concepts:

- NI 6533, 6534 Timing Engines

CompactDAQ, CompactRIO, and TestScale Simultaneous Tasks

To achieve maximum performance and synchronization, synchronize all analog input channels across modules or output channels across modules by combining them into a single task to create a multidevice task. This is only plausible if all channels in the task have the same timing needs.

The remainder of this topic provides information regarding how many tasks you can run simultaneously at different rates on the following controllers:

- Gen II CompactDAQ chassis:
    - cDAQ-9132, 9133, 9134, 9135, 9136, and 9137
    - cDAQ-9171, 9174, 9178, 9179
    - cDAQ-9181, 9184, 9185, 9188, 9188XT, and 9189
    - cDAQ-9191

- cRIO-9040, 9041, 9042, 9043, 9045, 9046, 9047, 9048, 9049, 9053, 9054, 9055, 9056, 9057, and 9058
- sbRIO-9603, 9608, 9609, 9628, 9629, and 9638
- TS-15000 and TS-15010

Because hardware timed tasks typically use more onboard resources than software timed tasks, the total number of possible concurrent tasks typically depends on whether you are running tasks that use hardware timing or software (on demand) timing.

Related concepts:

- C Series Multidevice Tasks
- cDAQ-91xx and TestScale Chassis Timing Engines
- CompactRIO Timing Engines

Hardware-Timed Tasks

Hardware-timed tasks require timing information from the System Timing Controller to use for sample clocks, reference clocks, triggers, and so on.

The System Timing Controller for the Gen II chassis has the following timing engines built in:

- Three AI timing engines
- One AO timing engine
- One DI timing engine
- One DO timing engine
- Four general purpose counters

The System Timing Controller for the CompactRIO controller has the following timing engines built in:

- Eight input timing engines shared between AI and DI
- Eight output timing engines shared between AO and DO
- Four general purpose counters

The following table illustrates the number of hardware timed tasks available to the

user.

Hardware-Timed Tasks	CompactDAQ and TestScale: Number of Tasks per Chassis	CompactRIO/Single-Board RIO: Number of Tasks per Controller	CompactDAQ, CompactRIO, Single-Board RIO, TestScale: Number of Tasks per Module
Analog Input	3^{51}	$8^{52[2]}$	1
Analog Output	1	$8^{[2]}$	1
Counter Input/Output	4, $2^{53[3]}$	4, $8^{[2]54[4]}$	4, $1^{55[5]}$
Digital Input	1	$8^{[2]}$	1
Digital Output	1	$8^{[2]}$	1

Streaming Limitations on CompactDAQ Gen II Chassis

On CompactDAQ and TestScale chassis, there is also a streaming buffer for hardware timed tasks that puts a limit on the total number of hardware timed tasks that can run simultaneously. The NI cDAQ-9171/9181/9191 support six data streams and the remaining models support seven data streams. These independent, high-speed data streams allow for up to six or seven simultaneous hardware-timed tasks, such as analog input, analog output, buffered counter/timers, hardware-timed digital input/output, or CAN communication.

51. The number of analog input tasks for Dynamic Signal Analyzer (DSA) devices in Gen II NI CompactDAQ chassis is limited to two because the chassis support a maximum of two synchronization pulse signals, which is required when using devices with different oversample clock timebases, such as DSA.
52. On Single-Board RIO controllers, you can have one task at a time per slot or onboard IO module. The number of slots and onboard IO modules varies by sbRIO device. See the specifications for your device.
53. On the Gen II NI CompactDAQ chassis, there is a limit of two NI 9361 tasks per chassis.
54. On CompactRIO controllers, there is a limit of eight NI 9361 tasks per chassis. On Single-Board RIO Controllers, there is a limit of one NI 9361 task per slot. You can add counter channels from one or more NI 9361s to a single task. Using the NI 9361s and the 4 general purpose counters, there can be a total of 12 tasks.
55. One NI 9361 only supports a single task. Therefore, multiple counters from a NI 9361 must be used in the same task. Other parallel digital modules allow creating separate tasks for each one of the four onboard counters from the chassis.

> **Note** CAN communication always takes up two data streams, but does not use any timing engines.

On CompactDAQ and TestScale chassis, the data streams are comprised by an 8 KB block of memory that is divided up into six or seven First In First Out (FIFO) data buffers. These data buffers vary in size, and the largest data buffers are assigned to the first tasks that get reserved. Therefore, to get the best streaming performance, make sure to reserve your highest bandwidth tasks first. Your first two tasks will reserve 2048 bytes each, the third, fourth and fifth tasks will reserve 1024 bytes each and the sixth and seventh tasks will reserve 512 bytes each.

Software-Timed Tasks

Most software-timed tasks do not require a signal from the System Timing Controller to run. Software timed means the host computer is controlling how often a sample is read from or written to the C Series module.

Analog Input tasks will still use one of the AI timing engines, so the limit for AI tasks is always the same as hardware times tasks. Each counter input task using an NI 9361 will also use an AI timing engine. This is not the case for AO, DI, or DO. Subsequently, the software timed task limits for these types of acquisitions depends on the number of channels you have available to use. See following table for a summary.

Software-Timed (On Demand) Tasks	CompactDAQ and TestScale: Number of Tasks per Chassis	CompactRIO, Single-Board RIO: Number of Tasks per Controller	CompactDAQ, CompactRIO, Single-Board RIO, TestScale: Number of Tasks per Module
Analog Input	$3^{56[6]}$	$8^{[][6]}$	1
Analog Output	Number of total AO channels in chassis (up to 128 total tasks).	Number of total AO channels in controller (up to 128 total tasks).	1 Task for each channel (up to 16 tasks per module).
Counter Input/ Output	4, 2[]	4, 8[][][]	4, 1[]

56. Only Slow-Sampled modules, Multiplexed modules, and Simultaneous SAR modules support software-timed analog input tasks.

Software-Timed (On Demand) Tasks	CompactDAQ and TestScale: Number of Tasks per Chassis	CompactRIO, Single-Board RIO: Number of Tasks per Controller	CompactDAQ, CompactRIO, Single-Board RIO, TestScale: Number of Tasks per Module
Digital Input	Dependent on modules. Typically, at least 2 static DI per slot.	Dependent on modules. Typically, at least 2 static DI per slot.	1 task per port.[57][7]
Digital Output	Dependent on modules. Typically, at least 2 static DO per slot.	Dependent on modules. Typically, at least 2 static DO per slot.	1 task per port.[7]

The number of counter tasks is always limited by the fact that you have four counters. Depending on whether you are using a counter task that uses two counters (like measuring frequency using the two counter methods), you may be restricted to only two counter tasks. But typically, you can have all fout counters running simultaneously. To determine if you need one or two counters for your counter task, see the KnowledgeBase topic KB 4L0A62E9.

Related concepts:

- C Series Device Groupings
- Digital I/O Considerations for C Series and TestScale Devices
- Digital I/O Considerations for C Series Devices

Related information:

- Controller Area Network (CAN) Overview
- KB 4L0A62E9

Multidevice Tasks

Tasks can contain channels from multiple devices for these devices:

- C Series

57. The NI 9401 and TS-15050 DIO P0 are 1 task per nibble.

- FieldDAQ
- S Series
- DSA, SC Express, and X Series

Related concepts:

- C Series Multidevice Tasks
- C Series Multichassis Device Tasks
- FieldDAQ Multidevice Tasks
- S Series Multidevice Tasks
- DSA, SC Express, and X Series Multidevice Tasks

C Series Multidevice Tasks

When you include channels from multiple C Series modules in a task, NI-DAQmx automatically synchronizes the modules. A task can include channels from multiple C Series modules, given the following conditions:

- All channels in the task must be of the same I/O type. Multiple counter I/O channels are only allowed for *Devices That Support Multi-Counter Tasks*.
- If the task includes channels from a mixture of C Series Delta-Sigma devices, you must account for filter delay differences between the devices. This delay is also known as the input delay. Refer to your device specifications for the value.
- The modules must all be in the same NI CompactDAQ chassis or meet the conditions for multichassis device tasks.

> **Note**
>
> - AI tasks containing only 16-bit or lower resolution AI modules use half the USB bandwidth of tasks with 24-bit AI modules.
> - The format of raw data returned by a C Series AI task varies depending on if any 24-bit AI modules are in the task and might not correspond to the order of the channels in the task. Scaled or unscaled data is preferable to raw data with the NI CompactDAQ chassis.
> - The acquisition mode of the NI 9775 will vary depending on the other

> modules in the task. If the task uses only NI 9775 modules, all the modules can operate in Record Mode. If the task uses an NI 9775 module and other types of modules, the NI 9775 operates in Continuous Mode.

Related concepts:

- C Series Device Groupings
- Filter Delay (DSA, C Series, and NI 433x)
- Devices That Support Multi-Counter Tasks

C Series Multichassis Device Tasks

A task can include channels from multiple C Series modules in separate CompactDAQ chassis[58], CompactDAQ controllers[59], CompactRIO controllers[60] and CompactRIO Single-Board controllers[61], given the following conditions:

- Network-synchronized devices [62] must be part of the same time-synchronized network. Refer to the device specifications for supported synchronization protocols and topologies.
- Chassis that are not network-synchronized must be connected through a NI 9469.
- If a task has a network-synchronized chassis and connections through the NI 9469 to chassis that are not synchronized, the first channel in the task must be in the network-synchronized chassis along with the NI 9469 connection to other chassis that are not synchronized.
- When using signal-based synchronization using the NI 9469, the first channel in the channel list must be on a module in the master chassis. The master chassis is determined by the physical configuration of the chassis connections, and it should be able to output signals to the slave chassis.

58. cDAQ-9171, 9174, 9178, 9179, 9181, 9184, 9185, 9188, 9188XT, 9189, and 9191.
59. cDAQ-9132, 9133, 9134, 9135, 9136, and 9137.
60. cRIO-9040, 9041, 9042, 9043, 9045, 9046, 9047, 9048, 9049, 9053, 9054, 9055, 9056, 9057, and 9058.
61. sbRIO-9603, 9608, 9609, 9628, 9629, and 9638.
62. Network-synchronized devices include the cDAQ-9185, 9189; FD-11601, FD-11603, FD-11605, FD-11613, FD-11614, FD-11634, FD-11637; cRIO-9040, 9041, 9042, 9043, 9045, 9046, 9047, 9048, 9049, 9053, 9054, 9055, 9056, 9057, 9058; and sbRIO-9603, 9608, 9609, 9628, 9629, and 9638.

- If an Analog Input task contains Delta-Sigma modules, a channel on a Delta-Sigma module in the master chassis needs to be first on the channel list.
- If an Analog Input task contains a device with a reference clock, a channel from a reference-clocked device needs to be first on the channel list.
- No more than one CompactRIO controller may participate in a multichassis device task.

Exceptions

- The NI 9361 does not support multichassis device tasks.
- The NI 9260 must drive the idle output to zero. If you don't, you will receive an error.
- The NI 9775 does not support multichassis device tasks.

Related concepts:

- Time-Based Features for Network-Synchronized Devices

DSA, SC Express, and X Series Multidevice Tasks

A task can include channels from multiple DSA, SC Express, and X Series devices, given the following conditions. When you include channels from multiple devices in a task, NI-DAQmx automatically synchronizes the devices and enables trigger skew correction for the devices. All channels in a task must be of the same I/O type, such as analog input or analog output.

Note For additional device-specific DSA synchronization information, refer to the **NI Dynamic Signal Acquisition User Manual**, which you can find at ni.com/manuals.

Related concepts:

- Filter Delay (DSA, C Series, and NI 433x)
- Trigger Skew Correction

Analog Input

- USB X Series Devices

  You cannot use USB X Series devices in multidevice tasks.

- PCIe X Series Devices

  You must use a RTSI cable to connect the devices, and you must identify the cable in MAX.

- PXI/PXIe Devices
  - The devices must be in a single chassis, and you must identify the chassis in MAX.
  - NI-DAQmx accounts for filter delay differences between devices with some qualifications:
    - NI 4461 and 4462 devices lack the ability to compensate for filter delay.
    - For all other devices, if the task includes channels from different device families, NI-DAQmx will account for filter delay differences between the devices.

Use the following table to check the multidevice task compatibility of any one device to another device. A checkmark in a cell indicates that the devices in the corresponding row and column can be used together in a multidevice task. For groups of more than two devices, every device must be compatible with every other device. An example of a group of devices that can be used in a multidevice task is the NI 4300, 4353, 4357 and PXIe X Series (63xx) as all of these devices are compatible with every other device listed in the group.

Table 5. PXI/PXIe Multidevice Task Compatibility

	4300	4302/ 4303/ 4304/ 4305	4309	4310	4330/ 4331	4339	4340	4353	4357	4461/ 4462	4464	4466/ 4467	4480/ 4481	449x	6386/ 6396	Other X Series (63xx)
4300	√	√	√	√	-	√	-	√	√	-	√	√	√	-	√	√
4302/ 4303/ 4304/ 4305	√	√	-	√	√	√	√	√	-	√	√	√	√	-	-	√

Device																
4309	✓	-	✓	-	-	-	-	-	-	-	-	-	-	-	✓	✓
4310	✓	✓	-	✓	-	-	✓	✓	-	-	✓	✓	✓	-	✓	✓
4330/4331	-	✓	-	-	✓	✓	✓	-	-	-	✓	✓	✓	✓	-	-
4339	✓	✓	-	✓	✓	✓	✓	-	-	✓	✓	✓	✓	-	✓	✓
4340	-	✓	-	-	✓	✓	✓	-	-	✓	✓	✓	✓	-	✓	✓
4353	✓	✓	-	✓	-	-	-	✓	✓	-	-	-	-	-	-	✓
4357	✓	-	-	-	-	-	-	✓	✓	-	-	-	-	-	-	✓
4461/4462	-	✓	-	-	-	✓	✓	-	-	✓	✓	✓	-	✓	-	-
4464	✓	✓	-	✓	✓	✓	✓	-	-	✓	✓	✓	✓	-	✓	✓
4466/4467	✓	✓		✓	✓	✓	✓			✓	✓	✓	✓		✓	✓
4480/4481	✓	✓	-	✓	✓	✓	✓	-	-	-	✓	✓	✓	-	✓	✓
449x	-	-	-	-	✓	-	-	-	-	✓	-	-	-	✓	-	-
6386/6396	✓	-	✓	✓	-	✓	✓	-	-	-	✓	✓	✓	-	✓	✓
Other X Series (63xx)	✓	✓	✓	✓	-	✓	✓	✓	✓	-	✓	✓	✓	-	✓	✓

Analog Output

- PXIe-4463, 4466, and 4467 devices

  The devices must be in a single chassis and the chassis must be identified in MAX. All devices in the task must be PXIe-4463s.

- PXIe X Series, NI 4322, and NI 6738/6739 Devices

  The devices must be in a single chassis and the chassis must be identified in MAX.

- PCIe X Series Devices

  You must use a RTSI cable to connect the devices, and you must identify the cable in MAX.

- USB X Series Devices

You cannot use USB X Series devices in multidevice tasks.

FieldDAQ Multidevice Tasks

Multiple FieldDAQ devices can be added to the same task, and they will be automatically synchronized. There are no restrictions on which FieldDAQ devices can be synchronized together.

S Series Multidevice Tasks

A task can include channels from multiple S Series devices, given the following conditions. When you include channels from multiple S Series devices in a task, NI-DAQmx automatically synchronizes the devices.

- All Devices

  All channels in the task must be analog input channels.

  When you include channels from different S Series devices and use an external clock setup, you must import the external clock into the device with the longest pipeline of all the devices in the task. Failing to do so results in an incomplete acquisition, with the device importing the clock not receiving enough sample clock pulses.

- PXI Devices

  The devices must all be in a single chassis, and you must identify the chassis in MAX.

- PCI Devices

  You must use a RTSI cable to connect the devices, and you must identify the cable in MAX.

Related concepts:

- [Timing Considerations for S Series](#)

Bridge Measurement Type Support

The following devices support the force, pressure, torque, and bridge (V/V) measurement types for taking measurements from bridge-based sensors:

- NI FD-11637
- PXIe-433x
- NI 9218
- NI 9219
- NI 9235
- NI 9236
- NI 9237

C Series Device Groupings

C Series devices can be grouped into scanned devices, slow sample devices, and delta-sigma devices based on acquisition behavior.

> **Note** This list does not include all C Series devices.

C Series Scanned Devices

- NI 9201
- NI 9203
- NI 9205
- NI 9206
- NI 9207
- NI 9208
- NI 9209
- NI 9210
- NI 9211
- NI 9213
- NI 9214
- NI 9216
- NI 9217
- NI 9221

- NI 9226

C Series Slow Sample Devices

- NI 9207
- NI 9208
- NI 9209
- NI 9210
- NI 9211
- NI 9212
- NI 9213
- NI 9214
- NI 9216
- NI 9217
- NI 9219
- NI 9224
- NI 9226
- NI 9228

C Series Delta-Sigma Devices

- NI 9202
- NI 9218
- NI 9225
- NI 9227
- NI 9229
- NI 9230
- NI 9231
- NI 9232
- NI 9234
- NI 9235
- NI 9236
- NI 9237
- NI 9238
- NI 9239
- NI 9242
- NI 9244

- NI 9246
- NI 9247
- NI 9250
- NI 9251
- NI 9260

C Series Reference Clocked Devices

- NI 9775

C Series Simultaneous SAR Devices

- NI 9215
- NI 9220
- NI 9222
- NI 9223

Common-Mode Over-Range Detection
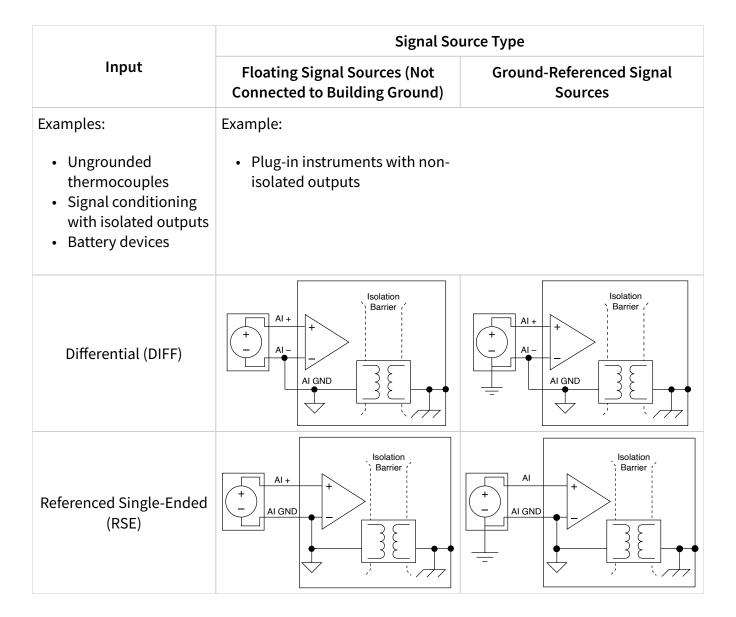
All input channels share a common ground, COM, that is isolated from other modules in the system. The device's common-mode range is the maximum voltage between any channel and COM. The NI 9213, NI 9214, and PXIe 4353 measures the common-mode voltage level of each channel, and its over-range status can be monitored in NI-DAQmx.

To determine if a common-mode over-range detection has occurred, use the Common Mode Range Error Channels Exist and Common Mode Range Error Channels properties within the DAQmx Read property node. Common Mode Range Error Channels Exist returns a Boolean of true if one or more channels exceed the common-mode input range since the last time the property was queried, and Common Mode Range Error Channels returns the names of the virtual channels that exceed the common-mode input range.

If a common-mode voltage out of range is detected, the accuracy of the data on any channel in the task may be impacted. If a thermocouple is connected to the device, but is not in the task, make sure the channel does not exceed the common-mode voltage range. A floating thermocouple, or a channel that is left unconnected, will not exceed

the common-mode voltage range. Refer to the devices Specifications document for more information about the common-mode voltage range.

Connecting Analog Voltage Input Signals for Isolated Devices

Input	Signal Source Type	
	Floating Signal Sources (Not Connected to Building Ground)	Ground-Referenced Signal Sources
Examples: • Ungrounded thermocouples • Signal conditioning with isolated outputs • Battery devices	Example: • Plug-in instruments with non-isolated outputs	
Differential (DIFF)		
Referenced Single-Ended (RSE)		

Related concepts:

- [Terminal Configurations (Analog Input Ground Reference Settings) for Isolated Devices](#)

CompactRIO Considerations

This section contains information on using the following CompactRIO and CompactRIO single-board controllers with NI-DAQmx:

- cRIO-9040, 9041, 9042, 9043, 9045, 9046, 9047, 9048, 9049, 9053, 9054, 9055, 9056, 9057, and 9058 controllers.
- sbRIO-9603, 9608, 9609, 9628, 9629, and 9638 controllers.

Related concepts:

- [CompactRIO Timing Engines](#)
- [CompactDAQ, CompactRIO, and TestScale Simultaneous Tasks](#)
- [Time-Based Features for Network-Synchronized Devices](#)

Slot Program Mode

In MAX, when you add a C Series module to a CompactRIO controller there are three program modes to choose from.

- Real-Time (NI-DAQmx)
- Real-Time Scan
- LabVIEW FPGA

In order for DAQmx to communicate with the module, the module must be in **Real-Time (NI-DAQmx)** mode.

> ✎ **Note** Not all modules support Real-Time (NI-DAQmx).

