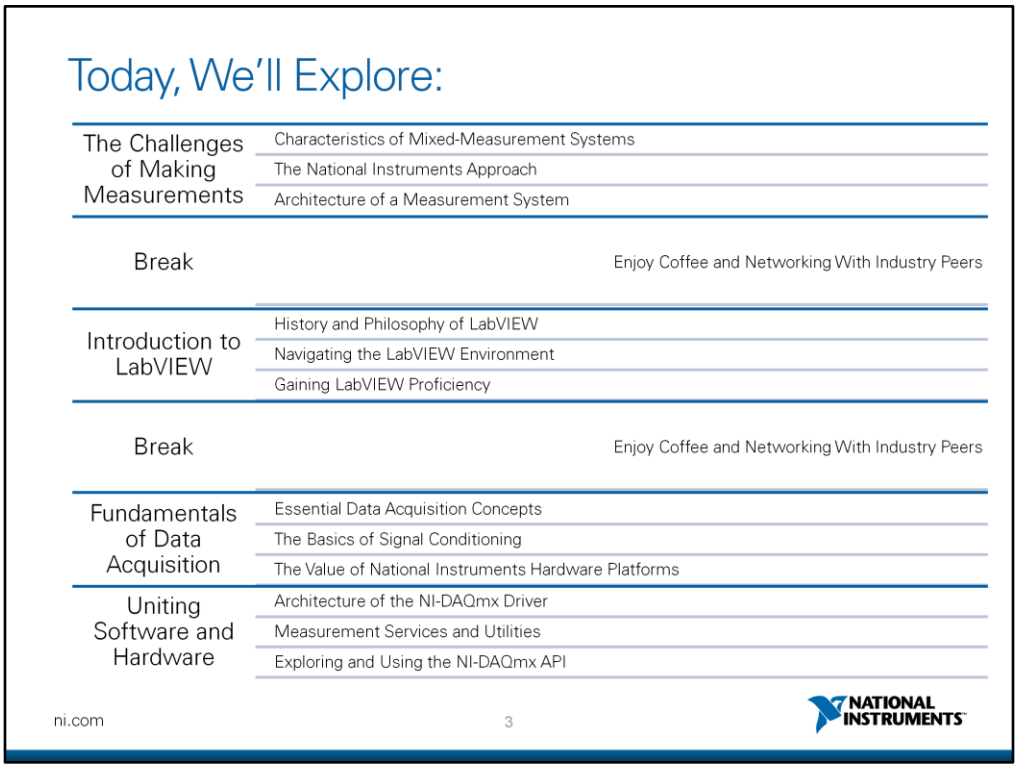
Introduction to NI LabVIEW and Computer-Based Measurements

National Instruments

Welcome to the Introduction to LabVIEW and Computer-Based Measurements seminar. This seminar will provide you an introduction to building measurement and automation solutions using graphical system design – the highly differentiated, industry-preferred approach to making automated measurements.

Through a combination of presentation slides, instructor-led demos, and hands-on exercises, you will leave today's seminar with a confident understanding as to why the combination of NI LabVIEW system design software and NI DAQ hardware is the most productive, flexible approach to building accurate and reliable automated measurement solutions.

ni.com/labview
ni.com/daq



It is important to begin today's seminar by gaining alignment on the challenges that engineers and scientists face when building an automated measurement system. From buying turnkey systems to building all components from scratch, there are a plethora of options available on the market. Depending on the size and complexity of the application, your investment can be a substantial one; by making an uneducated decision on any aspect comprising your solution, you may find yourself having wasted both time and money on an approach that is unsuitable for any number of unforeseen reasons.