Related information:

- [Software Support for CompactRIO and CompactDAQ](#)

Hardware Timed Single Point Sample Mode

The CompactRIO controllers support hardware timed single point (HWTSP) sample mode with a few caveats.

- The NI 9260 does not support HWTSP.
- The NI 9361 does not support HWTSP.
- **C Series Scanned Devices** do not default to the lowest latency mode. They default to a slower convert rate to allow more time for settling. This behavior limits the maximum HWTSP acquisition rate of the module to allow more time for settling. If you prefer, you can configure the module for a faster acquisition rate with less time for settling. For additional information, see **Sampling Rate Considerations**.

Related concepts:

- Sampling Rate Considerations
- C Series Scanned Devices

Model Names

CompactRIO may use slightly different C Series model names and product IDs compared to CompactDAQ. For CompactRIO, see **C Series Module IDs** on ni.com.

Related information:

- C Series Module IDs

Timing Considerations

CompactRIO controllers handle sampling rate and the default value for the hardware-timed mode differently than CompactDAQ chassis.

In a CompactRIO or Single-Board RIO controller with C Series Slow Sample devices, such as the NI 9211, if the sampling rate of a hardware-timed acquisition exceeds the maximum sampling rate of the module, DAQmx generates warning or errors. When a Slow Sample device is in the same task as a non-Slow Sample device, exceeding the maximum sampling rate of the Slow Sample device results in the most recently acquired sample being read multiple times. In this scenario, the first sample of a hardware-timed acquisition with C Series Slow Sample devices is sampled when the task is committed.

Default Settings for the AI.ADCTimingMode Attribute/Property

For all modules in the CompactRIO and Single-Board RIO controllers, the default value in hardware-timed mode is automatically determined based on Sample Clock Rate.

Related concepts:

- Sampling Rate Considerations
- Configurable Timing for C Series Devices
- CompactRIO Timing Engines
- CompactDAQ, CompactRIO, and TestScale Simultaneous Tasks
- C Series Device Groupings
- CompactDAQ and CompactRIO Simultaneous Tasks

Shared Trigger Bus

On the CompactRIO and CompactRIO Single-Board controllers, NI-DAQmx provides a simple trigger bus between LabVIEW FPGA and NI-DAQmx with the following characteristics:

CompactRIO[63]	CompactRIO Single-Board[64]

- Four fixed direction LabVIEW FPGA to NI-DAQmx Lines:
    ◦ cRIO_Trig0, cRIO_Trig1, cRIO_Trig2, and cRIO_Trig3 are terminals that are driven from LabVIEW FPGA to NI-DAQmx.
    ◦ LabVIEW FPGA: output boolean chassis I/O
    ◦ NI-DAQmx: common-visibility source terminals
        ▪ Can be used with *Immediate Routing* and *Task-Based Routing*
        ▪ Does not prescribe to *Lazy Line Transitions* rules
	- Two fixed direction LabVIEW FPGA to NI-DAQmx Lines:
	    ◦ cRIO_Trig0 and cRIO_Trig1 are terminals that are driven from LabVIEW FPGA to NI-DAQmx.
	    ◦ LabVIEW FPGA: output boolean chassis I/O
	    ◦ NI-DAQmx: common-visibility source terminals
	        ▪ Can be used with *Immediate Routing* and *Task-Based Routing*
	        ▪ Does not prescribe to *Lazy Line Transitions* rules

63. cRIO-9040, 9041, 9042, 9043, 9045, 9046, 9047, 9048, 9049, 9053, 9054, 9055, 9056, 9057, and 9058
64. sbRIO-9603, 9608, 9609, 9628, 9629, and 9638

CompactRIO	CompactRIO Single-Board

• Four Fixed Direction NI-DAQmx to LabVIEW FPGA Lines:
  ◦ cRIO_Trig4, cRIO_Trig5, cRIO_Trig6, and cRIO_Trig7 are terminals that are driven from NI-DAQmx to LabVIEW FPGA.
  ◦ LabVIEW FPGA: input boolean chassis I/O
    ▪ Minimum pulse width requirements:
      ▪ 12.5 ns for most destination terminals
      ▪ 150 ns for SyncPulse terminals
  ◦ NI-DAQmx: common-visibility destination terminals
    ▪ Can be used with ***Immediate Routing*** and ***Task-Based Routing***
    ▪ Does not prescribe to ***Lazy Line Transitions*** rules
	• Two Fixed Direction NI-DAQmx to LabVIEW FPGA Lines:
	  ◦ cRIO_Trig2 and cRIO_Trig3 are terminals that are driven from NI-DAQmx to LabVIEW FPGA.
	  ◦ LabVIEW FPGA: input boolean chassis I/O
	    ▪ Minimum pulse width requirements:
	      ▪ 12.5 ns for most destination terminals
	      ▪ 150 ns for SyncPulse terminals
	  ◦ NI-DAQmx: common-visibility destination terminals
	    ▪ Can be used with ***Immediate Routing*** and ***Task-Based Routing***
	    ▪ Does not prescribe to ***Lazy Line Transitions*** rules

Note The signals available for export from DAQmx vary in pulse width. On a cRIO controller, if you route any of the following signals over the cRIO_Trig bus to the FPGA Target, the pulse is too short to be seen in the default top level clock domain of 40 MHz:

- Change detection event exported from a buffered change detection DI task
- Sample clock exported from a hardware-timed digital input or output task running faster than 3.5 MHz

- Counter output event exported from a counter task that configures its output behavior to pulse

To fix this problem, consider routing the signal to a counter to widen the pulse.

Related concepts:

- [Immediate Routing](#)
- [Task-Based Routing](#)
- [Lazy Line Transitions](#)

Devices That Support Multi-Counter Tasks

The following devices have multiple counters and support adding multiple counter channels to the same task.

- NI 9361

Digital AI Filtering

The NI 4302, 4303, 4304, 4305, 4339, 4480, and 4481 modules have a user-configurable lowpass filter for their analog input channels. This filtering is implemented with a digital elliptical filter. It is configured independently of the tracking anti-aliasing filter. It may be helpful when measuring in an environment with excessive high frequency noise below the anti-aliasing filter's cutoff.

When sampling in hardware-timed single point mode, the filter is always enabled. You can set or query its cutoff frequency using the AI.DigFltr.Lowpass.CutoffFreq attribute/property. If your application does not require this additional filtering, leave this property set to its default value. This will configure the widest bandwidth filter possible.

Digital AI filtering for hardware-timed single point mode is supported on the following modules:

Modules	Default Cutoff Frequency	Support Cutoff Frequencies
PXIe-4302, 4303, 4304, 4305	2 kHz	100 Hz, 200 Hz, 500 Hz, 1 kHz, 2 kHz, 3 kHz
PXIe-4339	2 kHz	200 Hz, 500 Hz, 1 kHz, 2 kHz , 3 kHz

When using other sampling modes, the filter is optional. This filter can be enabled by using the AI.DigFltr.Enable attribute/property, and you can set or query its cutoff frequency using the AI.DigFltr.Lowpass.CutoffFreq attribute/property.

The following modules support digital AI filtering in other sampling modes:

Modules	Default Cutoff Frequency	Support Cutoff Frequencies
PXIe-4302, 4303, 4304, 4305	None	2 Hz, 20 Hz, 200 Hz, 1 kHz, 2 kHz, none
PXIe-4480, 4481	2 MHz	2 MHz, user-defined

> **Note** User-defined cutoff frequencies for the 4480 and 4481 are supported only for sample rates greater than 1.25 MHz.

Excitation Fault Detection

Two NI-DAQmx Read attributes/properties allow you to check for excitation faults. The first is ExcitFaultChansExist. This attribute/property returns a Boolean of true if one or more channels experience an excitation fault condition. The second is ExcitFaultChans. This attribute/property returns an array of strings indicating which channels (if any) experienced an excitation fault condition. You must query the ExcitFaultChansExist attribute/property before querying the ExcitFaultChans attribute/property.

ExcitFaultChansExist reads the excitation fault condition from the device and stores it in NI-DAQmx. Subsequent reads of ExcitFaultChans will read the excitation fault channel information stored in NI-DAQmx from the previous ExcitFaultChansExist query.

NI-DAQmx returns all data regardless of whether an excitation fault happens. If your application needs excitation fault checking, you should read the excitation fault attributes/properties after each call to read. Your program should discard questionable data or return a flag when NI-DAQmx reports an excitation fault.

The NI 9218 can detect if there is an overcurrent fault between EX+ and EX- pins of either channel for either the strain mode case or the powered sensor measurement

case. When the NI 9218 is configured for powered sensor measurement, which requires a +12 V excitation, it can also detect the absence of the required 9-30 V external power supply connected to the Vsup terminal. Both of these types of faults are reported as excitation faults.

The PXIe-4340 can detect if there is an overcurrent fault between the EX+ and EX- pins of any channel. This fault occurs when a voltage source is connected to the EX+ and EX- pins of any channel, or the load impedance is too low (excessive current). Ensure that only the primary winding of an LVDT/RVDT sensor is connected to the EX+ and EX- pins of any channel on this module.

External Overvoltage Detection

An external overvoltage condition occurs when an analog output channel is connected to an external voltage source that is high enough to overdrive the analog output channel. Two NI-DAQmx Write attributes/properties allow you to check for external overvoltage channels.

The first is ExternalOvervoltageChansExist. This attribute/property returns a Boolean true if one or more channels experience an external overvoltage condition. The second is ExternalOvervoltageChans. This attribute/property returns an array of strings indicating which channels (if any) experienced an external overvoltage condition. You must query the ExternalOvervoltageChansExist attribute/property before querying the ExternalOvervoltageChans attribute/property.

ExternalOvervoltageChansExist reads the external overvoltage fault condition from the device and caches it in the driver. It also resets this condition of the device after it is read. Subsequent reads of ExternalOvervoltageChans attribute/property will read the faulted channel information cached in the driver from the previous ExternalOvervoltageChansExist query.

> **Note** NI-DAQmx generates all data whether or not an external overvoltage fault occurs. If your application requires external overvoltage checking, it is recommended that you read the above attributes/properties after each call to Write. Your program should discard questionable output or return a flag when the driver reports an external overvoltage.

Consult your device documentation for more information on fault reporting thresholds and capabilities specific to your device.

External Reference Sources

For 625x M Series devices, you can use APFI 0 or APFI 1 for analog output external reference sources. With the NI ELVIS II Family devices, APFI 0 and APFI 1 are not available as analog output external reference sources. For 628x M Series devices, you can use ao0 through ao3. On some other STC-based devices, you can use EXTREF as the analog output external reference source. Using an external voltage reference enables you to maximize the resolution of your device. If the voltages you want to generate do not exceed a certain level and you can supply an external reference voltage at that level, you achieve your device's maximum resolution. You also can use external reference voltages to apply a gain to a DC voltage or to a time-varying waveform.

Refer to the specifications for your device for additional information.

FD-11637 Signal Conditioning

Learn about excitation, quarter-bridge completion, and shunt calibration for the FD-11637.

Excitation

The FD-11637 supports internal excitation at 3 V, 5 V, and 10 V. 10 V excitation is only supported in Full-Bridge and Half-Bridge modes.

Quarter-Bridge Completion

The FD-11637 supports internal quarter-bridge completion with resistances of 120 Ω and 350 Ω.

Shunt Calibration

The FD-11637 has a built-in resistor for shunt calibration. Shunt calibration also only

works in Quarter-Bridge mode.

Initialized States for Terminals and Output Channels

When you use MAX or the NI-DAQmx API to reset a device, NI-DAQmx sets terminals and output channels to an initialized state.

- Digital I/O Lines

  NI-DAQmx sets all digital I/O lines to the configured power-up state. NI-DAQmx tristates all digital I/O lines on devices that do not support configurable power-up states. NI-DAQmx outputs 0 on all digital output-only lines on devices that do not support configurable power-up states.

- PFI Lines

  NI-DAQmx tristates all PFI lines, unless they are also digital I/O lines. In that case, the digital I/O line behavior applies.

- AO Channels

  On E Series, S Series, and AO Series devices, NI-DAQmx does not alter the AO channels. They continue to generate the DC voltage you last set them to.

  With static AO devices, all voltage outputs are at their user-defined values to full accuracy within 1 s of power-up device reset. Before this time, the voltage outputs can float to unspecified values.

  On DSA devices, NI-DAQmx sets all AO channels to high impedance.

  On M Series, C Series, and X Series, the AO channels are set to 0 Volts.

Input Limits Fault Detection

The NI-DAQmx input limits fault properties allow you to check for samples that are outside the configurable upper and lower limits for each channel in the task.

The fault detection applies to both positive and negative inputs. For instance, if you specify a lower limit of 2 mA and an upper limit of 12 mA, NI-DAQmx detects a fault at 15 mA and -15 mA, but not at -6 mA because it is in the range of -2 mA to -12 mA.

AI.InputLimitsFaultDetectEnable—Enables input limits fault detection.

AI.InputLimitsFaultDetect.UpperLimit—Specifies the level of the upper limit for input limits detection. An input sample outside the upper and lower bounds causes a fault.

AI.InputLimitsFaultDetect.LowerLimit—Specifies the level of the lower limit for input limits detection. An input sample outside the upper and lower bounds causes a fault.

InputLimitsFaultChansExist—Indicates if the device or devices detected a sample that was outside the upper or lower limits configured for each channel in the task. Reading this property clears the input limits fault channel status for all channels in the task.

InputLimitsFaultChans—Lists the virtual channels that have detected samples beyond the upper or lower limits configured for each channel in the task. You must read InputLimitsFaultChansExist before you read this property.

Internal PLL Unlock Status

Two NI-DAQmx Read attributes/properties allow you to check if the PLL was unlocked during data acquisition for a task. The first is PLL.UnlockedChansExist. This attribute/property returns a Boolean true if one or more channel PLLs became unlocked during the previous acquisition. The second is PLL.UnlockedChans. This attribute/property returns an array of strings indicating which channel PLLs (if any) became unlocked.

PLL.UnlockedChansExist can only be used when the task is in a stop state but is still reserved. You can also explicitly commit, start, and then stop the task without explicitly uncommitting using the DAQmx Control Task VI.

NI 9775 Considerations

The NI 9775 module operates in two different modes, Continuous Mode and Record Mode. You can set the NI 9775 operation to be Record Mode by setting the

DataXferReqCond attribute/property to When Acquisition Complete. Any other supported value of the DataXferReqCond attribute/property will force the NI 9775 into Continuous Mode. If a value is not set for DataXferReqCond attribute/property, the module will coerce a value based on other task settings.

Continuous Mode

While in Continuous Mode, the NI 9775 returns data as the data is acquired from the source.

The module does not use the module's onboard buffer when in Continuous Mode. NI-DAQmx will automatically put the NI 9775 in Continuous Mode if the task is set for continuous acquisition or if the specified rate is less than or equal to 4 MS/s aggregate for each module.

Continuous Mode Triggering

In Continuous Mode, the NI 9775 cannot be the source for Analog Edge or Analog Multi Edge Triggering.

The NI 9775 also does not support Analog Triggering or Retriggering, unless in a task with the NI 9205 or NI 9206. While in Continuous Mode, Reference and Start triggers can be configured on the same task.

Record Mode

While in Record Mode, the data is stored on a buffer in the module until the entire acquisition is complete. The data on the buffer is stored in records based on a trigger signal sent to the device.

Acquisition is considered complete when any of the following conditions are satisfied:

- The maximum number of triggers are detected and the data for all detected triggers have been acquired.

  By default, the NI 9775 acquires one record. To acquire more than one record, you can set the Retriggerable attribute/property to true, which will give you as many records that can fit in the buffer, up to 32 records. Or, you can use the Maximum

Number of Triggers to Detect attribute/property to specify how many records you want, up to 32 records. Any ignored triggers do not count towards the maximum of 32 records.

- The Trigger Window has elapsed.

  This time setting can be configured through the Trigger Window attribute/ property. The Trigger Window begins when the modules receive a sync pulse, which synchronizes modules in a task. This sync pulse is sent during the commit stage of the task which happens in the Start Task function/VI. If the Trigger Window expires, the module finishes acquiring post-trigger data. If the module is already finished acquiring post-trigger data, the module returns all of the acquired data.

- The Retrigger Window has elapsed.

  This time setting can be configured through the Retrigger Window attribute/ property. The Retrigger Window begins once the module has detected a trigger and resets every time it detects another trigger. If the Retrigger Window expires, the module finishes acquiring post-trigger data. If the module is already finished acquiring post-trigger data, the module returns all of the acquired data.

NI-DAQmx will automatically put the NI 9775 in Record Mode if the channels on the module are used for trigger sources or if the rate is specified to be greater than 4 MS/s aggregate for finite acquisitions. When in Record Mode, the NI 9775 may only be in the same task as other NI 9775 modules in Record Mode.

Record Mode Triggering

In Record Mode, the NI 9775 supports the following triggers and configurations:

- Digital Edge Triggering.
- A single channel configured as the source when using Analog Edge Triggering.
- Up to 4 channels configured as the source, but all of the source channels must be on the same device when using Analog Multi Edge Triggering.
- Configuration of a single trigger type. You can have Start or Reference triggers configured, but not both.
- Retriggering on any edge trigger type: Digital Edge, Analog Edge, and Analog Multi Edge.

> ✎ **Note** The module will ignore triggers when acquiring pre-trigger buffer data or post-trigger data.

Onboard Memory Behavior

The retrieval of data from the onboard memory of the module may be faster than the chassis can transfer it to the host. If this occurs, the chassis FIFO will overflow, erroring the task and causing a loss of data. To prevent this, set the Data Transfer Maximum Rate attribute/property to a rate lower than the system's bandwidth.

NI USB-TC01 Considerations

If you are acquiring data from the device in a NI-DAQmx task or using the device's temperature logger, the NI USB-TC01 launch screen cannot acquire data.

In MAX, the **Reset Device** option is disabled for the NI USB-TC01. If you attempt to reset the device with the Reset Device VI/function, it returns an error if any task is using the device. To reset the NI USB-TC01, disconnect and reconnect the device.

Open Channel Detection

The DAQmx Read OpenChansExist and OpenChans attributes/properties allow you to check for open channels.

OpenChansExist returns a Boolean of true if one or more channels were disconnected since the last time the attribute/property was queried, and OpenChans returns the virtual channel names of the channels that were disconnected. You must query the OpenChansExist attribute/property before querying the OpenChans attribute/property.

OpenChansExist reads the open channel condition from the device and caches it in the driver. Subsequent reads of OpenChans attribute/property will read the channel information cached in the driver from the previous OpenChansExist query.

NI-DAQmx returns all data whether an open channel is detected. If your application

requires open channel checking, it is recommended that you read the open channel attributes/properties after each call to Read/Write. Your program should discard questionable data or return a flag when the driver reports an open channel.

You can also check for shorted channels using the DAQmx Read OvercurrentChansExist and OvercurrentChans attributes/properties. Refer to Overcurrent Detection.

Related concepts:

- Overcurrent Detection

Open Current Loop Detection

Two NI-DAQmx Read/Write attributes/properties allow you to check for disconnected sensors. The first is OpenCurrentLoopChansExist. This attribute/property returns a Boolean true if one or more channels experience an open current loop condition. The second is OpenCurrentLoopChans. This attribute/property returns an array of strings indicating which channels (if any) experienced an open current loop condition. You must query the OpenCurrentLoopChansExist attribute/property before querying the OpenCurrentLoopChans attribute/property.

OpenCurrentLoopChansExist reads the open current loop condition from the device and caches it in the driver. Subsequent reads of OpenCurrentLoopChans attribute/ property will read the open current loop channel information cached in the driver from the previous OpenCurrentLoopChansExist query.

> ✏️ **Note**
>
> - NI-DAQmx returns all data whether or not an open current loop occurs. If your application requires open current loop checking, it is recommended that you read the open current loop attributes/ properties after each call to Read. Your program should discard questionable data or return a flag when the driver reports an open current loop.
> - For Analog Input devices with IEPE, the IEPE excitation current source

must be enabled for open current loop detection to work. If IEPE is not turned on, an error is returned when OpenCurrentLoopChans is read.

Open Thermocouple Detection (OTD)

Breaking the hot junction of a thermocouple or disconnecting a thermocouple signal results in an open thermocouple channel. This causes the channel to return invalid data. Using open thermocouple detection, you can determine if a thermocouple channel is disconnected.

Devices have a set of pull-up and bias resistor networks that connect to channels, forming a voltage divider when a thermocouple is connected and working properly. When a thermocouple breaks or becomes open, the open thermocouple detection circuitry uses a small current source to push the input voltage out of range, which is possible to detect in software using limits or conditions. In normal operation, this current causes a voltage error on the input, known as a lead offset, when pushing against a source resistance. In many applications, this error is minimal, but in some applications where high accuracy is required and there are large source resistances caused by long, narrow gauge thermocouple wires, the error can be significant.

To determine if a thermocouple is disconnected or becomes open on the FD-11613, FD-11614, NI 4353, NI 9212, NI 9213, or NI 9214, you can use the OpenThrmcplChansExist and OpenThrmcplChans attributes/properties. OpenThrmcplChansExist returns a Boolean of true if one or more channels were disconnected since the last time the attribute/property was queried, and OpenThrmcplChans returns the virtual channel names of the channels that were disconnected.

Lead Offset Nulling

To eliminate lead offset, you can disable open thermocouple detection, which eliminates the bias current. If you are using an SCXI module, you can disable open thermocouple detection by removing the bias resistor networks. On the NI 4353 and NI 9214, you can disable OTD current with the OpenThrmcplDetectionEnable attribute/

property.

If an application requires high-accuracy and high-resistance thermocouple wires, but you do not want to disable the ability to detect open thermocouples, you can perform lead offset nulling. On the NI 9214, you can calibrate lead offset nulling programmatically. Lead offset nulling calibration compensates for lead offset error by taking the difference between a measurement with the bias current from open thermocouple detection and one without the bias current. This difference is then stored in the AI.Thrmcpl.LeadOffsetVoltage attribute/property and is applied to measurements to compensate for current from open thermocouple detection.

Overcurrent Detection

Two NI-DAQmx Read/Write attributes/properties allow you to check for shorted channels. The first is OvercurrentChansExist. This attribute/property returns a Boolean true if one or more channels experience an overcurrent condition. The second is OvercurrentChans. This attribute/property returns an array of strings indicating which channels (if any) experienced an overcurrent condition. You must query the OvercurrentChansExist attribute/property before querying the OvercurrentChans attribute/property.

OvercurrentChansExist reads the overcurrent condition from the device and caches it in the driver. Subsequent reads of OvercurrentChans attribute/property will read the overcurrent channel information cached in the driver from the previous OvercurrentChansExist query.

> **Note** For the NI 4610, use the AO.PowerAmp.Overcurrent attribute/property to check for shorted channels. On the NI 4610, AO.PowerAmp.Overcurrent detects the overcurrent condition for the specified channel and caches that information in the driver.

NI-DAQmx returns all data whether or not a short occurs. If your application requires overcurrent checking, it is recommended that you read the overcurrent attributes/ properties after each call to Read/Write. Your program should discard questionable data or return a flag when the driver reports a short.

IEPE must be turned on for overcurrent detection to work (except for the NI 4322, NI 4610, NI 9219, and NI 9269). If IEPE is not turned on, an error is returned when OvercurrentChansExist is read.

Shorted current channels are known as short circuits. Overcurrent detection attributes/properties can be used to detect short circuits.

Overtemperature Detection

An overtemperature condition occurs when the ambient temperature of the device exceeds a safe operating level. The device may disable any overtemperature channels until the ambient temperature returns to a safe level. Use the OvertemperatureChansExist attribute/property to determine if the device is experiencing an overtemperature condition. Devices may have front panel LEDs that will indicate an overtemperature condition has occurred.

> ✎ **Note** Refer to device specific documentation for further information on meaning of LED status.

Remote and Local Sensing

If you use a cable of significant length, you might experience reduced accuracy in your measurement due to reduced voltage from wire resistance. You can adjust for this voltage drop using either remote sense or local sense.

Remote sense uses a separate connection to quantify the voltage drop. This accounts for the length of the cable in the original connection. To use remote sense, you must connect the RS+ and RS- terminals on the device to the primary winding terminals on the sensor.

Local sense measures the voltage drop internally at the excitation terminals. This method does not require the use of additional wires, but is not as accurate as remote sense.

On the PXIe-4340, you can set each channel to use either remote sense or local sense

by using the AI.Excit.Source attribute/property.

Power Supply and Power Channel Considerations

Refer to **Hardware Input and Output** » **DAQmx** » **Power** in the LabVIEW Example Finder for examples that implement many of the following considerations.

Default Values

The following property values take effect when the task is committed or started.

Property Name	Default Value
Pwr.IdleOutputBehavior	Maintain Existing Value
Pwr.RemoteSense	Local Sense
Pwr.OutputEnable	True
Pwr.CurrentSetpoint	0.03A
Pwr.VoltageSetpoint	0 V

Configuring Voltage and Current Setpoints

Use the DAQmx Create Channel VI/function to set the current and voltage setpoints. Two properties reflect the current and voltage setpoints for a power channel, and are settable at runtime via the DAQmx Channel property node:

- **Pwr.Current.Setpoint**—The constant output current, in amperes. If the load draws current greater than this value, output voltage is reduced and the device operates in constant current mode.
- **Pwr.Voltage.Setpoint**—The constant output voltage, in volts.

Configuring and Reading Power Output States and Behavior

Use the DAQmx Create Channel VI/function with the Pwr.OutputEnable property to enable or disable power channel output. At runtime, set this property via the DAQmx Channel property node.

Use the DAQmx Channel property node with the Pwr.IdleOutputBehavior property to enable or disable power channel output when the task is uncommitted.

Use the DAQmx Channel property node to read the Pwr.OutputState property and determine the channel state.

> **Note** Some DAQmx Read instances/functions return power channel samples in I16 format. These instance will always return a value of 0 for current and voltage while a power channel is disabled. Verify the power output state to validate a current or voltage sample value of 0.

Detecting Auxiliary Power Errors

Use the DAQmx Read property node to read auxiliary power supply errors. Two properties reflect the error state for auxiliary power.

> **Note** You must read AuxPowerErrorChansExist before you read AuxPowerErrorChans. Otherwise you will receive an error message.

- **AuxPowerErrorChansExist**—Indicates if the device detected something is wrong with an auxiliary power supply.
- **AuxPowerErrorChans**—Indicates a list of names of any auxiliary power supply error virtual channels.

Detecting Remote Sense Errors

Use the DAQmx Read property node to read remote sense errors. Two properties reflect the error state for Remote Sense.

> **Note** You must read RemoteSenseErrorChansExist before you read RemoteSenseErrorChans. Otherwise you will receive an error message.

- **RemoteSenseErrorChansExist**—Indicates if the device detected an error on a hardware or remote sense connection. You must disable the output and resolve the hardware connection issue to clear the remote sense error status for all channels in the task.
- **RemoteSenseErrorChans**—Indicates a list of names of any remote sense error

virtual channels.

Detecting Over-Temperature Errors

Use the DAQmx Read property node to read over-temperature errors. Two properties reflect an over-temperature condition.

> ✎ **Note** You must read OvertemperatureChansExist before you read OvertemperatureChans. Otherwise you will receive an error message.

- **OvertemperatureChansExist**—Indicates if the device detected an overtemperature condition in any virtual channel in the task. Reading this property clears the overtemperature status for all channels in the task.
- **OvertemperatureChans**—Indicates a list of names of any overtemperature virtual channels.

Detecting Reverse Voltage Errors

Use the DAQmx Read property node to read reverse voltage errors. Two properties reflect the reverse voltage error state of channel

> ✎ **Note** You must read ReverseVoltageErrorChansExist before you read ReverseVoltageErrorChans. Otherwise you will receive an error message.

- **ReverseVoltageErrorChansExist**—Indicates if the device detected a reverse voltage error in any channel in the task. Reverse voltage error occurs if the local voltage is equal to the negative saturated voltage.
- **ReverseVoltageErrorChans**—Indicates a list of names of any reverse voltage error virtual channels.

Configuring Remote Sense

Use the DAQmx Channel property to set the Pwr.RemoteSense property.

- DAQmx Read will return local voltage if Pwr.RemoteSense is configured to `Local`.
- DAQmx Read will return remote voltage if Pwr.RemoteSense if configured to `Remote`.

Power Supply Fault Detection

The NI 9265 and NI 9266 can detect if there is an insufficient external power or no external power connected to the Vsup terminal. Two NI-DAQmx Read attributes/properties allow you to check the external power. The first is PowerSupplyFaultChansExist. This attribute/property returns a Boolean true if one or more channels lack external power. The second is PowerSupplyFaultChans. This attribute/property returns an array of strings indicating which channels (if any) experienced a power supply fault. You must query the PowerSupplyFaultChansExist attribute/property before querying the PowerSupplyFaultChans attribute/property.

Push-Pull and Open Collector Mode

The NI USB-6000 /6001/6002/6003 are programmable as either push-pull or open-collector (open-drain) mode. The default configuration is push-pull.

The NI USB-6008 has open-collector (open-drain) mode only, but each channel on the NI USB-6501, the NI USB-6009, and SensorDAQ are programmable as either push-pull or open-collector (open-drain) mode. The default configuration of the SensorDAQ, the NI USB-6008, the NI USB-6009, and the NI USB-6501 DIO ports is open-drain, allowing 5 V operation, with an onboard 4.7 k pull-up resistor. An external, user-provided, pull-up resistor can be added to increase the source current drive up to a 8.5 mA limit per line. Refer to the device documentation for more information and instructions on determining the value of the pull-up resistor.

Querying Device Capabilities with C Series Devices

When querying DAQmx Device and DAQmx Physical Channel attributes/properties with C Series devices, the supported attributes/properties depend on the slot you plug a device into. If the device is not in a supported slot, you cannot perform counter I/O, nor can you query counter-specific device capabilities such as the counter size.

Reading Available Samples on USB or Ethernet DAQ

Reading -1 or querying AvailableSamplesPerChannel sends a request to the device forcing it to flush any samples that may be sitting in the buffer.

When using USB or Ethernet DAQ devices, these samples may not reach your computer's buffer in time, resulting in an incorrect number of samples available being displayed. To ensure all samples get read, either use a second read also querying -1 (which will then contain the desired samples), or specify the number of samples to read so the device waits until the specified number of samples have entered the computer's buffer before returning the data.

RTSI Triggering with M Series USB and NI ELVIS II Family Devices

M Series USB and NI ELVIS II Family devices do not support RTSI triggering.

SC Express Smart Accessory Connections

The following connections are supported on these SC Express accessories:

Accessory	Supported Connections	Description
CAL-4353	_tc_calibration	Connects all TC channels to the calibration input. This connection is used to verify and adjust TC gain accuracy.
	_cjc_calibration	Connects one CJC channel to the calibration input. This connection is used to verify and adjust CJC gain and offset accuracy.
	_short_tc_terminals	Connects all TC channels together.
RM-4339	_remote_sense_floating	Disconnects remote sensing for calibration.
	_cal_channel	Connects all channels to an external calibration source.
RM-4302	_cal_channel	Connects all channels to an external calibration source.
RM-4304	_cal_channel	Connects all channels to an external calibration source.

SCC Signal Conditioning Device Considerations

The following section applies only to analog input (AI) SCC modules.

In a single stage AI SCC configuration, you connect your external signal to an SCC module which conditions the signal and passes it to the DAQ device. You can install single stage AI modules in sockets J1-J8 in the SC-2345 carrier.

Sometimes, you can cascade two AI SCC modules together on a single AI channel to form a dual-stage configuration. In this configuration, you connect the external signal to the first-stage AI module, which conditions the signal and passes it to the second-stage AI module. Then, the signal is passed to the DAQ device. First-stage of dual-stage AI modules can be located in sockets J9-J16. Second-stage of dual-stage AI modules can be located in sockets J1-J8.

An example of a dual-stage configuration is a voltage attenuator module (SCC-A10) in the first-stage SCC slot followed by a lowpass filter module (SCC-LP01) in the second-stage SCC slot to create a combined attenuator and lowpass filter signal conditioning on the specified AI channel. The following table shows all the SCC devices that support dual-stage configuration.

SCC Module	Single Stage AI (J1-J8)	First-Stage of Dual-Stage AI (J9-J16)	Second-Stage of Dual-Stage AI (J1-J8)
SCC-AI Series	Yes	Yes	No
SCC-A10	Yes	Yes	No
SCC-RTD01	Yes	Yes	No
SCC-CI20	Yes	Yes	No
SCC-ACC01	Yes	Yes	No
SCC-TC Series	Yes	Yes	No
SCC-FV01	Yes	No	Yes
SCC-LP Series	Yes	Yes	Yes

SCC Module	Single Stage AI (J1-J8)	First-Stage of Dual-Stage AI (J9-J16)	Second-Stage of Dual-Stage AI (J1-J8)
SCC-FT01	Yes	Yes	Yes
SCC-SG Series	Yes	Yes	No

SCC modules that are not listed in the table above do not support dual-stage configuration. This includes all analog output, digital input, and digital output SCC modules.

Self-Powered Compared to Bus-Powered USB Devices

Bus-powered USB DAQ devices require no external power source. Examples of bus-powered devices include the following.

- NI mioDAQ devices
  - USB-6421
  - USB-6423
  - USB-6451
  - USB-6453
- M Series devices
  - NI 6210
  - NI 6211
  - NI 6212
  - NI 6215
  - NI 6216
  - NI 6218

Self-powered USB devices require an external power source such as a wall outlet or a battery. Self-powered devices can drive higher currents than bus-powered devices.

Setting Power-Up States for M Series, NI 670*X*, and Software-Timed Digital I/O Devices

You can set the state of physical channels for some devices when your computer is powered on or the device is reset in NI-DAQmx. However, for all NI-DAQmx simulated devices, power-up states are not persisted.

> ⚠️ **Caution** Devices have limited numbers of writes to the EEPROM, so change power-up states infrequently.

Setting Digital States for M Series, X Series and Software-Timed Digital I/O Devices

You can set the digital power-up state for M Series, X Series, and software-timed digital I/O devices to logic low, logic high, or tristate (floating) in MAX. You also can set power-up states with the Set Power Up States (Digital) function/VI, but using MAX is the recommended method. You can only specify a programmable power-up state of tristate on devices with configurable direction. Refer to your device documentation to see if your device supports configurable direction. For NI 6230/36 devices, you can also specify a power-up state of tristate for digital output. The power-up state can be specified on a port basis only. The power-up state of all other NI 623x devices can be specified by line but cannot be set to tristate.

> 📝 **Note** I/O direction on software-timed digital I/O devices is port configurable only*. Therefore, you can set the power-up state to tristate only on a port-by-port basis. You can, however, set individual digital output lines in a port to logic low or logic high.
>
> * Except on the PCIe/PXIe-6509 which are also configurable by line.

Setting Analog States for NI 4322 Devices

You can set the analog power-up state for NI 4322 devices in MAX. You also can set power-up states with the DAQmx Set Power Up States (Analog With Output Type) function/VI.

Setting Analog States for NI 670*x* Devices

You can set the analog power-up state for NI 670x devices in MAX. You also can set power-up states with the DAQmx Set Power Up States (Analog) function/VI, but using MAX is the recommended method.

Supported Device ID Numbers

Find the ID for your supported device.

For information about which versions of NI-DAQmx support your device, go to `ni.com/info` and enter `rdsoftwareversion`.

Table 6. Supported device ID numbers

Device ID	Device name
0x0160	PCI-DIO-96
0x075C	DAQCard-DIO-24
0x075F	DAQCard-6715
0x1150	PCI-DIO-32HS
0x1290	PCI-6704
0x12B0	PCI-6534
0x1310	PCI-6602
0x1320	PXI-6533
0x1360	PXI-6602
0x13C0	PXI-6508
0x1490	PXI-6534
0x14E0	PCI-6110
0x14F0	PCI-6111
0x1710	PXI-6509
0x17D0	PCI-6503

Device ID	Device name
0x1870	PCI-6713
0x1880	PCI-6711
0x1920	PXI-6704
0x1AD0	PCI-6133
0x1AE0	PXI-6133
0x1E30	PCI-6624
0x1E40	PXI-6624
0x2410	PCI-6733
0x2420	PXI-6733
0x2430	PCI-6731
0x24F0	PXI-4472
0x2510	PCI-4472
0x2520	PCI-4474
0x27A0	PCI-6123
0x27B0	PXI-6123
0x2B10	PXI-6527
0x2B20	PCI-6527
0x2B80	PXI-6713
0x2B90	PXI-6711
0x2C60	PCI-6601
0x2C90	PCI-6703
0x2CC0	PXI-6608
0x2EC0	PXI-6115
0x2ED0	PCI-6115
0x2EE0	PXI-6120
0x2EF0	PCI-6120
0x7023	PXI-2593

Device ID	Device name
0x703F	PXI-2566
0x7040	PXI-2567
0x704C	PXI-2530
0x7067	PXI-2529
0x7073	PCI-6723
0x707E	PXI-4462
0x7085	PCI-6509
0x7086	PXI-6528
0x7087	PCI-6515
0x7088	PCI-6514
0x708C	PXI-2568
0x708D	PXI-2569
0x709F	USB-9421
0x70A1	USB-9472
0x70A2	USB-9481
0x70A4	USB-9201
0x70A5	USB-9221
0x70A7	USB-9263
0x70A8	USB-9233
0x70A9	PCI-6528
0x70AA	PCI-6229
0x70AB	PCI-6259
0x70AC	PCI-6289
0x70AD	PXI-6251
0x70AE	PXI-6220
0x70AF	PCI-6221
0x70B0	PCI-6220

Device ID	Device name
0x70B1	PXI-6229
0x70B2	PXI-6259
0x70B3	PXI-6289
0x70B4	PCI-6250
0x70B5	PXI-6221
0x70B6	PCI-6280
0x70B7	PCI-6254
0x70B8	PCI-6251
0x70B9	PXI-6250
0x70BA	PXI-6254
0x70BB	PXI-6280
0x70BC	PCI-6284
0x70BD	PCI-6281
0x70BE	PXI-6284
0x70BF	PXI-6281
0x70C0	PCI-6143
0x70C3	PCI-6511
0x70C8	PCI-6513
0x70C9	PXI-6515
0x70CC	PCI-6512
0x70CD	PXI-6514
0x70D0	PXI-2570
0x70D1	PXI-6513
0x70D2	PXI-6512
0x70D3	PXI-6511
0x70D4	PCI-6722
0x70E1	PXI-2532

Device ID	Device name
0x70F2	PCI-6224
0x70F3	PXI-6224
0x70FF	PXI-6723
0x7100	PXI-6722
0x710D	PXI-6143
0x7124	PCI-6510
0x7125	PCI-6516
0x7126	PCI-6517
0x7127	PCI-6518
0x7128	PCI-6519
0x712C	USB-9265
0x712E	USB-9421 (DSUB)
0x7132	USB-9472 (DSUB)
0x7137	PXI-2575
0x713C	PXI-2585
0x713D	PXI-2586
0x7146	PCI-6132
0x7147	PXI-6132
0x7148	PCI-6122
0x7149	PXI-6122
0x7150	PXI-2564
0x715F	NI 9221
0x7160	NI 9421
0x7161	NI 9421 (DSUB)
0x7162	NI 9472
0x7163	NI 9472 (DSUB)
0x7164	NI 9481

Device ID	Device name
0x7165	NI 9401
0x716B	PCI-6230
0x716C	PCI-6225
0x716D	PXI-6225
0x716F	PCI-4461
0x7170	PCI-4462
0x7171	PCI-6010
0x7177	PXI-6230
0x717A	USB-6008
0x717B	USB-6009
0x717D	PCIe-6251
0x717F	PCIe-6259
0x718A	USB-6501
0x718B	PCI-6521
0x718C	PXI-6521
0x7191	PCI-6154
0x71A1	USB-9201 (DSUB)
0x71A2	USB-9221 (DSUB)
0x71A5	PXI-2594
0x71A7	PXI-2595
0x71A9	PXI-2596
0x71AA	PXI-2597
0x71AB	PXI-2598
0x71AC	PXI-2599
0x71B0	NI 9211
0x71B1	NI 9215
0x71B2	NI 9215 (BNC)

Device ID	Device name
0x71B3	NI 9205 (DSUB)
0x71B4	NI 9263
0x71BB	PXI-2584
0x71BC	PCI-6221 (37-pin)
0x71C2	USB-9239
0x71C3	USB-9237
0x71C5	PCI-6520
0x71C6	PXI-2576
0x71D9	USB-9211A
0x71DA	USB-9215A
0x71DB	USB-9215A (BNC)
0x71DF	USB-6525
0x71E0	PCI-6255
0x71E1	PXI-6255
0x7209	PCI-6233
0x720A	PXI-6233
0x720B	PCI-6238
0x720C	PXI-6238
0x7252	USB-6251
0x7253	USB-6259
0x7263	NI 9234
0x7264	NI 9206
0x7265	NI 9205
0x726F	USB-6210
0x7270	USB-6211
0x7271	USB-6215
0x7272	USB-6218

Device ID	Device name
0x7279	PCI-6232
0x727A	PXI-6232
0x727B	PCI-6239
0x727C	PXI-6239
0x7281	PCI-6236
0x7282	PXI-6236
0x7283	PXI-2554
0x7285	NI 9237
0x72A0	USB-6251 (Mass Termination)
0x72A1	USB-6259 (Mass Termination)
0x72B5	USB-9234
0x72B9	NI 9411
0x72BA	NI 9422
0x72BB	NI 9423
0x72BC	NI 9435
0x72BD	NI 9474
0x72BE	NI 9485
0x72BF	NI 9403
0x72C0	NI 9425
0x72C1	NI 9476
0x72C2	NI 9477
0x72C3	NI 9264
0x72C4	NI 9265
0x72C5	NI 9201
0x72C6	NI 9201 (DSUB)
0x72C7	NI 9221 (DSUB)
0x72C8	NI 9203

Device ID	Device name
0x72C9	NI 9217
0x72CA	NI 9219
0x72CB	NI 9239
0x72CC	SensorDAQ
0x72D0	PXI-2545
0x72D1	PXI-2546
0x72D2	PXI-2547
0x72D3	PXI-2548
0x72D4	PXI-2549
0x72D5	PXI-2555
0x72D6	PXI-2556
0x72D7	PXI-2557
0x72D8	PXI-2558
0x72D9	PXI-2559
0x72DC	USB-6221
0x72DE	USB-6229
0x72E8	PXIe-6251
0x72E9	PXIe-6259
0x72EF	PXI-4498
0x72F0	PXI-4496
0x72F3	USB-6005 VSA
0x72F6	USB-9264
0x72FA	NI 9229
0x72FD	USB-9229
0x72FF	USB-6509
0x730C	USB-9219
0x731C	PXI-2535