Throughout the day, you will be provided your first introduction to LabVIEW system design software, learn the concepts necessary to understand the fundamentals of data acquisition, and explore National Instruments data acquisition hardware offerings. Finally, you'll see how to combine your new software and hardware knowledge into an integrated solution.

ni.com/labview
ni.com/daq

The Challenges of Making Measurements

Exploring the Traditional Approach to Measurements

ni.com

NATIONAL INSTRUMENTS

The Origin of Automated Measurements

- Traditional pen-and-paper approach
- Redundant circuitry between instruments (e.g., displays)
- Manual data recording and analysis
- Error-prone processes
- Difficult to reproduce or redo

Thermoelectric Voltage in mV

°C	0	1	2	3	4	5	6	7	8	9	10
0	0.000	0.050	0.101	0.151	0.202	0.253	0.303	0.354	0.405	0.456	0.507
10	0.507	0.558	0.609	0.660	0.711	0.762	0.814	0.865	0.916	0.968	1.019
20	1.019	1.071	1.122	1.174	1.226	1.277	1.329	1.381	1.433	1.485	1.537
30	1.537	1.589	1.641	1.693	1.745	1.797	1.849	1.902	1.954	2.006	2.059
40	2.059	2.111	2.164	2.216	2.269	2.322	2.374	2.427	2.480	2.532	2.585
50	2.585	2.638	2.691	2.744	2.797	2.850	2.903	2.956	3.009	3.062	3.116
60	3.116	3.169	3.222	3.275	3.329	3.382	3.436	3.489	3.543	3.596	3.650
[illegible]	3.650	3.703	3.757	3.810	3.864	3.918	3.971	4.025	4.079	4.133	4.187
[illegible]	[illegible]	4.240	4.294	4.348	4.402	4.456	4.510	4.564	4.618	4.672	4.726
[illegible]	[illegible]	[illegible]	4.835	4.889	4.943	4.997	5.052	5.106	5.160	5.215	5.269

ni.com 5 NATIONAL INSTRUMENTS

Consider a common scenario: you're an engineer in a lab designing a new circuit and you're running some tests to see how it behaves under different conditions. Most any engineer has had to do this at some point in their career, and getting information about the system's behavior during these experiments is key to producing the right result. The instruments that are often used for this are very simple, but also very large and very expensive. The display looks like it's from the 1980s, but actually the instrument pictured here continues to be one of the most popular DMMs on the market and it sells for over a thousand dollars. Keep in mind, you're looking at an instrument that can only acquire, has a max sample rate of a 1000 readings a second, and only two channels.

So as we monitor our system's behavior, we're often writing values down by hand or punching them into a spreadsheet. More than likely, this is only one of several instruments we'd be using for this project. Consider what you end up with: a notepad or excel sheet filled with numbers we punched in, which is obviously prone to human error, and results that are very hard to reproduce or redo if we need to start over.

Now, those results are often just the starting point – they probably aren't actually the information you need… we'll have to analyze the numbers to do that, which is an entirely different challenge in and of itself.

Measurement Challenges Are Compounded By:

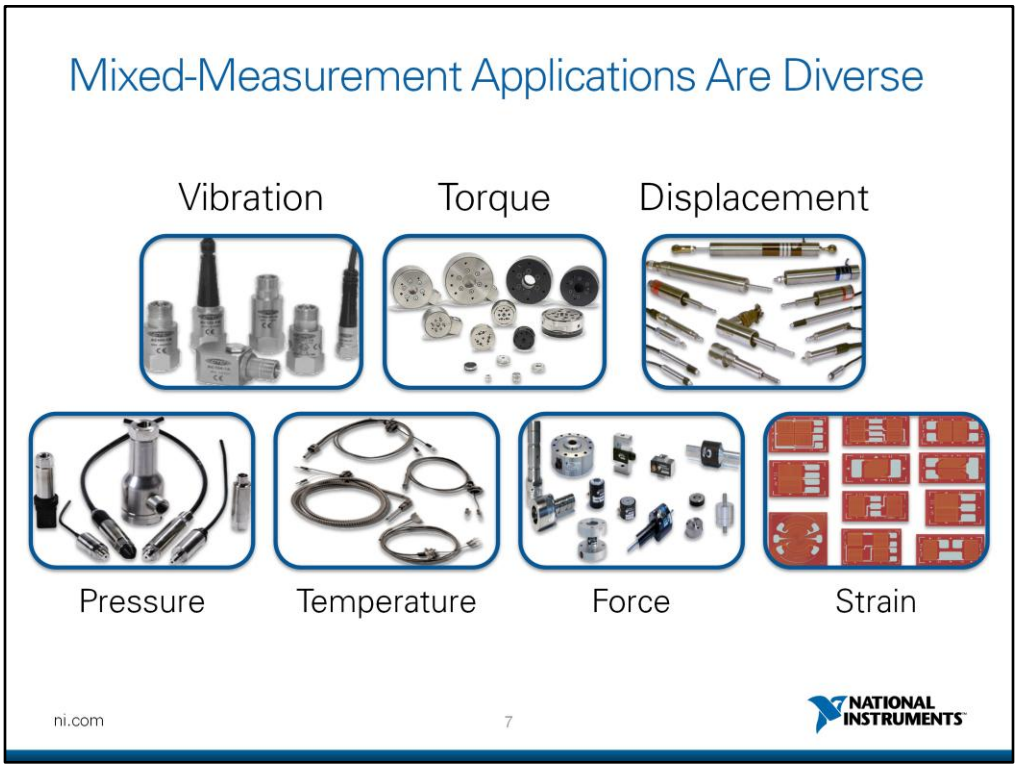
- Compressed Timelines
- Fixed Software and Hardware
- Conflicting Programming Approaches
- Inadequate Hardware Performance
- Disparate Driver APIs
- Varying Sensors and Connectivity
- Custom Signal Conditioning
- Advanced Visualization
- Changing Application Requirements
- Complex Analysis Algorithms
- Evolving Technology Trends
- Confusing Data Storage
- Differing Sampling Rates

Automated measurements are the natural evolution of the manual, error-prone recording processes we discussed previously; however, today's measurement applications are not without their own roadblocks. Whether your individual application faces any these challenges today or not, it is important to be aware of each of the potential complications listed on this slide, as adopting the wrong approach could subject your solution to unforeseen consequences that could have been easily mitigated or avoided had all drawbacks been considered. It is also imperative that you take into account the potential ways in which your application could fluctuate in the future, as requirements are not always clear-cut.

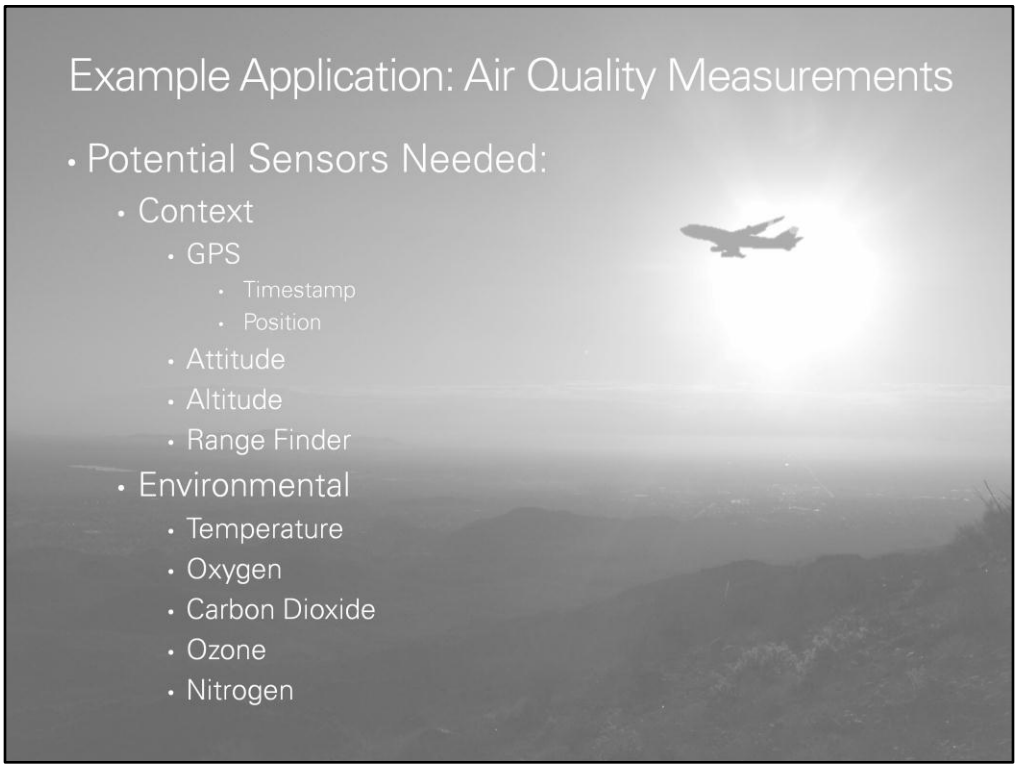
The obvious and straightforward penalties of adopting the wrong measurement methodology – as indicated by the negative space represented in the slide imagery – are wasted time and money, the luxury of which you may or may not have in the era of doing more with less.