Device ID	Device name
0x731D	PXI-2536
0x7322	PXIe-6124
0x7326C4C4	PCIe-6509
0x7327	PXI-6529
0x732D	USB-6255
0x732E	USB-6255 (Mass Termination)
0x732F	USB-6225
0x7330	USB-6225 (Mass Termination)
0x7335	PXI-2533
0x7336	PXI-2534
0x7337	NI 9402
0x7338	NI 9375
0x7339	USB-6212
0x733B	USB-6216
0x733F	USB-6281
0x7340	USB-6281 (Mass Termination)
0x7342	PXI-4461
0x7343	USB-6289
0x7344	USB-6289 (Mass Termination)
0x7345	USB-6221 (BNC)
0x7346	USB-6229 (BNC)
0x7347	USB-6251 (BNC)
0x7348	USB-6259 (BNC)
0x7359	PXI-4495
0x7367	USB-9239 (BNC)
0x7368	USB-9229 (BNC)
0x7369	USB-9263 (BNC)

Device ID	Device name
0x737A	NI 9229 (BNC)
0x737B	NI 9239 (BNC)
0x737C	NI 9263 (BNC)
0x7381	NI 9235
0x7382	NI 9236
0x7388	NI 9225
0x7389	USB-6212 (Mass Termination)
0x738A	USB-6216 (Mass Termination)
0x73A1	PXIe-4498
0x73A2	PXIe-4496
0x73A6	ELVIS II
0x73C5	PXIe-2527
0x73C6	PXIe-2529
0x73C9	PXIe-2532
0x73CA	PXIe-2569
0x73CB	PXIe-2575
0x73CC	PXIe-2593
0x73D1	USB-4432
0x73E2	NI 9213
0x73E3	NI 9426
0x73E4	NI 9475
0x73E5	NI 9478
0x73E6	NI 9237 (DSUB)
0x73E7	ELVIS II+
0x73F4	PXI-2515
0x73FC	USB-6212 (BNC)
0x73FD	USB-6216 (BNC)

Device ID	Device name
0x73FE	USB-6218 (BNC)
0x7420	NI 9227
0x7425C4C4	PCIe-6320
0x7427C4C4	PCIe-6321
0x7429C4C4	PCIe-6323
0x742AC4C4	PXIe-6341
0x742BC4C4	PCIe-6341
0x742DC4C4	PCIe-6343
0x742FC4C4	PCIe-6351
0x7431C4C4	PCIe-6353
0x7432C4C4	PXIe-6361
0x7433C4C4	PCIe-6361
0x7434C4C4	PXIe-6363
0x7435C4C4	PCIe-6363
0x7436C4C4	PXIe-6356
0x7437C4C4	PXIe-6358
0x7438C4C4	PXIe-6366
0x7439C4C4	PXIe-6368
0x7448	PXI-2510
0x7449	USB-9213
0x744F	USB-4431
0x7454	PXI-2512
0x7455	PXI-2514
0x7456	PXIe-2512
0x7457	PXIe-2514
0x745C	USB-9264 (DSUB)
0x7492C4C4	PXIe-4300

Device ID	Device name
0x7493	cDAQ-9188
0x7494	NI 9264 (DSUB)
0x74A4	cDAQ-9178
0x74A5	cDAQ-9174
0x74A8C4C4	PXIe-4330
0x74A9C4C4	PXIe-4331
0x74AE	PXIe-2515
0x74B2C4C4	PXIe-4353
0x74B4	PXI-2531
0x74B5	PXIe-2531
0x74B7	USB-TC01
0x74DF	NI 9207 (DSUB)
0x74E0	NI 9208 (DSUB)
0x74E6	NI 9269
0x74F7	USB-6343
0x74F8	USB-6341
0x74FA	USB-6351
0x74FB	USB-6353
0x74FD	USB-6361
0x74FE	USB-6363
0x7510	NI 2810
0x7511	NI 2811
0x7512	NI 2815
0x7513	NI 2816
0x7528	PXIe-4497
0x7529	PXIe-4499
0x752A	PXIe-4492

Device ID	Device name
0x752B	NI 9214
0x7530	NI 9222
0x7531	NI 9223
0x7559	NI 9232
0x755B	myDAQ
0x755F	cDAQ-9181
0x7560	cDAQ-9191
0x7561	USB-6356 (32 MS)
0x7563	USB-6356 (64 MS)
0x7567	USB-6366 (32 MS)
0x7569	USB-6366 (64 MS)
0x7578	NI 9375 (DSUB)
0x757B	NI 2812
0x757E	NI 2813
0x7581	NI 2814
0x7584	NI 2817
0x7587	NI 2833
0x758A	NI 2834
0x7598	PXI-2571
0x759D	USB-6361 (Mass Termination)
0x759E	USB-6363 (Mass Termination)
0x75A1	USB-6366 (64 MS) (Mass Termination)
0x75BA	PXI-2543
0x75BB	PXIe-2543
0x75CFC4C4	PXIe-4357
0x75DA	USB-6341 (BNC)
0x75DB	USB-6343 (BNC)

Device ID	Device name
0x75DC	USB-6361 (BNC)
0x75DD	USB-6363 (BNC)
0x75DE	USB-6356 (32 MS) (BNC)
0x75DF	USB-6366 (64 MS) (BNC)
0x75EB	NI 9469
0x75F0	cDAQ-9171
0x7606	NI 9220
0x7614	cDAQ-9138
0x7616	NI 9220 (DSUB)
0x7617	cDAQ-9184
0x761F	PXI-2540
0x7620	PXIe-2540
0x7621	PXI-2541
0x7622	PXIe-2541
0x7625	USB-6346
0x7628	cDAQ-9139
0x7638	PXI-2720
0x7639	PXI-2722
0x763A	PXIe-2725
0x763B	PXIe-2727
0x763DC4C4	PXIe-6349
0x764B	PXIe-2790
0x764C	PXI-2520
0x764D	PXI-2521
0x764E	PXI-2522
0x764F	PXI-2523
0x7654	PXI-2796

Device ID	Device name
0x7655	PXI-2797
0x7656	PXI-2798
0x7657	PXI-2799
0x765D	PXI-2542
0x765E	PXIe-2542
0x765F	PXI-2544
0x7660	PXIe-2544
0x76ABC4C4	PXIe-4322
0x76AF	USB-6000
0x76BF	USB-6001
0x76C4	USB-6002
0x76C6	USB-6003
0x76C8C4C4	PXIe-6614
0x76C9C4C4	PXIe-6612
0x76DCC4C4	PXIe-4610
0x76E3	USB-9482
0x76E7	cDAQ-9188XT
0x76F5	NI 9482
0x770B	NI 2865
0x770F	NI 9244
0x7711C4C4	PXIe-4464
0x7712C4C4	PXIe-4463
0x7716C4C4	PCIe-6612
0x771EC4C4	PXIe-4339
0x7739	cDAQ-9132
0x773A	cDAQ-9134
0x773C	NI 9222 (BNC)

Device ID	Device name
0x773D	NI 9223 (BNC)
0x7749	NI 9242
0x7751	NI 9238
0x7779	NI 9218 (DSUB)
0x777A	NI 9218
0x777F	NI 9361
0x778D	NI 9437
0x77A5C4C4	PXIe-6345
0x77A6C4C4	PXIe-6355
0x77A7C4C4	PXIe-6365
0x77A8C4C4	PXIe-6375
0x77AB	NI 9212
0x77BE	cDAQ-9133
0x77BF	cDAQ-9135
0x77CAC4C4	PXIe-6738
0x77CBC4C4	PXIe-6739
0x77CE	NI 9230
0x77CF	NI 9260 (BNC)
0x77D0	NI 9260 (Mini XLR)
0x77D8	NI 9216
0x77D9	NI 9226
0x77E6	NI 9344
0x77E7	NI 9247
0x77EB	NI 9251 (Mini XLR)
0x77EC	NI 9250 (BNC)
0x77F1	NI 9209 (DSUB)
0x77F4	cDAQ-9179

Device ID	Device name
0x77F6	NI 9246
0x77F9	NI 9216 (DSUB)
0x77FA	NI 9226 (DSUB)
0x7802C4C4	PXIe-4302
0x7803C4C4	PXIe-4303
0x7805C4C4	PXIe-4305
0x780F	NI 9224
0x7810	NI 9228
0x7829C4C4	PXIe-4340
0x7834	cDAQ-9136
0x7836	cDAQ-9137
0x7844C4C4	PXIe-4480
0x7845C4C4	PXIe-4481
0x7868	NI 9232 (BNC)
0x7869	NI 9230 (BNC)
0x786D	NI 9436
0x7870	Simulated DAQ Device
0x7882C4C4	PXIe-6376
0x7883C4C4	PXIe-6378
0x788B	NI 9202 (DSUB)
0x788D	NI 9202
0x788EC4C4	PXIe-4304
0x78A0	NI 9775
0x78ACC4C4	PXIe-4309
0x78B2	NI 9210 (miniTC)
0x78B5	NI 9266
0x78B9	cDAQ-9189

Device ID	Device name
0x78C7	cDAQ-9185
0x78DFC4C4	PXIe-4310
0x7903	cRIO-9040
0x7905	cRIO-9045
0x7907	cRIO-9043
0x7909	cRIO-9048
0x790D	cRIO-9048 (TPM)
0x790F	cRIO-9042
0x7911	cRIO-9047
0x7913	cRIO-9049
0x7918	NI 9207
0x7919	NI 9208
0x791A	NI 9209
0x7923	NI 9425 (Spring)
0x7924	NI 9476 (Spring)
0x7931	NI 9231
0x7941	FD-11613
0x7943	FD-11637
0x7945	FD-11603
0x7988C4C4	PCIe-6738
0x7998	NI 9262
0x7999C4C4	PCIe-6376
0x799AC4C4	PCIe-6374
0x799EC4C4	PXIe-6386
0x799FC4C4	PXIe-6396
0x79A7C4C4	PCIe-6346
0x79C5	NI 9253

Device ID	Device name
0x79C9	NI 9252
0x79CB	NI 9252 (DSUB)
0x79CF	cRIO-9046
0x79D1	cRIO-9041
0x79E1	cRIO-9053
0x79E2	cRIO-9054
0x79E3	cRIO-9056
0x79E4	cRIO-9057
0x79F7	USB-6349
0x79FD	NI 9266 (DSUB)
0x79FF	FD-11614
0x7A07	FD-11601
0x7A08	USB-6346 (BNC)
0x7A1C	FD-11634
0x7A27	NI 9210
0x7A29	cRIO-9058
0x7A2C	cRIO-9055
0x7A32	FD-11605
0x7A3C	sbRIO-9603
0x7A3D	sbRIO-9608
0x7A3E	sbRIO-9628
0x7A3F	sbRIO-9638
0x7A42	sbRIO-9609
0x7A44	sbRIO-9629
0x7A47	NI AI 0-15
0x7A48	NI AO 0-3
0x7A49	NI DIO 0-3

Device ID	Device name
0x7A4A	NI DIO 4-11
0x7A50	NI DIO 12-19
0x7A51	NI DIO 20-27
0x7A5F	NI 9326
0x7A77C4C4	PXIe-6509
0x7AD5	TS-15100
0x7AD6	TS-15110
0x7AD7	TS-15120
0x7AD8	TS-15130
0x7ADA	TS-15010
0x7ADC	TS-15000
0x7ADE	TS-15050 DIO P0
0x7ADf	TS-15200
0x7B3E	USB-6421
0x7B40	USB-6423
0x7B42	USB-6451
0x7B44	USB-6453
0x9020	PXI-2501
0x9030	PXI-2503
0x9040	PXI-2527
0x9050	PXI-2565
0x9060	PXI-2590
0x9070	PXI-2591

Sync Lock Lost Detection

Network-synchronized devices that are part of a time network are synchronized to a grand master. When the network conditions cause a chassis to lose synchronization,

any task running on that chassis will error.

This can be configured in a task through the DAQmx Channel Property SyncUnlockBehavior, which defaults to StopTaskAndError, and can be disabled by setting the value to IgnoreLostSyncLock. If IgnoreLostSyncLock is configured, the task will keep running regardless of synchronization status. To query the state on a given task, the DAQmx Read or Write Property Sync.UnlockedChansExist returns whether the target is currently locked to the grand master. Sync.UnlockedChans returns the channels from the devices in an unlocked target.

Network-synchronized devices include the following devices:

- cDAQ-9185, and 9189.
- FD-11601, FD-11603, FD-11605, FD-11613, FD-11614, FD-11634, and FD-11637.
- cRIO-9040, 9041, 9042, 9043, 9045, 9046, 9047, 9048, 9049, 9053, 9054, 9055, 9056, 9057, and 9058.
- sbRIO-9603, 9608, 9609, 9628, 9629, and 9638.

Taking Custom Voltage Measurements with the PXIe-4339

The PXIe-4339 supports taking regular (non-ratiometric) voltage measurements using an AI Voltage task. In addition, as of NI-DAQmx 14.5, the PXIe-4339 can perform custom voltage measurements while providing voltage excitation, for example, when using DC powered sensors that return a non-ratiometric voltage. To enable this type of measurement, configure the DAQmx Create Virtual Channel VI to AI Custom Voltage with Excitation mode.

Setting the `use excitation for scaling` input to False will prevent the PXIe-4339's ADCs from scaling the measured voltage ratiometrically to the excitation voltage.

Setting the `use excitation for scaling` input to True, will enable the PXIe-4339 to use the excitation value that the user sets with the DAQmx driver for scaling the ratiometric input. The device will not use the voltage present on the remote sense lines for scaling as it does when providing internal excitation. Thus, while the driver will support using external excitation for ratiometric measurements, this is not a recommended configuration.

© National Instruments   557

The table below covers each possible combination of AI channel type, `voltage excitation source` input value, and `use excitation for scaling` input value. The cases highlighted will throw DAQmx errors, and the reasons for the errors are described in the Use Case and Notes column of the table.

Channel Type	Voltage Excitation Source	Use Excitation for Scaling	Use Case and Notes
Bridge / Strain	Internal	True	Basic bridge measurement
	Internal	False	Bridge measurements are always ratiometric
	External	True	Bridge measurement with external excitation
	External	False	Bridge measurements are always ratiometric
	None	True	Bridge measurements require excitation
	None	False	Bridge measurements require excitation
Custom Voltage	Internal	True	Basic bridge measurement
	Internal	False	Powered sensor using PXIe-4339 excitation
	External	True	Bridge measurement using external excitation
	External	False	Powered sensor using external excitation
	None	True	Bridge measurements require excitation
	None	False	Basic voltage measurement

Note An RM-4339 is required to select external excitation.

Related reference:

- Using the RM-4339 with the PXIe-4339

Time-Based Features for Network-Synchronized Devices

Network-synchronized devices feature automatic network-based synchronization when connected together across a compatible network. When these devices are synchronized across a compatible network all device timebases, time triggers, and timestamps will automatically be synchronized.

Network-synchronized devices include the following devices:

- cDAQ-9185, and 9189.
- FD-11601, FD-11603, FD-11605, FD-11613, FD-11614, FD-11634, and FD-11637.

- cRIO-9040, 9041, 9042, 9043, 9045, 9046, 9047, 9048, 9049, 9053, 9054, 9055, 9056, 9057, and 9058.
- sbRIO-9603, 9608, 9609, 9628, 9629, and 9638.

The cDAQ-9185/9189 are tethered chassis that can be daisy-chained to each other or connected to external networks that support 802.1AS synchronization. To learn about this feature, refer to the ***cDAQ-9185/9189 User Manual*** and the related concepts in this help.

The FD-116xx are devices that can be daisy-chained to each other and connected to external networks that support 802.1AS synchronization. To learn about this feature, refer to your ***FieldDAQ User Guide*** and the related concepts in this help.

Related concepts:

- [Time Triggering](#)
- [C Series Multichassis Device Tasks](#)
- [Sync Lock Lost Detection](#)
- [Timestamps](#)

Using Chopping to Remove Offset Voltages

Chopping is a feature that can be used to remove offset voltages and other low frequency errors. The signal is measured twice, once normally and once with the inputs inverted. These measurements are then averaged by the device to create each sample.

The NI 4309 supports chopping but disables it by default. Chopping can be enabled with the `AI.Chop.Enable` attribute/property, or when configuring a voltage input task in the DAQ Assistant by selecting the **Enable Chopping** option. Refer to the NI 4309 User Manual for wiring instructions.

Using the RM-4339 with the PXIe-4339

This section contains information on using the NI RM-4339 with the PXIe-4339. Refer to

SC Express Smart Accessory Connections for specific terminal connections.

Excitation

The RM-4339 has a connection for an external excitation source. Each channel can also use it's own channel-specific internal excitation. Use the AI.Excit.Src attribute/property to set the excitation connection.

Quarter-Bridge Completion

The RM-4339 includes three internal resistors you can use for quarter-bridge completion. These resistors are 120 Ω, 350 Ω, and 1 kΩ. Use the AI.Bridge.NomResistance attribute/property to set quarter-bridge completion.

Shunt Calibration

The RM-4339 includes two independent shunt calibration resistors, named A and B. You can select which resistor to use using the AI.Bridge.ShuntCal.Select attribute/property.

You can use an internal or external resistor for shunt resistor A. Use the AI.Bridge.ShuntCal.ShuntCalASrc attribute/property to set this. Select Built-In to use the internal 100 kΩ resistor, or User Provided to use an external resistor.

Shunt resistor B is an internal resistor that you can use only with quarter-bridge completion. You can use a 50 kΩ or 100 kΩ resistor by setting the AI.Bridge.ShuntCal.ShuntCalBResistance attribute/property.

> ✏️ **Note**
>
> The DAQmx Perform Shunt Calibration function/VI and the Strain Gage Calibration Wizard in MAX do not support AI.Bridge.ShuntCal.ShuntCalASrc or AI.Bridge.ShuntCal.ShuntCalBResistance.

Related concepts:

- SC Express Smart Accessory Connections

X Series Device Groupings

X Series Multiplexed Sampling Devices:

- NI 6320
- NI 6321
- NI 6323
- NI 6341
- NI 6343
- NI 6345
- NI 6351
- NI 6353
- NI 6355
- NI 6361
- NI 6363
- NI 6365
- NI 6375

X Series Simultaneous Sampling Devices:
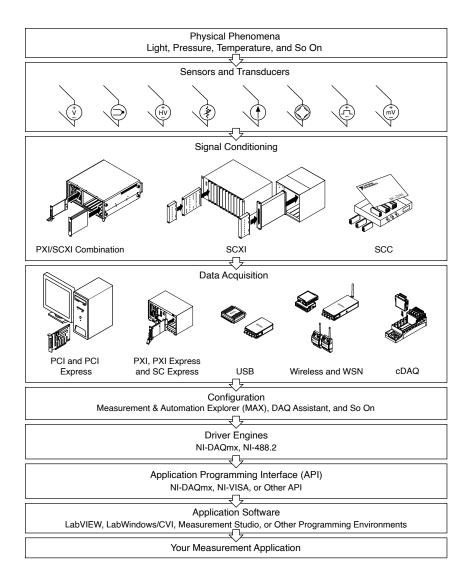
- NI 6346
- NI 6349
- NI 6356
- NI 6358
- NI 6366
- NI 6368
- NI 6374
- NI 6376
- NI 6378
- NI 6386
- NI 6396

Measurement Fundamentals

Measurement Fundamentals covers API-independent information that might help you as you develop applications. Topics include an explanation of different signal types, sensors commonly used with measurement devices, signal conditioning, and control fundamentals.

Measurement System Overview—Hardware and NI-DAQmx

The following figure depicts the measurement system overview, showing the path of real-world physical phenomena to your measurement application.
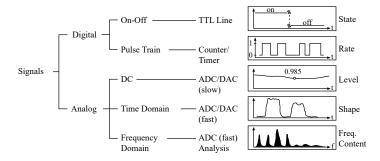
Sensors and transducers detect physical phenomena. Signal conditioning components condition physical phenomena so that the measurement device can receive the data. The computer receives the data through the measurement device. Software controls the measurement system, telling the measurement device when and from which channels to acquire or generate data. Software also takes the raw data, analyzes it, and presents it in a form you can understand, such as a graph, chart, or file for a report.

NI measurement devices and application software are packaged with NI-DAQmx driver software to program all the features of your NI measurement device such as configuring, acquiring, and generating data from and sending data to NI measurement devices. Using NI-DAQmx saves you from having to write these programs yourself. Application software, such as LabVIEW, LabWindows™/CVI™, and Measurement Studio, sends the commands to the driver, such as acquire and return a thermocouple reading, and then displays and analyzes the data acquired.

You can use the NI-DAQmx driver from NI application software or from any programming environment that supports calling dynamic link libraries (DLLs) through ANSI C interfaces. Regardless of the programming environment, your DAQ application uses NI-DAQmx, as shown in the figure.

Signal Types

A signal is classified as analog or digital by the way it conveys information. A digital (or binary) signal has only two possible discrete levels—high level (on) or low level (off). An analog signal, on the other hand, contains information in the continuous variation of the signal with respect to time. A breakdown of the main signal types is shown in the following figure.

Analog Connection Considerations

To measure analog signals, you need to know the signal source—grounded or floating. You also must consider the measurement system—differential, referenced single-ended, or nonreferenced single-ended.

Related concepts:

- Grounded Signal Sources
- Floating Signal Sources
- Differential Measurement System
- Referenced and Nonreferenced Single-Ended Measurement Systems

Connecting Analog Input Signals
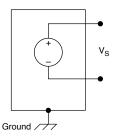
Signal connections vary depending on your device, connector block, and signal

conditioning module. The DAQ Assistant contains connection diagrams that show terminal connections for all common analog input measurements, such as measuring strain, temperature, current, voltage, and so on. Refer to **Viewing Connection Diagrams** in the **DAQ Assistant Help** for additional information.

For terminals specific to your device, refer to your device documentation.

Floating Signal Sources

In a floating source, the voltage signal is not connected to any absolute reference or any common ground, such as earth or building ground as shown in the following figure.

Floating signal sources are also called nonreferenced signal sources. Some common examples of floating signal sources are batteries, thermocouples, transformers, and isolation amplifiers. Notice in the figure that neither terminal of the source is connected to the electrical outlet ground, so each terminal is independent of the system ground.

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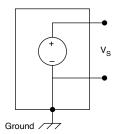
You can measure floating signal sources with both differential and single-ended measurement systems. In the case of the differential measurement system, however, make sure the common-mode voltage level of the signal with respect to the measurement system ground remains in the common-mode input range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

Related concepts:

- [Differential Measurement System](#)
- [Referenced and Nonreferenced Single-Ended Measurement Systems](#)

Grounded Signal Sources

A grounded source is one in which the voltage signals are referenced to a system ground, such as earth or building ground, as shown in the following figure.

Ground

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 the 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.

Measuring Grounded Signal Sources

A grounded signal source is best measured with a differential or an NRSE measurement system. If you use an RSE measurement system with a grounded source, the result is typically a noisy measurement system often showing power-line frequency (60 Hz) components in the readings. Ground-loop introduced noise can have both AC and DC components, introducing offset errors and noise in the measurements. The potential difference between the two grounds causes a current to flow in the interconnection. This current is called ground-loop current.

However, you can still use an RSE measurement 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 the signal to a ground-referenced measurement system because the signal source can be short-circuited to ground, which can damage the signal source.

Related concepts:

- Differential Measurement System
- Referenced and Nonreferenced Single-Ended Measurement Systems

Measurement System Types and Signal Sources

The type of input signal source (grounded or floating) and the configuration of the measurement system (differential, single-ended, pseudodifferential) determine how you connect signals to measurement devices.

The following table provides an application-independent summary of analog input connections.

Input	Signal Source Type	
	Floating Signal Source (Not Connected to Building Ground)	Grounded Signal Source
	Examples: Ungrounded thermocouples, signal conditioning with isolated outputs, battery devices	Example: Instruments with nonisolated outputs
Differential (DIFF)		

Input	Signal Source Type	
	Floating Signal Source (Not Connected to Building Ground)	Grounded Signal Source
	Examples: Ungrounded thermocouples, signal conditioning with isolated outputs, battery devices	Example: Instruments with nonisolated outputs
Ground Referenced Single-Ended (RSE) Note: AI GND is shared as a reference for all RSE channels	V_s, AI, AI GND	NOT RECOMMENDED V_1, AI, $+ V_g -$, AI GND Ground-loop losses, V_g, are added to measured signal.
Nonreferenced Single-Ended (NRSE) Note: AI SENSE is shared as a reference for all NRSE channels	V_s, AI, AI SENSE, R_{ext}, AI GND, AI GND	V_s, AI, AI SENSE, AI GND
Pseudodifferential	V_s, AI +, AI −, Z, AI GND	V_s, AI +, AI −, Z, AI GND

R_{ext} is the external bias resistor that you add.
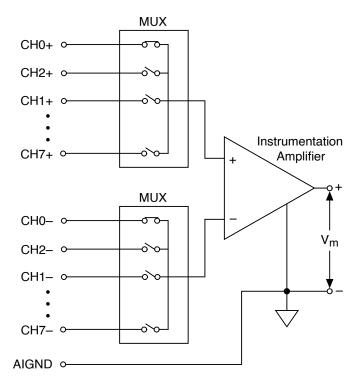
Related concepts:

- Grounded Signal Sources
- Floating Signal Sources
- Differential Measurement System
- Referenced and Nonreferenced Single-Ended Measurement Systems
- Pseudodifferential Measurement System

Differential Measurement System

A differential measurement system has neither of its inputs tied to a fixed reference, such as earth or building ground. A differential measurement system is similar to a floating signal source in that the measurement is made with respect to a floating ground that is different from the measurement system ground. Hand-held, battery-powered instruments and DAQ devices with instrumentation amplifiers are examples of differential measurement systems.

The following figure shows an implementation of an 8-channel differential measurement system used in a typical NI device. Analog multiplexers are used in the signal path to increase the number of measurement channels when there is only a single instrumentation amplifier. For this device, the pin labeled AIGND, the analog input ground, is the measurement system ground.

Your signal source—floating or grounded—helps determine if you should use a differential measurement system.

Related concepts:

- Floating Signal Sources
- Grounded Signal Sources

Rejecting Common-Mode Voltages

An ideal differential measurement system responds only to the potential difference between its two terminals—the positive (+) and negative (–) inputs. Any voltage measured with respect to the instrumentation amplifier ground that is present at both amplifier inputs is referred to as a common-mode voltage. Common-mode voltage is completely rejected (not measured) by an ideal differential measurement system. This capability is useful in rejection of noise, because unwanted noise is often introduced as common-mode voltage in the circuit making up the cabling system.

Real-world devices have several factors, described by parameters such as common-mode voltage range and common-mode rejection ratio (CMRR), that limit the ability to reject the common-mode voltage.

Common-Mode Voltage

The common-mode voltage range limits the allowable voltage swing 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. Common-mode voltage (Vcm) is defined using the following formula:

- Vcm = (V+ + V–) / 2
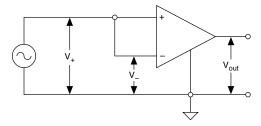
where V+ is the voltage at the noninverting terminal of the measurement system with respect to the measurement system ground, and V– is the voltage at the inverting terminal of the measurement system with respect to the measurement system ground.

CMRR

CMRR measures the ability of a differential measurement system to reject the common-mode voltage signal. For instance, if you are measuring a thermocouple in a noisy environment, the noise from the environment appears on both input leads. Therefore, this noise is a common mode voltage signal that is rejected by an amount equal to the CMRR of the instrument. Most DAQ devices specify the CMRR up to 60 Hz, the power line frequency. CMRR in decibels (dB) is defined using the following formula:
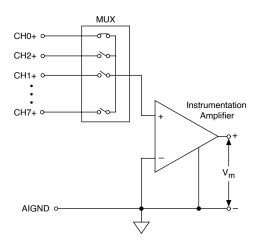
- CMRR(dB) = 20 log (Differential Gain/Common-Mode Gain)

A simple circuit is shown in the following figure. In this circuit, CMRR in decibels is measured as $20 \log V_{out}/V_{cm}$, where $V_{cm} = (V+ + V-)/2$.
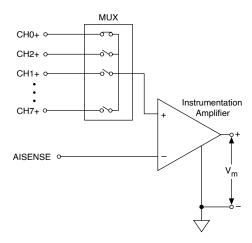
Referenced and Nonreferenced Single-Ended Measurement Systems

Referenced and nonreferenced single-ended measurement systems are similar to grounded sources in that the measurement is made with respect to ground. A referenced single-ended (RSE) measurement system measures voltage with respect to the ground, AIGND in the figure, which is directly connected to the measurement system ground. The following figure shows an 8-channel referenced single-ended measurement system.

DAQ devices often use a variant of the referenced single-ended measurement technique, known as nonreferenced single-ended (NRSE). The following figure shows an NRSE system.
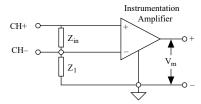
In an NRSE measurement system, all measurements are still made with respect to a single-node analog input sense (AISENSE), but the potential at this node can vary with respect to the measurement system ground. The previous figure illustrates that a single-channel NRSE measurement system is the same as a single-channel differential measurement system.

Pseudodifferential Measurement System

A pseudodifferential measurement system combines some characteristics of a differential input channel and a referenced single-ended (RSE) input channel. Like a differential input channel, a pseudodifferential measurement system exposes both the positive and negative sides of the channel. You connect the positive and negative inputs to the respective outputs of the unit under test. The negative input is tied to system ground through a relatively small impedance (designated as Z_1 in the diagram below). The impedance between the negative input and ground may include both resistive and capacitive components. The positive and negative sides of the input channel are separated by a larger impedance (designated by Z_{in}).
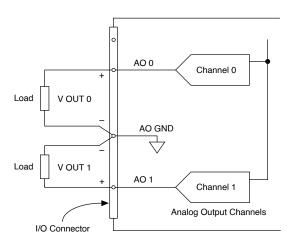
Pseudodifferential input configurations are common in simultaneous sampling and dynamic signal acquisition (DSA) devices that do not employ a multiplexed signal architecture. A pseudodifferential system is well-suited for measuring the output of floating or isolated devices under test such as battery-powered instruments or most accelerometers. The pseudodifferential setup can also be used to measure referenced signals if the signal reference potential does not differ greatly from the ground potential of the measurement device. However, ground loops may pose an issue if the potential of the negative leg of the signal differs significantly from chassis ground. In general, a differential input offers a better common-mode rejection ratio (CMRR) than

a pseudodifferential input.

Connecting Analog Output Signals

Signal connections vary depending on your device, connector block, and signal conditioning module. The following figure shows how to make analog output connections for a typical NI device.

For terminals specific to your device, refer to your device documentation.

Sampling Considerations

When sampling a signal it is important to consider device range, input limits, sampling rate, resolution, and code width.

Related concepts:

- Device Range
- Input Limits (Maximum and Minimum Values)
- Sampling Rate
- Resolution

- [Calculating the Smallest Detectable Change—Code Width](#)

Device Range

Device range refers to the minimum and maximum analog signal levels that the ADC can digitize. Many measurement devices can select from several ranges by changing from unipolar mode to bipolar mode or by selecting from multiple gains, allowing the ADC to take full advantage of its resolution to digitize the signal.
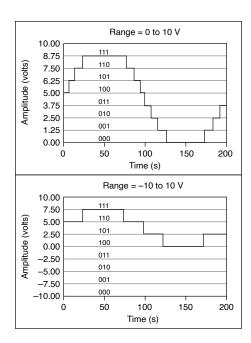
Related concepts:

- [Input Limits (Maximum and Minimum Values)](#)
- [Gain for DSA Devices](#)

Unipolar and Bipolar Modes

Unipolar mode means that a device only supports a range of 0 V to +X V. Bipolar mode means that a device supports a range of -X V to +X V. Some devices support only one mode or the other, while other devices can change from unipolar mode to bipolar mode.

Devices that can change from unipolar to bipolar mode are able to select the mode that best fits the signal to measure. The first chart of the following figure illustrates unipolar mode for a 3-bit ADC. The ADC has eight digital divisions in the range from 0 to 10 V. In bipolar mode, the range is -10.00 to 10.00 V, as shown in the second chart. The same ADC now separates a 20 V range into eight divisions. The smallest detectable difference in voltage increases from 1.25 to 2.50 V, and you now have a much less accurate representation of the signal. The device selects the best mode available based on the input limits you specify when you create a virtual channel.

Gain Adjustment

If a device has multiple gains, it multiplies an input signal by one of the gains to make the signal take up more of the full device range. This essentially gives the device multiple ranges it can select from. For example, a device with an overall range of -10 V to 10 V and possible gains of 1, 2, and 4 can select between ranges of -10 V to 10 V, -5 V to 5 V, and -2.5 V to 2.5 V. The device selects the best gain available according to the input limits you specify when you create a virtual channel.

> ✏️ **Note** Gain works differently for DSA devices

Input Limits (Maximum and Minimum Values)

Input limits are the maximum and minimum values you expect to measure, after any scaling, including custom scaling.

Input limits are sometimes confused with device range. Device range refers only to the input range of a particular device.

If you set your minimum to 0 and your maximum to 7, and your device only has a 0-5 and a 0-10 device range, the device range will be coerced to 0-10 by NI-DAQmx. This also applies when using a custom scale.

The maximum and minimum input limit values will match the units of measurement of the task. For instance, the device range for a DAQ device might be 0 to 10 V, but that device might be used with a temperature sensor that outputs 100 mV for every 1 °C. The input limits in that case could be 0 to 100, with 10 V corresponding to 100 °C. For an analog current input task, the units would be amps.

Input limits in a smaller range can improve the precision of your measurement. If your device has multiple input ranges, you can strategically choose the minimum and maximum values to detect smaller differences in your signal of interest. Consider a device that has a 0-10 V and a 0-5 V range. In the previous temperature sensor example, if you knew that the temperature would never be higher than 50 °C, you could choose a minimum value of 0 and a maximum value of 50. The device can then detect smaller differences in temperature because it is digitizing a voltage between 0 and 5 V, rather than 0 and 10 V.

Related concepts:

- [Device Range](#)

Sampling Rate

One of the most important parameters of an analog input or output system is the rate at which the measurement device samples an incoming signal or generates the output signal. The **sampling rate**, which is called the scan rate in Traditional NI-DAQ (Legacy), is the speed at which a device acquires or generates a sample on each channel. 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. Generating a 1 Hz signal using 1000 points per cycle at 1000 S/s produces a much finer representation than using 10 points per cycle at a sample rate of 10 S/s.
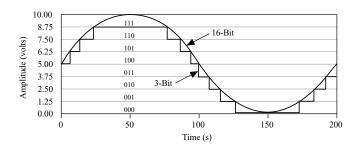
Sampling too slowly results in a poor representation of the analog signal. Undersampling causes the signal to appear as if it has a different frequency than it actually does. This misrepresentation of a signal is called **aliasing**.

Resolution

Resolution is the smallest amount of input signal change that a device or sensor can detect. The number of bits used to represent an analog signal determines the

resolution of the ADC. You can compare the resolution on a measurement device to the marks on a ruler. The more marks you have, the more precise your measurements. Similarly, the higher the resolution, the higher the number of divisions into which your 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 8 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 figure shows a sine wave digital image as obtained by a 3-bit ADC. Clearly, 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. By increasing the resolution to 16 bits, however, the number of divisions of the ADC increases from 8 to 65,536 (2^{16}). The ADC now can obtain an extremely accurate representation of the analog signal.

Calculating the Smallest Detectable Change—Code Width

The resolution and device range of a measurement device determine the smallest detectable change, called the **code width**, in the input signal. The smaller your code width, the more accurate your measurements are.

You can calculate the code width using the following formula:

- code width = device range/ $2^{resolution}$

For example, a 12-bit measurement device with a 0 to 10 V range detects a 2.4 mV change, while the same device with a -10 to 10 V input range detects only a change of 4.8 mV:

- device range/ $2^{resolution}$ = $10/2^{12}$ = 2.4 mV

- device range/ $2^{resolution} = 20/2^{12} = 4.8$ mV

A high-resolution A/D converter (ADC) provides a smaller code width given the preceding device voltage ranges.

- device range/ $2^{resolution} = 10/2^{16} = 0.15$ mV
- device range/ $2^{resolution} = 20/2^{16} = 0.3$ mV

The following table shows how the code width of a 12-bit measurement device varies by device range. The device selects the best possible range based on the input limits you specify when you create a virtual channel. Select input limits that accurately reflect the signal you want to measure in order to achieve the smallest possible code width. NI-DAQmx coerces those input limits to fit the selected device range.

Overall Device Range	Possible Device Ranges with Gain Adjustment	Precision[65]
0 to 10 V	0 to 10 V 0 to 5 V 0 to 2.5 V 0 to 1.25 V 0 to 1 V 0 to 0.1 V 0 to 20 mV	2.44 mV 1.22 mV 610 µV 305 µV 244 µV 24.4 µV 4.88 µV
−5 to 5 V	-5 to 5 V -2.5 to 2.5 V -1.25 to 1.25 V -0.625 to 0.625 V -0.5 to 0.5 V -50 to 50 mV -10 to 10 mV	2.44 mV 1.22 mV 610 µV 305 µV 244 µV 24.4 µV 4.88 µV
-10 to 10 V	-10 to 10 V -5 to 5 V	4.88 mV 2.44 mV

65. The value of 1 Least Significant Bit (LSB) of the 12-bit ADC. In other words, the voltage increment corresponding to a change of 1 count in the ADC 12-bit count.

Overall Device Range	Possible Device Ranges with Gain Adjustment	Precision
	• -2.5 to 2.5 V • -1.25 to 1.25 V • -1 to 1 V • -0.1 to 0.1 V • -20 to 20 mV	• 1.22 mV • 610 µV • 488 µV • 48.8 µV • 9.76 µV

> ✎ **Note** The NI 4472 is a 24-bit device with a range of -10 V to 10 V. However, one bit is reserved, leaving an effective resolution of 23 bits. Thus, the code width is $20/2^{23}$ = 2.38 µV.
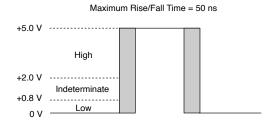
Related concepts:

- Resolution
- Device Range
- Input Limits (Maximum and Minimum Values)
- Input Limit Coercion

Digital Signals

A digital signal has two discrete levels—a high and a low level. One example of a digital signal is a transistor-transistor logic (TTL) compatible signal. A TTL-compatible signal has the following characteristics:
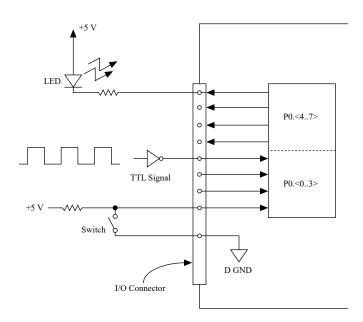
- 0 V to 0.8 V = logic low
- 2 V to 5 V = logic high
- Maximum rise/fall time = 50 ns

Digital devices can monitor the state of the pulse and can transition the pulse from one state to another. A counter can also monitor the state as well as detect rising edges, a transition from logic low to logic high, and falling edges, a transition from logic high to logic low. Counters are used commonly to count edges and for time measurements, such as measuring digital frequency or the period of a signal.

Connecting Digital I/O Signals

The number of digital lines varies from device to device. The following figure shows signal connections for three typical DIO applications.

The figure shows PO <0..3> configured for digital input and PO <4..7> configured for digital output. Digital input applications include receiving TTL signals and sensing external device states such as the state of a switch. Digital output applications include sending TTL signals and driving external devices such as the LED shown in the figure.

Counters

Counters measure and generate digital signals. Counters are used commonly to count edges and for time measurements, such as measuring digital frequency or the period of a signal. The signal connections required for counters vary depending on the device and your application.

Related concepts:

- <u>Connecting Counter Signals</u>

Digital Logic States

Test engineers can choose from a number of different digital I/O instruments with a range of features for communication and test applications. Beyond the basic capabilities of driving a digital pattern of 1s and 0s, digital instruments often support waveforms that can include some or all of the logic states shown in the following table.

	Logic State	Drive Data	Expected Response
Drive States	0	Logic Low	Don't Care
	1	Logic High	Don't Care
	Z	Disable	Don't Care
Compare States	L	Disable	Logic Low
	H	Disable	Logic High
	X	Disable	Don't Care

The six logic states control the voltage driver and, if supported, the compare engine of the digital tester (such as a DAQ device) on a per clock cycle basis. Drive states specify what stimulus data the digital tester drives on a particular channel or when to disable the voltage driver (referred to as the tristate or high-impedence state). Compare states indicate the expected response from the device under test. These six logic states make it possible to perform bidirectional communication and real-time hardware comparison of acquired response data.
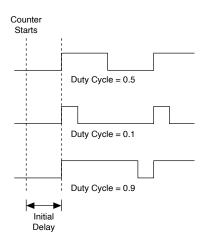
Duty Cycle

The duty cycle is a characteristic of a pulse. Use the following equation to calculate the duty cycle of a pulse whose high time and low time are unequal:

- Duty Cycle = High Time/Pulse Period

where Pulse Period is high time plus low time.

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

Signal Analysis

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

Filtering and windowing are two signal analysis techniques.

Related concepts:

- Filtering
- Windowing

Filtering

Filtering is one of the most commonly used signal processing techniques. Signal conditioning systems can filter unwanted signals or noise from the signal you are measuring. Use a noise filter on low-rate, or slowly changing, signals, such as temperature, to eliminate higher frequency signals that can reduce signal accuracy. A common use of a filter is to eliminate the noise from a 50 or 60 Hz AC power line. A lowpass filter of 4 Hz removes the 50 or 60 Hz AC noise from signals sampled at low

rates. A lowpass filter eliminates all signal frequency components above the cutoff frequency. Many signal conditioning modules have lowpass filters that have software-selectable cutoff frequencies from 10 Hz to 25 kHz.

Windowing

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

Spectral Leakage

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

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

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

Window	Signal Type and Description	Applications
Rectangular (no window)	Transient signals that are shorter than the length of the window; truncates a window to within a finite time interval	Order tracking, system analysis (frequency response measurements) with pseudorandom excitation, separation of two tones with frequencies very close to each other but with

Window	Signal Type and Description	Applications
		almost equal amplitudes
Triangle	Window that is the shape of a triangle	General-purpose applications
Hanning	Transient signals that are longer than the length of the window	Often used in speech signal processing
Hamming	Transient signals that are longer than the length of the window; a modified version of the Hanning window that is discontinuous at the edges	Often used in speech signal processing
Blackman	Transient signals; similar to Hanning and Hamming windows but adds one additional cosine term to reduce ripple	General-purpose applications
Flat Top	Has the best amplitude accuracy of all the windows but limits frequency selectivity	Accurate, single-tone amplitude measurements with no nearby frequency components

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

Signal Conditioning
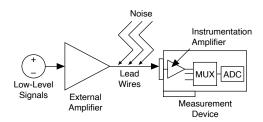
Sensors can generate electrical signals to measure physical phenomena, such as temperature, force, sound, or light. To measure signals from transducers, you must convert them into a form that a DAQ device can accept. For example, the output voltage of most thermocouples is very small and susceptible to noise. Therefore, you may need to amplify or filter the thermocouple output before digitizing it.

The manipulation of signals to prepare them for digitizing is called ***signal conditioning***. Common types of signal conditioning include the following.

Amplification

Amplification is a type of signal conditioning that improves accuracy in the resulting digitized signal by increasing signal amplitude relative to noise.

For the highest possible accuracy, amplify the signal so the maximum voltage swing equals the maximum input range of the ADC, or digitizer. Your system should amplify low-level signals at the measurement device located nearest the signal source, as shown in the following figure.

Tip Use shielded cables or a twisted pair of cables. By minimizing wire length, you can minimize noise that the lead wires pick up. Keep signal wires away from AC power cables and monitors to reduce 50 or 60 Hz noise.

If you amplify the signal at the measurement device, the signal is measured and digitized with noise that may have entered the lead wires. However, if you amplify the signal close to the signal source with an SCXI module, noise has less impact on the measured signal.

Linearization

Linearization is a type of signal conditioning in which software linearizes the voltage levels from transducers, so the voltages can be scaled to measure physical phenomena. For example, a change in voltage of 10 mV for a thermocouple usually does not reflect a change of 10 degrees. However, with linearization in software or hardware, the thermocouple can be scaled to the appropriate temperature in your application. Most transducers have linearization tables that describe scaling the transducer.

Transducer Excitation

Signal conditioning systems can generate excitation for some transducers. Strain gages and RTDs require external voltage and current, respectively, to excite their circuitry into measuring physical phenomena. This type of excitation is similar to the power a radio needs to receive and decode audio signals. Several measurement devices provide the necessary excitation for transducers. Consult your device documentation to see if your device can generate excitation.

Isolation

Signals often can exceed the limits that a measurement device can handle. Trying to measure a signal that is too large for the measurement device can damage the device or you. To keep you and your device safe from large voltages, you can apply a signal conditioning technique called *isolation*. The signal conditioning hardware attenuates high common mode voltages and extracts a signal that measurement devices can handle. Isolation also ensures that differences in ground potentials do not affect your device.

Common Sensors

Depending on your application, you may use several different kinds of sensors. Some commonly used ones are strain gages, thermocouples, thermistors, angular encoders, linear encoders, bridge-based sensors, and resistance temperature detectors (RTDs).
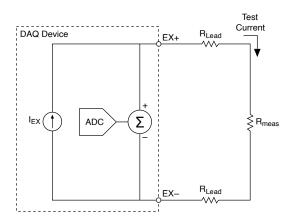
Related concepts:

- Strain Gages
- Thermocouples
- Thermistors
- Quadrature Encoders
- Two-Pulse Encoders
- Bridge-Based Sensors
- Resistance Temperature Detectors (RTDs)

2-Wire Resistance
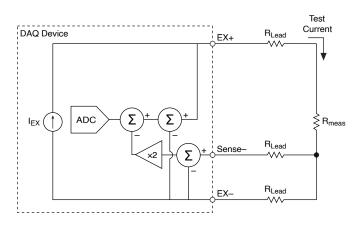
Resistance measurements in the range above 100 Ω are generally made using the 2-wire method shown in the following figure. The excitation current flows through the leads and the unknown resistance, R_{meas}. Your device measures the voltage across the resistance through the same set of leads and computes the resistance accordingly.

Errors in the 2-wire measurements are introduced by the lead resistance, R_{Lead}, when measuring lower resistances. Because there is a voltage drop across the lead resistance equal to $I \times R_{Lead}$, the voltage measured by your device is not exactly the same as the voltage across the resistance, R_{meas}. Because typical lead resistances lie in the range of 0.01–1 Ω, accurate 2-wire resistance measurements are very difficult to obtain if R_{meas} is below 100 Ω.

3-Wire Resistance

Use the 3-wire resistance method, as shown in the following figure, to measure resistance on resistors that have three lead wires.
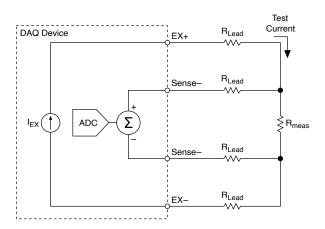
The 3-wire method uses three test leads, one pair for the excitation current (EX+, EX-), and a third wire (Sense-) to compensate for the lead wire resistances. The third wire measures the voltage developed over the lead resistance in the EX- leg of the excitation current path. By subtracting its value from the overall differential signal, the device can compensate for parasitic lead resistances in the EX+ leg. However, this would only compensate for lead resistance in the EX+ leg and not in the EX- leg. To compensate for lead resistance both in the EX- leg and in the EX+ leg, the device approximates the EX+ leg by assuming the voltage is the same as in the EX- leg. Thus the voltage between Sense- and EX- is multiplied by two before being subtracted from the overall differential signal. This method works well when lead resistances in the EX+ leg match resistances in the EX- leg.

Some legacy devices do not provide compensation. In that case, you need to specify the lead-wire resistance so that it can be subtracted in software.

4-Wire Resistance
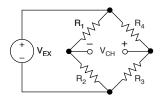
Use the 4-wire resistance method, as shown in the following figure, to measure resistances of less than 100 Ω. The 4-wire method is more accurate than the 2-wire method.

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

Bridge-Based Sensors

Bridge-based sensors operate by correlating a physical phenomena, such as strain, temperature, or force, to a change in resistance in one or more legs of a Wheatstone bridge. The general Wheatstone bridge, shown in the following figure, is a network of four resistive legs with an excitation voltage, V_{EX}, that is applied across the bridge. One or more of these legs can be active sensing elements.

The Wheatstone bridge is the electrical equivalent of two parallel voltage divider circuits. R1 and R2 compose one voltage divider circuit, and R4 and R3 compose the second voltage divider circuit. You measure the output of a Wheatstone bridge between the middle nodes of the two voltage dividers.

A physical phenomena, such as a temperature shift or a change in strain applied to a specimen, changes the resistance of the sensing elements in the Wheatstone bridge. You can use the Wheatstone bridge configuration to help measure the small variations in resistance that the sensing elements produce corresponding to a physical change in the specimen.

Bridge Measurement Types

NI-DAQmx provides several measurement types for taking measurements from a bridge-based sensor.

- **Strain gage**—Use the strain gage measurement type for performing strain measurements.
- **Force, pressure, and torque**—Use these measurement types for performing bridge-based force, pressure, or torque measurements on supported devices.

NI-DAQmx provides additional measurement types from which you read data in electrical units. Use a custom scale or write scaling code to convert the electrical units

to physical units.

- **Bridge (V/V)**—Channels using the bridge V/V measurement type return a voltage ratio instead of physical units. Use this measurement type on supported devices for other bridge-based sensors, for sensors that scale data to physical units NI-DAQmx does not support, or to measure torsional strain.
- **Custom voltage with excitation**—Use this measurement type for bridge-based measurements on devices that do not support the strain gage, force, pressure, torque, or bridge (V/V) measurement types. Specify to use excitation for scaling to acquire a voltage ratio instead of a voltage. On ratiometric devices, you must specify to use excitation for scaling because those devices can only acquire a voltage ratio.
- **Voltage**—Use this measurement type with bridge-based sensors that include an internal amplifier and output voltage.
- **Current**—Use this measurement type with bridge-based sensors that include an internal amplifier and output current.

Related concepts:

- [Bridge Measurement Type Support](#)

Bridge Sensor Scaling

Measurements from a bridge-based sensor are based on a ratio of measured voltage to excitation voltage. NI-DAQmx uses the following equation to calculate that ratio:

$$V_R = \frac{(V - V_{IB}) \times G}{(V_{EX})}$$

where V_R is the voltage ratio; V is the voltage output from the bridge; V_{IB} is the initial bridge voltage, as determined by offset nulling; V_{EX} is the excitation voltage supplied to the bridge; and G is the gain adjustment from shunt calibration.

Ratiometric devices divide the voltage output from the bridge by the excitation voltage in hardware. Therefore, V/V_{EX} must be within the device range of the device. On voltage devices, the voltage output from the bridge must be within the device range. The initial bridge voltage and gain adjustment affect the association between device range and input limits. For example, on a device that can measure ±5 V, an initial

bridge voltage of 1 V means that the minimum and maximum input limits must correspond to -6 V to 4 V.

NI-DAQmx uses various methods to scale that voltage ratio to physical units, depending on the bridge configuration for strain measurements or on the scaling type that best matches the specifications provided by the sensor manufacturer for other bridge-based sensors NI-DAQmx supports.

Related concepts:

- Offset Nulling (Bridge Balancing)
- Shunt Calibration (Gain Adjustment)
- Device Range
- Input Limits (Maximum and Minimum Values)
- Strain Gage Bridge Configurations
- Bridge Scaling Types

Bridge Configurations

There are three types of bridge configurations: quarter-bridge, half-bridge, and full-bridge. The number of active element legs in the Wheatstone bridge determines the kind of bridge configuration.

Configuration	Number of Active Elements
Quarter-bridge	1
Half-bridge	2
Full-bridge	4

Strain gages use variations of these bridge configurations.

Related concepts:

- Strain Gage Bridge Configurations

Signal Conditioning Requirements for Bridge-Based Sensors

Common signal conditioning requirements for bridge-based sensors include:

- Bridge completion
- Bridge excitation and remote sensing
- Signal amplification

In addition, you should calibrate your sensor periodically to account for changes in the physical characteristics of the sensor and in the material the sensor is measuring, to account for variations in the lead wire resistance, and to compensate for imperfections in the measurement system. Calibrating bridge-based sensors usually involves two steps: offset nulling, or bridge balancing, and shunt calibration, or gain adjustment.

Related concepts:

- Bridge Completion
- Bridge Excitation
- Signal Amplification
- Offset Nulling (Bridge Balancing)
- Shunt Calibration (Gain Adjustment)

Bridge Completion

Unless you are using a full-bridge sensor, you must complete the bridge with reference resistors. Therefore, signal conditioners for bridge-based sensors typically provide half-bridge completion networks consisting of two high-precision reference resistors. The nominal resistance of the completion resistors is less important than how well the two resistors match. Ideally, the resistors match well 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. However, using completion resistors that are too large can result in increased noise and errors due to bias currents.

Signal Amplification

The output of bridge-based sensors is relatively small. For example, most strain gage bridges and strain-based transducers output less than 10 mV/V, or 10 millivolts of output per volt of excitation voltage. Therefore, signal conditioners for bridge-based sensors usually include amplifiers to boost the signal level, to increase measurement

resolution, and to improve signal-to-noise ratios. For example, SCXI signal conditioning modules include configurable gain amplifiers with gains up to 2,000. Other devices support multiple device ranges.

Related concepts:

- Device Range

Bridge Excitation

Bridge-based sensors require a constant voltage to power the bridge. Bridge signal conditioners typically include a voltage source. While there is no standard voltage level that is recognized industry wide, excitation voltage levels of around 3 V and 10 V are common.

Excitation sources can suffer from stability and accuracy issues. To compensate, ratiometric devices constantly measure the actual excitation voltage and use it, rather than an intended excitation value, when scaling data.

Related concepts:

- Bridge Sensor Scaling

Remote Sensing

If the bridge circuit is located away from the signal conditioner and excitation source, a possible source of error is voltage drops caused by resistance in the wires that connect 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 to 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. You then can use the measured excitation voltage in the voltage-to-strain conversion to compensate for lead losses.

Bridge-Based Sensor Calibration

Calibrate bridge-based sensors periodically to account for changes in the physical characteristics of the sensor and in the material the sensor is measuring, to account for variations in the lead wire resistance, and to compensate for imperfections in the measurement system. Calibrating bridge-based sensors usually involves two steps: offset nulling, or bridge balancing, and shunt calibration, or gain adjustment.

Related concepts:

- [Offset Nulling (Bridge Balancing)](#)
- [Shunt Calibration (Gain Adjustment)](#)

Offset Nulling (Bridge Balancing)

When you install a bridge-based sensor, the bridge probably will not output exactly 0 V when not under load. Slight variations in resistance among the bridge legs generate some nonzero initial offset voltage. Use the DAQmx Perform Bridge Offset Nulling Calibration VI/function or the DAQ Assistant to perform an offset nulling calibration, which performs bridge balancing in a few different ways. Refer to the device documentation to determine the offset nulling methods your device provides.

Related concepts:

- [Bridge Sensor Scaling](#)

Software Compensation (Initial Bridge Voltage)

This method of bridge balancing compensates for the initial voltage in software. With this method, NI-DAQmx measures the bridge while not under load. NI-DAQmx then uses this measurement as the initial bridge voltage when scaling readings from the bridge. This method is simple, fast, and requires no manual adjustments. The disadvantage of the software compensation method is that the method does not remove the offset of the bridge. 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 bridge balancing method uses an adjustable resistor, or potentiometer, to electrically adjust the output of the bridge to 0 V.

Hardware Nulling Compensation

The third method, like the software compensation method, does not affect the bridge directly. A nulling circuit adds an adjustable DC voltage, positive or negative, to the output of the instrumentation amplifier to compensate for initial bridge offset.

Shunt Calibration (Gain Adjustment)

You can verify the output of a bridge-based measurement system by comparing the measured bridge output with a calculated value if the physical load on the sensor is known. NI-DAQmx can then use the difference (if any) between the calculated and the measured values as a gain adjustment factor for each measurement. You can simulate applying a load to the bridge by connecting a large known resistor in parallel with the bridge. This resistor, called a shunt resistor, offsets the zero voltage of the bridge. Because the value of the shunt resistor is known, you can calculate the physical load corresponding to the voltage drop of the resistor.

Use the DAQmx Perform Shunt Calibration VI/function or the DAQ Assistant to perform a shunt calibration, which sets the gain adjustment for a virtual channel. NI-DAQmx then uses this gain adjustment when scaling readings from the bridge. Some NI products include internal shunt resistors.

Related concepts:

- Bridge Sensor Scaling

Bridge-Based Force, Pressure, and Torque Sensors

Sensors for measuring force, pressure, and torque, such as a load cell, are often based on a Wheatstone bridge. These sensors typically use a full-bridge configuration with a

350 Ω nominal bridge resistance.

Sensor manufacturers often provide a table or polynomial equation to describe how electrical values scale to the physical phenomena the sensor measures. NI-DAQmx provides several scaling types for scaling data according to the specifications provided by the sensor manufacturer.

Related concepts:

- Bridge-Based Sensors
- Bridge Configurations
- Bridge Scaling Types

Bridge Scaling Types

NI-DAQmx provides three methods to scale electrical values (voltage ratios) from a bridge-based sensor to physical units:

- **Two-Point Linear**—You provide two pairs of electrical values and their corresponding physical values. NI-DAQmx uses those values to calculate the slope and y-intercept of a linear equation and uses that equation to scale electrical values to physical values. Measured electrical and physical values can fall outside the range of the values specified for calculating the slope and y-intercept.
- **Table**—You provide a set of electrical values and the corresponding physical values. NI-DAQmx performs linear scaling between each pair of electrical and physical values. The input limits must fall within the smallest and largest physical values.
- **Polynomial**—You provide the forward and reverse coefficients of a polynomial equation. NI-DAQmx uses that equation to scale electrical values to physical values. Use the DAQmx Compute Reverse Polynomial Coefficients VI/function to determine one set of coefficients if you know only the other set.

Datasheets or calibration certificates from sensor manufacturers often include a table of electrical and physical values or a polynomial equation for scaling. If you do not have a table or polynomial equation for your sensor, use two-point linear scaling. Use the rated output of the sensor and the sensor capacity as one pair of electrical and physical values. Use zero for the other pair of electrical and physical values.

> **Note** Strain gages use specific equations for scaling, depending on the bridge configuration.

Related concepts:

- Bridge Sensor Scaling
- Input Limits (Maximum and Minimum Values)
- Strain Gage Bridge Configurations

Strain Gages

You can measure strain with a strain gage, which is a device with electrical resistance that varies in proportion to the amount of strain in the device, and with signal conditioning. When using a strain gage, you bond the strain gage to the device under test, apply force, and measure the strain by detecting changes in resistance (Ω). Strain gages return varying voltages in response to stress or vibrations in materials. Resistance changes in parts of the strain gage to indicate deformation of the material. Strain gages require excitation, generally voltage excitation, and linearization of the voltage measurements.

Strain measurements rarely involve quantities larger than a few microstrain ($\mu\varepsilon$). Therefore, measuring strain requires accurate measurements of very small changes in resistance. For example, if a test specimen undergoes a substantial strain of 500 $\mu\varepsilon$, a strain gage with a gage factor of 2 exhibits a change in electrical resistance of only 2 × (500×10^{-6}) = 0.1%. For 120 Ω, this is a change of only 0.12 Ω.

To measure such small changes in resistance and to compensate for temperature sensitivity, strain gages often use a Wheatstone bridge with a voltage or current excitation source, arranged in one of several bridge configurations. The gage is the collection of all of the active elements of the Wheatstone bridge.

NI-DAQmx supports measuring axial strain, bending strain, or both. While you can use some similar configuration types to measure torsional strain, NI software scaling does not support these configuration types. It is possible to use NI products to measure torsional strain, but to properly scale these configuration types you must use a custom scale with a bridge (V/V) or a custom voltage with excitation channel.

Gage Factor

A fundamental parameter of the strain gage is its sensitivity to strain, expressed quantitatively as the gage factor (GF). Gage factor is the ratio of the fractional change in electrical resistance to the fractional change in length, or strain. The gage factor must be the same for each gage in the bridge.

The gage factor for metallic strain gages is usually around 2. You can obtain the actual gage factor of a particular strain gage from the sensor vendor or sensor documentation.

Nominal Gage Resistance

Nominal gage resistance is the resistance of a strain gage in an unstrained position. You can obtain the nominal gage resistance of a particular gage from the sensor vendor or sensor documentation.

Related concepts:

- Strain Rosette
- Signal Conditioning Requirements for Bridge-Based Sensors
- Bridge-Based Sensors
- Strain Gage Bridge Configurations
- Custom Scales

Strain Gage Bridge Configurations

Connect strain gages in a variation on a generic Wheatstone bridge configuration. These configurations vary based on the placement of strain gages within the bridge; their location and orientation on the material you want to measure; and whether the gages are active sensing gages or dummy gages, used for temperature compensation.
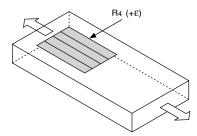
Related concepts:

- Bridge Configurations
- Quarter-Bridge Type I
- Quarter-Bridge Type II
- Half-Bridge Type I
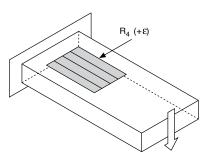
- Half-Bridge Type II
- Full-Bridge Type I
- Full-Bridge Type II
- Full-Bridge Type III

Quarter-Bridge Type I

The following figure shows how to position a strain gage resistor in an axial configuration for the quarter-bridge type I.

The following figure shows how to position a strain gage resistor in a bending configuration for the quarter-bridge type I.
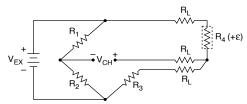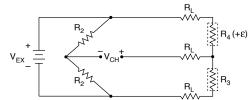

Quarter-bridge type I strain gage configurations have the following characteristics:

- A single active strain gage element mounted in the principle direction of axial or bending strain.
- A passive quarter-bridge completion resistor, known as a dummy resistor, in addition to half-bridge completion.
- Temperature variation decreasing the accuracy of the measurements.
- Sensitivity at 1000 με is ~ 0.5 mV$_{out}$ / V$_{EX}$ input.

Related concepts:

- [Bridge Sensor Scaling](#)

Quarter-Bridge Type I Circuit Diagram

The following symbols apply to the circuit diagram:

- R_1 is the half-bridge completion resistor.
- R_2 is the half-bridge completion resistor.
- R_3 is the quarter-bridge completion resistor, known as a dummy resistor.
- R_4 is the active strain gage element measuring tensile strain ($+\varepsilon$).
- V_{EX} is the excitation voltage.
- R_L is the lead resistance.
- V_{CH} is the measured voltage.

The following equation converts voltage ratios to strain units for quarter-bridge configurations.

$$strain(\varepsilon) = \frac{-4V_r}{GF(1+2V_r)} \cdot \left(1+\frac{R_L}{R_g}\right)$$
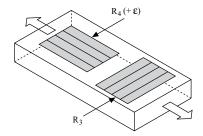
where V_r is the voltage ratio that virtual channels use in the voltage-to-strain conversion equation, GF is the gage factor, R_L is the lead resistance, and R_g is the nominal gage resistance.

Quarter-Bridge Type II

The following figure shows how to position a strain gage resistor in an axial configuration for the quarter-bridge type II.

The following figure shows how to position a strain gage resistor in a bending configuration for the quarter-bridge type II.

Quarter-bridge type II strain gage configurations have the following characteristics:

- One active strain gage element and one passive, temperature-sensing quarter-bridge element, known as a dummy resistor. The active element is mounted in the direction of axial or bending strain. The dummy gage is mounted in close thermal contact with the strain specimen but is not bonded to the specimen, and is usually mounted transverse, or perpendicular, to the principle axis of strain. This configuration is often confused with the half-bridge type I configuration, but in the half-bridge type I configuration, the R_3 element is active and bonded to the strain specimen to measure the effect of Poisson's ratio.
- Completion resistors which provide half-bridge completion.
- Compensation for temperature.
- Sensitivity at 1000 $\mu\varepsilon$ is ~ 0.5 mV$_{out}$/ V$_{EX}$ input.

Related concepts:

- Bridge Sensor Scaling

Quarter-Bridge Type II Circuit Diagram

The following symbols apply to the circuit diagram:

- R_1 is the half-bridge completion resistor.
- R_2 is the half-bridge completion resistor.
- R_3 is the quarter-bridge temperature sensing element, known as a dummy resistor.
- R_4 is the active strain gage element measuring tensile strain ($+\varepsilon$).
- V_{EX} is the excitation voltage.
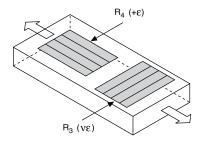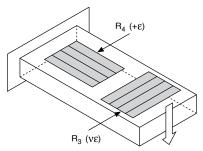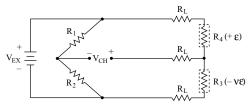- R_L is the lead resistance.
- V_{CH} is the measured voltage.

The following equation converts voltage ratios to strain units for quarter-bridge configurations.

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

where V_r is the voltage ratio that virtual channels use in the voltage-to-strain conversion equation, GF is the gage factor, R_L is the lead resistance, and R_g is the nominal gage resistance.

Half-Bridge Type I

The following figure shows how to position strain gage resistors in an axial configuration for the half-bridge type I.

The following figure shows how to position strain gage resistors in a bending

configuration for the half-bridge type I.

Half-bridge type I strain gage configurations have the following characteristics:

- Two active strain gage elements, one mounted in the direction of axial strain and the other acting as a Poisson gage and mounted transverse, or perpendicular, to the principal axis of strain.
- Completion resistors which provide half-bridge completion.
- Sensitivity to both axial and bending strain.
- Compensation for temperature.
- Compensation for the aggregate effect on the principle strain measurement due to the Poisson's ratio of the material.
- Sensitivity at 1000 $\mu\varepsilon$ is ~ 0.65 mV$_{out}$/ V$_{EX}$ input

Related concepts:

- [Bridge Sensor Scaling](#)

Half-Bridge Type I Circuit Diagram

The following symbols apply to the circuit diagram:

- R_1 is the half-bridge completion resistor.
- R_2 is the half-bridge completion resistor.
- R_3 is the active strain gage element measuring compression due to the Poisson effect ($-\varepsilon$).

- R_4 is the active strain gage element measuring tensile strain $(+\varepsilon)$.
- V_{EX} is the excitation voltage.
- R_L is the lead resistance.
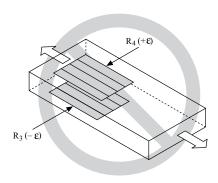- V_{CH} is the measured voltage.

The following equation converts voltage ratios to strain units for half-bridge type I configurations.

$$\text{strain}(\varepsilon) = \frac{-4V_r}{GF[(1+v)-2V_r(v-1)]} \cdot \left(1+\frac{R_L}{R_g}\right)$$
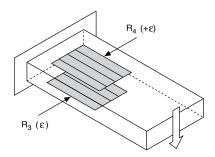
where V_r is the voltage ratio that virtual channels use in the voltage-to-strain conversion equation, GF is the gage factor, v is the Poisson's ratio, R_L is the lead resistance, and R_g is the nominal gage resistance.

Half-Bridge Type II

The half-bridge type II configuration only measures bending strain.

The following figure shows how to position strain gage resistors in a bending configuration for the half-bridge type II.
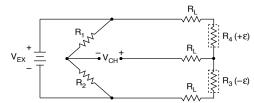
Half-bridge type II strain gage configurations have the following characteristics:

- Two active strain gage elements, one mounted in the direction of axial strain on the top side of the strain specimen and the other mounted in the direction of axial strain on the bottom side.
- Completion resistors which provide half-bridge completion.
- Sensitivity to bending strain.
- Rejection of axial strain.
- Compensation for temperature.
- Sensitivity at 1000 με is ~ 1 mV$_{out}$ / V$_{EX}$ input.

Related concepts:

- [Bridge Sensor Scaling](#)

Half-Bridge Type II Circuit Diagram

The following symbols apply to the circuit diagram:

- R_1 is the half-bridge completion resistor.
- R_2 is the half-bridge completion resistor.
- R_3 is the active strain gage element measuring compressive strain (–ε).
- R_4 is the active strain gage resistor measuring tensile strain (+ε).
- V_{EX} is the excitation voltage.
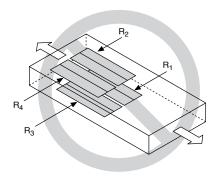- R_L is the lead resistance.
- V_{CH} is the measured voltage.

The following equation converts voltage ratios to strain units for half-bridge type II configurations.

$$\text{Strain(} \epsilon \text{)} = \frac{-2V_r}{GF} \cdot \left(1 + \frac{R_L}{R_g}\right)$$
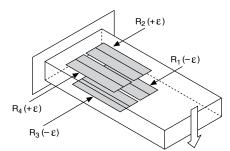
where V_r is the voltage ratio that virtual channels use in the voltage-to-strain conversion equation, GF is the gage factor, R_L is the lead resistance, and R_g is the nominal gage resistance.

Full-Bridge Type I

The full-bridge type I configuration only measures the bending strain.

The following figure shows how to position strain gage resistors in a bending configuration for the full-bridge type I.
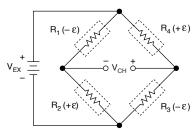
Full-bridge type I strain gage configurations have the following characteristics:

- Four active strain gage elements, two mounted in the direction of bending strain on the top side of the strain specimen and the other two mounted in the direction of bending strain on the bottom side.
- High sensitivity to bending strain.
- Rejection of axial strain.
- Compensation for temperature.
- Compensation for lead resistance.
- Sensitivity at 1000 µε is ~ 2.0 mV_{out} / V_{EX} input.

Related concepts:

- Bridge Sensor Scaling

Full-Bridge Type I Circuit Diagram

The following symbols apply to the circuit diagram:

- R_1 is the active strain gage element measuring compressive strain ($-\varepsilon$).
- R_2 is the active strain gage element measuring tensile strain ($+\varepsilon$).
- R_3 is the active strain gage element measuring compressive strain ($-\varepsilon$).
- R_4 is the active strain gage element measuring tensile strain ($+\varepsilon$).
- V_{EX} is the excitation voltage.
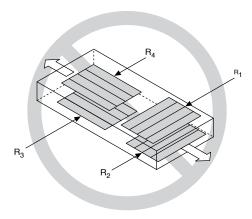- R_L is the lead resistance.
- V_{CH} is the measured voltage.

The following equation converts voltage ratios to strain units for full-bridge type I configurations.

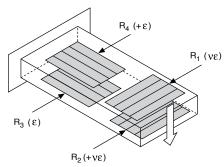$$\text{Strain}(\ \varepsilon) = \frac{-V_r}{GF}$$

where V_r is the voltage ratio that virtual channels use in the voltage-to-strain conversion equation, and GF is the gage factor.

Full-Bridge Type II

The full-bridge type II configuration only measures bending strain.

The following figure shows how to position strain gage elements in a bending configuration for the full-bridge type II.
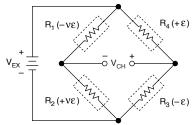

Full-bridge type II strain gage configurations have the following characteristics:

- Four active strain gage elements. Two are mounted in the direction of bending strain with one on the top side of the strain specimen and the other on the bottom side. The other two act together as a Poisson gage and are mounted transverse, or perpendicular, to the principal axis of strain with one on the top side of the strain specimen and the other on the bottom side.
- Rejection of axial strain.
- Compensation for temperature.
- Compensation for the aggregate effect on the principle strain measurement due to the Poisson's ratio of the material.
- Compensation for lead resistance.
- Sensitivity at 1000 $\mu\epsilon$ is ~ 1.3 mV_{out} / V_{EX} input.

Related concepts:

- [Bridge Sensor Scaling](#)

Full-Bridge Type II Circuit Diagram

The following symbols apply to the circuit diagram:

- R_1 is the active strain gage element measuring compressive Poisson effect ($-\varepsilon$).
- R_2 is the active strain gage element measuring tensile Poisson effect ($+\varepsilon$).
- R_3 is the active strain gage element measuring compressive strain ($-\varepsilon$).
- R_4 is the active strain gage element measuring tensile strain ($+\varepsilon$).
- V_{EX} is the excitation voltage.
- R_L is the lead resistance.
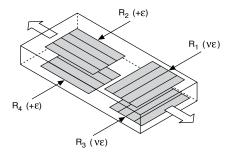- V_{CH} is the measured voltage.

The following equation converts voltage ratios to strain units for full-bridge type II configurations.

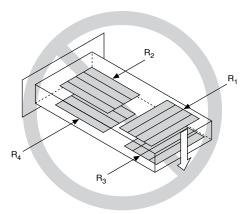$$\text{Strain}(\varepsilon) = \frac{-2V_r}{GF(\nu+1)}$$

where V_r is the voltage ratio that virtual channels use in the voltage-to-strain conversion equation, GF is the gage factor, and ν is the Poisson's ratio.

Full-Bridge Type III

The following figure shows how to position strain gage resistors in an axial configuration for the full-bridge type III.

The full-bridge type III configuration only measures the axial configuration.
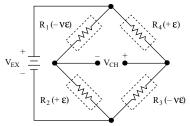

Full-bridge type III strain gage configurations have the following characteristics:

- Four active strain gage elements. Two are mounted in the direction of axial strain with one on the top side of the strain specimen and the other on the bottom side. The other two act together as a Poisson gage and are mounted transverse, or perpendicular, to the principal axis of strain with one on the top side of the strain specimen and the other on the bottom side.
- Compensation for temperature.
- Rejection of bending strain.
- Compensation for the aggregate effect on the principle strain measurement due to the Poisson's ratio of the material.
- Compensation for lead resistance.
- Sensitivity at 1000 µε is ~ 1.3 mV_{out} / V_{EX} input.

Related concepts:

- Bridge Sensor Scaling

Full-Bridge Type III Circuit Diagram

The following symbols apply to the circuit diagram:

- R_1 is the active strain gage element measuring compressive Poisson effect ($-\varepsilon$).
- R_2 is the active strain gage element measuring tensile strain ($+\varepsilon$).
- R_3 is the active strain gage element measuring compressive Poisson effect ($-\varepsilon$).
- R_4 is the active strain gage element measuring the tensile strain ($+\varepsilon$).
- V_{EX} is the excitation voltage.
- R_L is the lead resistance.
- V_{CH} is the measured voltage.

The following equation converts voltage ratios to strain units for full-bridge type III configurations.

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

where V_r is the voltage ratio that virtual channels use in the voltage-to-strain conversion equation, GF is the gage factor, and ν is the Poisson's ratio.

Strain Rosette

A strain gage can measure strain in only one direction—the axis along which the strain gage is mounted. To effectively measure the three independent components of plane strain (extensional strain along X and Y axis, as well as shear strain), three independent strain measurements are needed. Strain gage rosettes are used to perform such measurements.

A strain gage rosette is an arrangement of two or three closely positioned strain gages, separately oriented to measure the strains along different directions of the underlying

surface of the object being measured.

Strain-gage manufacturers offer three basic types of strain gage rosettes.

- **Tee Rosette**—A tee rosette consists of two gages oriented at 90 degrees with respect to each other.
- **Rectangular Rosette**—A rectangular rosette consists of three strain gages, each separated by a 45 degree angle.
- **Delta Rosette**—A delta rosette consists of three strain gages, each separated by a 60 degree angle.

Related concepts:

- Strain Gages

Eddy Current Proximity Probes

Proximity probes are sensors that measure relative proximity. They use changes in voltage to measure shaft surfaces that rotate or reciprocate. Because they are non-contacting transducers, proximity probes are mounted on a reasonably stationary mechanical structure, such as a bearing housing. From the mounting point, they measure the static and dynamic displacement behavior of the moving machinery. Use the proximity probe measurement type when you want to measure a dynamic position, such as an air gap between parts of moving machinery.

Eddy current proximity probes contain a driver and a monitor or regulated DC supply. The monitor or DC supply applies a power input of -24 VDC to the driver. The driver's internal oscillator converts some of the energy into a high-frequency radio signal. The signal is directed to the probe coil through a coaxial cable. The coil at the tip of the probe broadcasts the signal as a magnetic field into the surrounding area. If a conductive material intercepts the magnetic field, eddy currents are generated and the high-frequency radio signal loses power. The closer the conductive material is to the probe tip, the more power the signal loses. This power loss triggers a change in the voltage of the driver.

Proximity probe sensitivity is usually defined as the slope of a calibration curve as follows:

$$Sensitivity = \frac{Differential\ Voltage}{Differential\ Gap}$$

You can calculate the sensitivity by measuring two points in the sensor's dynamic range, using the following formula:

$$Sensitivity = \frac{Point1\ Voltage - Point2\ Voltage}{Point1\ Gap - Point2\ Gap}$$

> ✎ **Note** Proximity probes can differ in sensitivity. Refer to the documentation provided by the proximity probe manufacturer for information specific to your probe.

Offset voltage can occur when the tip of the probe touches the conductive material. Use the following formula to determine the offset voltage where offset is the output voltage of the sensor when the tip contacts the conductive material. In most cases, offset should be 0 V.

$$Voltage = Sensitivity * Gap + Offset$$

After you calibrate your proximity probe, you need to update the sensitivity and offset attributes. You can determine the physical distance between the sensor's tip and the target material by using the following formula:

$$Distance = (Voltage - Offset) / Sensitivity$$

Encoders

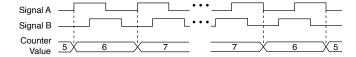There are two common types of encoders used for measuring position: two-pulse encoders and quadrature encoders.

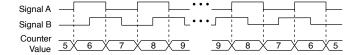**Related concepts:**

- Two-Pulse Encoders
- Quadrature Encoders

Quadrature Encoders

Quadrature encoders measure position by causing two signals to pulse while the encoder moves. These signals are signal A (also called channel A) and signal B (also called channel B). Signal A and B are offset by 90°, which determines the direction the encoder moves. For instance, in an angular quadrature encoder, if signal A leads, the encoder rotates clockwise. If signal B leads, the encoder rotates counter clockwise.
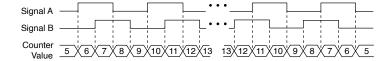
Counters on M Series, C Series, NI-TIO devices support three types of decoding for quadrature encoders: X1, X2, and X4. With X1 decoding, when signal A leads signal B, the counter increments on the rising edge of signal A. When signal B leads signal A, the counter decrements on the falling edge of signal A.

With X2 decoding, the same behavior holds as with X1, except the counter increments and decrements on both rising and falling edges of signal A.

Similarly, with X4 decoding, the counter increments and decrements on both rising and falling edges of both signal A and signal B. X4 decoding is more sensitive to position, but is also more likely to provide an incorrect measurement if there is vibration in the encoder.
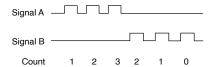

Many encoders also use z indexing for precise determination of a reference position.

Related concepts:

- [Z Indexing](#)

Two-Pulse Encoders

A two pulse encoder is a position measurement sensor that has two channels, A and B. When the encoder is moved, either signal A or signal B on the encoder pulses. A pulse on signal A represents a movement in one direction, and a pulse on signal B represents movement in the opposite direction. When signal A pulses, the counter increments. When signal B pulses, the counter decrements.

Many encoders also use z indexing for precise determination of a reference position.

Related concepts:

- [Z Indexing](#)

Z Indexing

Encoders typically use a third signal for Z indexing, which produces a pulse at fixed positions that you can use for precise determination of a reference position. For instance, if the Z index is 45° for an angular encoder, the encoder sends a pulse on the Z input terminal every time the encoder is turned to the 45° mark.

The behavior of signal Z differs with designs. You must refer to the documentation for an encoder to obtain the timing of signal Z in relation to the A and B signals. In NI-DAQmx, you can configure Z indexing with the **Z Index Phase** attribute/property.

Integrated Electronics Piezoelectric (IEPE) and Charge

Integrated Electronics Piezoelectric (IEPE) is a type of transducer that is packaged with a built-in amplifier. Because the charge produced by some sensors is very small, the electrical signal produced by the transducer is susceptible to noise, and sensitive electronics must be used to amplify and condition the signal. An IEPE sensor integrates

the sensitive electronics as close as possible to the transducer to ensure better noise immunity and convenient packaging. These sensors require a 4-20 mA current excitation to operate.

Charge Mode Operation

A charge mode sensor is a piezoelectric transducer which requires an amplifier, but unlike IEPE transducers, this amplifier is not built in. This separation of transducer and amplifier allows for the implementation of more complex amplifiers and filters because there is less constraints on physical space and power. The separation of piezoelectric element and signal conditioning also allows the transducer to operate over a higher temperature range. These sensors require no external excitation, but do require a device which has charge mode amplification.

Accelerometers

An accelerometer, a sensor that represents acceleration as a voltage, comes in two axial types. The most common accelerometer measures acceleration along only a single axis. This type is often used to measure mechanical vibration levels. The second type is a tri-axial accelerometer. This accelerometer can create a 3D vector of acceleration in the form of orthogonal components. Use this type when you need to determine the type of vibration—lateral, transverse, rotational, and so on—that a component is undergoing or the direction of acceleration of the component.

Both types of accelerometers come with either both leads insulated, or isolated, from the case or with one lead grounded to the case. Some accelerometers rely on the piezoelectric effect to generate voltage. To measure acceleration with this type of sensor, the sensor must be connected to a charge-sensitive amplifier.

Other accelerometers have a charge-sensitive amplifier built inside them. This amplifier accepts a constant current source and varies its impedance with respect to a varying charge on the piezoelectric crystal. You can see this change in impedance as a change in voltage across the inputs of the accelerometer. Thus, the accelerometer uses only two wires per axis for both sensor excitation, or current, and signal output, or voltage. The instrumentation for this type of accelerometer consists of a constant current source and an instrumentation, or differential, amplifier. The current source provides the excitation for the built-in amplifier of the sensor, while the instrumentation amplifier measures the voltage potential across the leads of the

sensor.

When choosing an accelerometer, pay attention to the most critical parameters. If the sensor must operate in extreme temperatures, you are limited to a sensor that relies on the piezoelectric effect to generate voltage. If the environment is very noisy, a sensor with a charge-sensitive amplifier built in might be the only usable choice.

To reduce errors when using an accelerometer, consider these factors:

- If the sensor is DC coupled, the DC offset of the accelerometer can drift with both temperature and age. This applies to both types of sensors because charge-sensitive amplifiers are prone to drift. AC coupling the output of the amplifier can minimize the drift in the system.
- Motors, transformers, and other industrial equipment can induce noise currents in the sensor cables. These currents can be an especially large source of noise with sensor systems that rely on the piezoelectric effect to generate voltage. Carefully routing sensor cables can minimize the noise in the cables.
- Accelerometers might have ground loops. Some accelerometers have their cases tied to a sense wire, while others are completely isolated from their cases. If you use a case-grounded sensor in a system with a grounded input amplifier, you set up a large ground loop, creating a source of noise.

Related concepts:

- Velocity Transducers

Force Sensors (Piezoelectric)

Piezoelectric force sensors are typically used to measure dynamic force events. Such sensors are divided into two main categories: load cells and impact hammers.

Piezoelectric load cells measure the amount of force transmitted through the sensor in response to an external stimulus, such as shaking or an impact.

Impact hammers allow you to apply an impact to a material and measure the actual amount of force applied. You can then correlate that force to the force measured by a load cell or to readings from accelerometers. Impact hammers can use tips of different size, shape, or material for measuring different frequencies.

> **Note** NI-DAQmx supports only bridge-based and IEPE force sensors.

Sensor manufacturers make load cells and impact hammers with varying sensitivity and other characteristics. Refer to the documentation from your sensor manufacturer for specifications and theory of operation for your load cell or impact hammer.

Sensor Calibration

Force sensors can drift from their documented sensitivity over time. Calibrating a force sensor involves determining its actual sensitivity. Use ratio calibration with a calibrated accelerometer to determine the actual sensitivity of a force sensor.
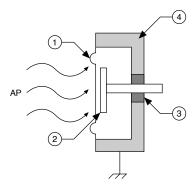
Related concepts:

- Accelerometers
- Velocity Transducers

Microphones

A microphone is a transducer that converts acoustical waves into electrical signals. The most common instrumentation microphone, a condenser microphone, uses a capacitive sensing element.

A condenser microphone incorporates a stretched metal diaphragm that forms one plate of a capacitor. A metal disk placed close to the diaphragm acts as a backplate. When a sound field excites the diaphragm, the capacitance between the two plates varies according to the variation in the sound pressure. A stable DC voltage is applied to the plates through a high resistance to keep electrical charges on the plate. The change in the capacitance generates an AC output proportional to the sound pressure. The following figure shows a condenser microphone.

AP = acoustic pressure, 1 = metal diaphragm, 2 = metal disk, 3 = insulator, 4 = case.

An instrumentation microphone usually consists of a microphone cartridge and a pre-amplifier. Sometimes these two components are independent; sometimes the components are combined and cannot be separated.

The major characteristics of a microphone are its sensitivity, usually expressed in mV/Pa, and its frequency response. Microphones are available in different diameters. Common diameters include: 1/8 in., 1/4 in., 1/2 in., and 1 in. Each diameter offers a specific compromise in terms of sensitivity and frequency response.

To reduce errors when using a microphone, keep several factors in mind:

- For measurements in a free field (a sound field with no major nearby reflections), use a free-field microphone pointed at the source of sound.
- For measurements in a diffuse field, such as inside in a highly reverberant room, where sound is coming from all directions, use a random incidence microphone.
- For measurements when the microphone is part of the surface of a room or of the object being measured, use a pressure microphone.
- For outdoor measurements, fit the microphone with suitable protection against the environment. This may include windscreens, rain caps, and built-in heaters to prevent condensation.
- To prevent vibrations from influencing the measurement, you might need to shock mount the microphone. Check the microphone specifications for vibration sensitivity.
- For reproducible measurements, make sure the microphone is mounted firmly and at a precisely reproducible location, both compared to the unit being tested and to the environment.
- Always calibrate the entire measurement chain, including the microphone, before starting the measurement. For highly critical measurements, as an extra

precaution, you may want to perform a new calibration immediately after the measurements are completed to make sure the system is still within tolerances.

Velocity Transducers

Velocity transducers are used to measure dynamic velocity such as that produced by a running machine or a vibrating structure.

Older velocity probes use a moving coil in a permanent magnetic field to generate a signal that is proportional to the velocity of the vibration.

More modern implementations operate using the same principles as IEPE accelerometers. Additionally, an integrating circuit is used to convert acceleration to velocity. Use the same best practices for storage and handling, calibration, mounting, and signal conditioning of IEPE velocity sensors as you would for IEPE accelerometers.

Note NI-DAQmx supports only IEPE velocity sensors.

Sensor manufacturers make IEPE velocity transducers with varying sensitivity and other characteristics. Use a sensor appropriate for your application.

Overview of Temperature Sensor Types

The three most commonly used transducers for temperature are thermocouples, resistance temperature detectors (RTDs), and thermistors. The following table illustrates some of the capabilities and limitations of these sensors. Use this table as a reference for choosing the right sensor for your temperature measurement application.

Sensor	Advantages	Disadvantages
Thermocouples	wide range, fast response, inexpensive	require CJC, nonlinear
RTDs	rugged, accurate	slow response, require excitation, lead resistance, nonlinear
Thermistors	repeatable, fine resolution, low	require excitation, narrow range,

Sensor	Advantages	Disadvantages
	current, fast response	nonlinear

Related concepts:

- Thermocouples
- Resistance Temperature Detectors (RTDs)
- Thermistors

Resistance Temperature Detectors (RTDs)

An RTD is a temperature sensing device with resistance that increases with temperature. An RTD is usually constructed with wire coil or deposited film of pure metal. RTDs can be made of different metals and have different nominal resistances, but the most popular RTD is platinum and has a nominal resistance of 100 Ω at 0 °C.

Signal conditioning is generally required to measure temperature using an RTD. Because an RTD is a resistive device, you must pass a current through the device to produce a measurable voltage. Providing current to take a resistive measurement is a form of signal conditioning called current excitation. In addition to producing current excitation for the RTD, signal conditioning amplifies the output voltage signal, and filters the signal to remove unwanted noise. You also can use signal conditioning to electrically isolate the RTD and the monitored system from the DAQ system and the host computer. Refer to *Signal Conditioning Requirements for Thermistors and RTDs* for more information.

Numerous types of RTDs exist, and they are typically defined by their material, their nominal resistance, and their temperature coefficient of resistance (TCR). The TCR of an RTD is the average temperature coefficient of resistance of the RTD from 0 to 100 °C and is the most common method of specifying the behavior of an RTD. The TCR for platinum RTDs is determined by the Callendar-Van Dusen equation.

Related concepts:

- Signal Conditioning Requirements for Thermistors and RTDs

- Callendar-Van Dusen Equation
- Platinum RTD Types
- 2-Wire Resistance
- 3-Wire Resistance
- 4-Wire Resistance

Platinum RTD Types

The following table lists common platinum RTD types and standards. All of these RTD types are supported in NI-DAQmx. Notice that there are some shared standards. The TCR and the Callendar-Van Dusen coefficients are more important than the standards.

Standards	Material	TCR	Typical R_0 (Ω)	Callendar-Van Dusen Coefficient	Notes
- IEC-751 - DIN 43760 - BS 1904 - ASTM-E1137 - EN-60751 - IEC-60751	Platinum	3851	- 100 Ω - 1000 Ω	- $A = 3.9083 \times 10^{-3}$ - $B = -5.775 \times 10^{-7}$ - $C = -4.183 \times 10^{-12}$	Most common RTDs
Low-cost vendor compliant RTD[66]	Platinum	3750	1000 Ω	- $A = 3.81 \times 10^{-3}$ - $B = -6.02 \times 10^{-7}$ - $C = -6.0 \times 10^{-12}$	Low-cost RTD
JISC 1604 1997	Platinum	3916	100 Ω	- $A = 3.9739 \times 10^{-3}$ - $B = -5.870 \times 10^{-7}$ - $C = -4.4 \times 10^{-12}$	Used primarily in Japan
US Industrial Standard D-100 American	Platinum	3920	100 Ω	- $A = 3.9787 \times 10^{-3}$ - $B = -5.8686 \times 10^{-7}$	Low-cost RTD

66. No standard. Check the TCR.

Standards	Material	TCR	Typical R_0 (Ω)	Callendar-Van Dusen Coefficient	Notes
				• $C = -4.167 \times 10^{-12}$	
US Industrial Standard American	Platinum	3911	100 Ω	• $A = 3.9692 \times 10^{-3}$ • $B = -5.8495 \times 10^{-7}$ • $C = -4.233 \times 10^{-12}$	Low-cost RTD
ITS-90	Platinum	3928	100 Ω	• $A = 3.9888 \times 10^{-3}$ • $B = -5.915 \times 10^{-7}$ • $C = -3.85 \times 10^{-12}$	The definition of temperature

Related concepts:

- [Callendar-Van Dusen Equation](#)

Callendar-Van Dusen Equation

Platinum RTDs use a linearization curve known as the Callendar-Van Dusen equation to measure the temperature of RTDs. The equation is as follows:

Temperatures below 0 °C:

$$R_T = R_0[1 + A \times T + B \times T^2 + C \times T^3 \times (T - 100 \text{ °C})]$$

Temperatures above 0 °C:

$$R_T = R_0[1 + A \times T + B \times T^2]$$

Where:

T = temperature in degrees Celsius

R_T = RTD resistance at temperature T

R_0 = RTD nominal resistance at 0 °C

A, B, and C = coefficients given in the table in Platinum RTD Types.

Related concepts:

- Platinum RTD Types

Thermistors

A thermistor is a piece of semiconductor made from metal oxides, pressed into a small bead, disk, wafer, or other shape, heated at high temperatures, and coated with epoxy or glass.

Like RTDs, by passing a current through a thermistor, you can read the voltage across the thermistor and thus determine its temperature. Unlike RTDs, thermistors have a higher resistance (anywhere from 2,000 to 10,000 Ω) and a much higher sensitivity (~200 Ω/°C). However, thermistors are generally used only up to the 300 °C temperature range.

NI-DAQmx scales the resistance of a thermistor to a temperature using the Steinhart-Hart thermistor equation:

$$\frac{1}{T} = A + B(\ln(R)) + C(\ln(R))^3$$

where T is the temperature in Kelvins, R is the measured resistance, and A, B, and C are constants provided by the thermistor manufacturer.

Because thermistors have high resistance, lead-wire resistance does not affect the accuracy of the measurements. Unlike RTDs, 2-wire measurements are adequate.

For more information about the signal conditioning requirements of a thermistor, refer to *Signal Conditioning Requirements for Thermistors and RTDs*.

Related concepts:

- [Signal Conditioning Requirements for Thermistors and RTDs](#)

Signal Conditioning Requirements for Thermistors and RTDs

Thermistors and RTDs require the following signal conditioning:

- **Current Excitation**—Because RTDs and thermistors are resistive devices, your DAQ system must provide a current excitation source to measure a voltage across the device. This current source must be constant and precise.
- **2-, 3-, and 4-Wire Configurations (RTDs only)**—RTDs come in 2-, 3-, and 4-wire configurations. Therefore, your system must support the type of RTD you choose. Thermistors are typically 2-wire devices because they have higher resistance characteristics, thus eliminating lead resistance considerations.
- **Linearization** —Neither RTD nor thermistor output voltage is linear with temperature. Therefore, your system must perform linearization either in hardware or software.

Related concepts:

- [2-Wire Resistance](#)
- [3-Wire Resistance](#)
- [4-Wire Resistance](#)

Thermocouples

Thermocouples are the most commonly used temperature sensors.

A thermocouple is created when two dissimilar metals touch and the contact point produces a small open-circuit voltage that corresponds to temperature. This thermoelectric voltage is known as Seebeck voltage and is nonlinear with respect to temperature. Thermocouples require signal conditioning.

Thermocouple types differ in composition and accurate range:

Thermocouple Type	Positive Conductor	Negative Conductor	Temperature Range (°C) for Polynomial Coefficients or for Table Conversion	Temperature Range (°C) for Inverse Polynomial Coefficients
J	Iron	Constantan	-210 to 1200	-210 to 1200
K	Chromel	Alumel	-270 to 1372	-200 to 1372
N	Nicrosil	Nisil	-270 to 1300	-200 to 1300
R	Platinum-13% Rhodium	Platinum	-50 to 1768	-50 to 1768
S	Platinum-10% Rhodium	Platinum	-50 to 1768	-50 to 1768
T	Copper	Constantan	-270 to 400	-200 to 400
B	Platinum	Rhodium	0 to 1820	250 to 1820
E	Chromel	Constantan	-270 to 1000	-200 to 1000

Use the temperature ranges for polynomial coefficients when converting temperature to voltage. For most thermocouples, the equation used for converting temperature to voltage is the following:

$$V_R = \sum_{i=0}^{n} c_i \left(t_{90}\right)^i$$

where E is the voltage in millivolts, t_{90} is the temperature in degrees Celsius, and c_i is the coefficient.

Use the temperature ranges for inverse polynomial coefficients when converting voltage to temperature. For most thermocouples, the equation for converting voltage to temperature is the following:

$$t_{90} = \sum_{i=0}^{n} D_i \left(E\right)^i$$

where t_{90} is the temperature in degrees Celsius, E is the voltage in millivolts, and D_i is the coefficient.

> **Note** For coefficients to use with each thermocouple type, visit the NIST ITS-90 thermocouple database available at nist.gov.

Related concepts:

- Signal Conditioning Requirements for Thermocouples
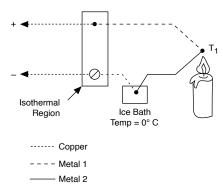
Related information:

- nist.gov

Signal Conditioning Requirements for Thermocouples

Thermocouples require the following signal conditioning:

- **Amplification for High-Resolution ADC**—Thermocouples generate very low-voltage signals, usually measured in microvolts. To acquire these signals with a measurement device, you must amplify the thermocouple signal to measure it accurately with a standard 12-bit measurement device. Alternatively, you can use a measurement device with a high-resolution ADC. NI recommends a device with 16 bits of resolution and amplification capabilities or a device with 24 bits of resolution.
- **Cold-Junction Compensation**—Thermocouples require some form of temperature reference to compensate for unwanted parasitic thermocouples. A parasitic thermocouple is created when you connect a thermocouple to an instrument. Because the terminals on the instrument are made of a different material than the thermocouple wire, voltage is created at the junctions, called cold junctions, which changes the voltage output by the actual thermocouple.

  Traditionally, the temperature reference was 0 °C. The National Institute of Standards and Technology (NIST) thermocouple reference tables are created using this setup. Although an ice bath reference is quite 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, such as a thermistor or an IC sensor, and then subtract the parasitic thermocouple thermoelectric contributions. This process is called cold-junction compensation.

------- Copper
- - - - Metal 1
——— Metal 2

- **Filtering**—A thermocouple can act much like an antenna, making it very susceptible to noise from nearby 50/60 Hz power sources. Therefore, apply a 2 Hz or 4 Hz lowpass filter to your thermocouple signal to remove power line noise.
- **Linearization**—The output voltage of a thermocouple is not linear with temperature. Therefore, your system must perform linearization either through hardware or software.

LVDTs

LVDTs operate on the principle of a transformer and consist of a stationary coil assembly and a moveable core. An LVDT measures displacement by associating a specific signal value for any given position of the core. LVDT signal conditioners generate a sine wave for the primary output signal and synchronously demodulate the secondary output signal. The demodulated output is passed through a lowpass filter to remove high-frequency ripple. The resulting output is a DC voltage proportional to core displacement. The sign of the DC voltage indicates whether the displacement is to the left or right.

LVDTs require special electronics designed for the sensor. LVDTs typically have a delay of approximately 10 ms caused by filtering in the signal conditioner.

LVDTs typically come in 4-wire, or open wire, and 5-wire, or ratiometric wire, configurations. Wires from the sensor connect to a signal conditioning circuit that translates the output of the LVDT to a measurable voltage. The method of signal conditioning used on the signals from the first and second secondaries differentiate the 4-wire and 5-wire configurations. In the 4-wire configuration, the sensor only measures the voltage difference between the two secondaries.

The benefit of using a 4-wire configuration is that you require a simpler signal conditioning system. However, temperature changes can alter the efficiency of the magnetic induction of the LVDT. Because the 4-wire scheme is also sensitive to phase changes between the primary and the resulting secondary voltage, long wires or a poor excitation source also can cause problems.

The 5-wire configuration is less sensitive to both temperature changes and phase differences between the primary and the secondaries. The device determines phase information at the signal conditioning circuitry without needing to reference the phase of the primary excitation source. Therefore, you can use longer wires between the LVDT and the signal conditioning circuitry.

LVDTs are extremely rugged, operate over wide temperature ranges, and are insensitive to moisture and dirt. LVDTs are a preferred sensor in harsh environments, where very long life is needed because there are no moving parts in contact or where very low friction is required. Also, LVDT technology lends itself well to applications requiring accurate measurements less than 0.1 in., such as measuring the thickness of sheet material. The main advantage of the LVDT transducer over other types of displacement transducer is the high degree of robustness. Because there is no physical contact across the sensing element, there is no wear in the sensing element.

Because the device relies on the coupling of magnetic flux, an LVDT can have infinite resolution. Therefore, suitable signal conditioning hardware can detect the smallest fraction of movement, and only the resolution of the data acquisition system determines the resolution of the transducer.

RVDTs

RVDTs are the rotational version of LVDTs and generally operate over an angular range of ±30°–70°. They are available in servo-mount and can rotate through 360° without stopping.

RVDTs require special electronics designed for the sensor. RVDTs typically have a delay of 10 ms caused by required filtering in the signal conditioner. They are extremely rugged and operate over wide temperature ranges. In environments characterized by extremes in temperature and shock, an RVDT is the clear choice for rotational applications when you need more than 70° of measurement range.

Related concepts:

- [LVDTs](#)

Transducer Electronic Data Sheets (TEDS)

IEEE P1451.4 is an emerging standard for adding plug and play capabilities to analog transducers. The underlying mechanism for plug and play identification is the standardization of a Transducer Electronic Data Sheet (TEDS). A TEDS contains the critical information needed by a device or measurement system to identify, characterize, interface, and properly use signals from an analog sensor. That information includes the sensor's model number, model ID, calibration constants, scaling constants, and more.

A TEDS is deployed for a sensor in one of two ways:

- A TEDS can reside in embedded memory, typically an EEPROM, within the sensor. To download a TEDS from the sensor, you need TEDS-supported hardware such as the BNC-2096 or the SCXI-1314T. You can then download the TEDS in MAX to use in your application. Refer to the ***Measurement & Automation Explorer Help for NI-DAQmx*** for additional information on downloading and using TEDS.
- A Virtual TEDS can exist as a separate file, downloadable from the internet. A Virtual TEDS extends the benefits of the standardized TEDS to legacy sensors and applications in which the embedded memory or EEPROM is not available. You can download Virtual TEDS from [ni.com](#) by typing in the sensor serial number. A Virtual TEDS does not require TEDS-supported hardware.

Writing Data to TEDS Sensors

Use the Write TEDS Data function/VI to write data to a TEDS sensor. The TEDS data must be in a virtual TEDS file or in a bitstream constructed according to the IEEE 1451.4 specification.

National Instruments provides a LabVIEW library for viewing and editing TEDS bitstreams and virtual TEDS files. You can download the TEDS Library for LabVIEW at [How Do I Install the TEDS Library into LabVIEW?](#).
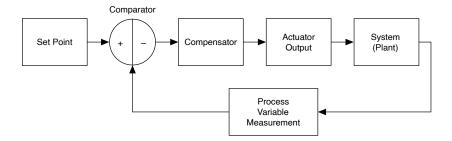
Related concepts:

- Transducer Electronic Data Sheets (TEDS)

Basic TEDS Data

Some TEDS sensors include a PROM, to which you can write data one time. When you write TEDS data, you can choose to write basic TEDS data to the PROM or to the EEPROM. Basic TEDS data includes the manufacturer ID, model number, serial number, version number, and version letter. If you write basic TEDS data to the PROM, the Write TEDS Data function/VI returns an error if you later attempt to write basic TEDS data to the EEPROM.

Control Overview

In a typical control application, there are one or more process variables that you want to control, such as temperature. Sensors measure the process variable in the dynamic system and provide the data to the control application. The set point is the value you want for the process variable. A comparator determines if a difference exists between the process variable and the set point. If a difference exists and if the control system deems the difference large enough, the compensator processes the data and determines the desired actuator output to drive the system closer to the set point.

For example, in a temperature measurement system, if the actual temperature is 100 °C and the temperature set point is 120 °C, the compensator needs to take some action to raise the temperature. One actuator output might be to drive a heater at 62 percent of its maximum output capacity. The increased heater actuator output causes the system to become warmer, which results in an increased temperature. This kind of system is called a closed-loop control system because the process of reading sensors

and calculating the actuator output you want repeats continuously at a fixed loop rate.

Related concepts:

- Proportional-Integral-Derivative (PID)
- Real Time
- Loop Cycle Time
- Jitter Overview for Control Applications
- Event Response

Proportional-Integral-Derivative (PID)

The Proportional-Integral-Derivative (PID) algorithm is the most common control algorithm used in industry. Often, people use PID to control processes that include heating and cooling systems, fluid level monitoring, flow control, and pressure control. In PID control, you must specify a process variable and a setpoint. The process variable is the system parameter you want to control, such as temperature, pressure, or flow rate, and the setpoint is the desired value for the parameter you are controlling. A PID controller determines a controller output value, such as the heater power or valve position. The controller applies the controller output value to the system, which in turn drives the process variable toward the setpoint value.

Real Time

Real time means that responses occur in time, or on time. With non-real-time systems, there is no way to ensure that a response occurs within any time period, and operations may finish much later or earlier than expected. In other words, real-time systems are deterministic, which guarantees that operations occur within a given time. Real-time systems are predictable.

For a system to be a real-time system, all parts of it need to be real time. For instance, even though a program runs in a real-time operating system, it does not mean that the program behaves with real-time characteristics. The program may rely on something that does not behave in real-time such as file I/O, which then causes the program to not behave in real-time.
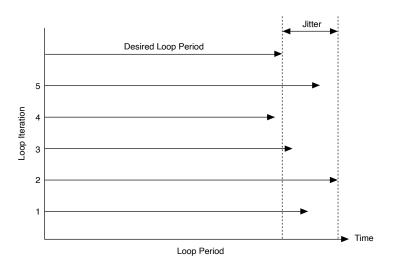
Loop Cycle Time

Many applications that require a real-time operating system are cyclic, such as a control application. The time between the start and finish of each cycle, T, is called the ***loop cycle time*** (or sample period). 1/T is the loop rate or sample rate. Even with real-time operating systems, the loop cycle time can vary between cycles, but will not be greater than the maximum jitter.

Related concepts:

- Jitter Overview for Control Applications

Jitter Overview for Control Applications

For control applications, the amount of time that the loop cycle time varies from the desired time is called jitter. The maximum amount that a loop cycle time varies from the desired loop cycle time is called maximum jitter.

In real-time systems, jitter is bounded. For instance, air bags must deploy within fractions of a second after a critical impact and are thus bound to a maximum jitter. In non-real-time systems, jitter is unbounded—or very large. Waiting for a bus is an example. Suppose that according to the schedule, the bus is supposed to arrive at 11:00 a.m. but actually arrives at 11:05 a.m. one day, 11:30 a.m. the next day, and has a flat tire the day after that. There is no bound on how late the bus could arrive.

Related concepts:

- [Loop Cycle Time](#)

Event Response

Event response applications require a response to a stimulus in a determined amount of time. An example is monitoring the temperature of an engine. When the temperature rises too high, the engine is slowed down. The event, in this case, is the temperature rising above a predetermined level, and the response is the engine slowing down. Another example comes from manufacturing. In a manufacturing line, a system senses when a part is in front of a station (the event) and takes a reading or manipulates the part (the response). If the system does not sense and respond to the presence of that part in a set amount of time, the manufacturing line creates defective parts.