Perhaps the more detrimental and indistinct dangers of the wrong approach to measurements, however, come from a system that inaccurately or inadequately measures the phenomena you're trying to capture, the precise results of which affect the welfare of human lives.

Word cloud generated from:
http://www.tagxedo.com/



One of the fundamental attributes that compound the challenges of today's measurement systems is the mixed-measurement nature of the solutions being built. Taking mixed measurements – which is extremely common in most of today's data acquisition applications due to the advances in technology and availability – is complex. No matter what physical phenomena you are trying to measure, there is a different sensor type with dozens of different sensor options, different outputs, different connection types, different signal conditioning needs, and even different units of measurement. Coordinating this effort is challenging. If a system is thrown together in a haphazard way, ease of scalability and likelihood of success are severely hindered.



To prove this point, let's examine an example mixed-measurement application.

Imagine you're tasked with building a system that will be affixed to commercial airplanes in order to take air quality measurements while the aircraft are flying their everyday routes. For the sake of simplicity, let's ignore the real-world restriction of budgets and timelines. What phenomena would you want to measure in a system such as this one?

In all likelihood, you've brainstormed a system that attempts to capture far more than just one single type of measurement. Common measurements in a real-world system such as this might record a series of both contextual and environmental information including such as:

- GPS (to accurately document the timestamp and position of the aircraft)
- Altitude (to measure the height at which sample was recorded)
- Range (to detect whether objects in the vicinity have drastically altered the validity of readings)

…in addition to each of the air quality measurements we would have been required to record in the first place.

Sensors, Interfaces, and Signal Conditioning

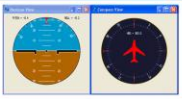
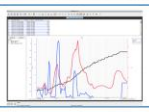
Sensor		Interface	Conditioning?
GPS		RS232	No
Attitude, Altitude		RS232	No
LiDAR		Ethernet	No
Temperature		Analog Voltage	Required
O_2, CO_2, O_3, NH_3		Analog Voltage	Required

NATIONAL INSTRUMENTS

A quick internet search yields an overwhelming number of sensor options for each measurement type. Dozens of different vendors sell sensors that would accomplish the measurements listed on the slide, and each sensor choice has varying amounts of features and functionality. One of the immediate challenges that sticks out is that each sensor seems to provide a different interface that we'll have to accommodate, with some connecting via serial, some via Ethernet, some outputting simple analog voltages. Along with the different connectivity that we'll have to work with, some sensors also require signal conditioning, which means that we'll have to build our own custom circuitry to take care of things such as filtering, amplification, cold junction compensation, isolation, or more.

ni.com/labview
ni.com/daq

Software Provided With Sensors

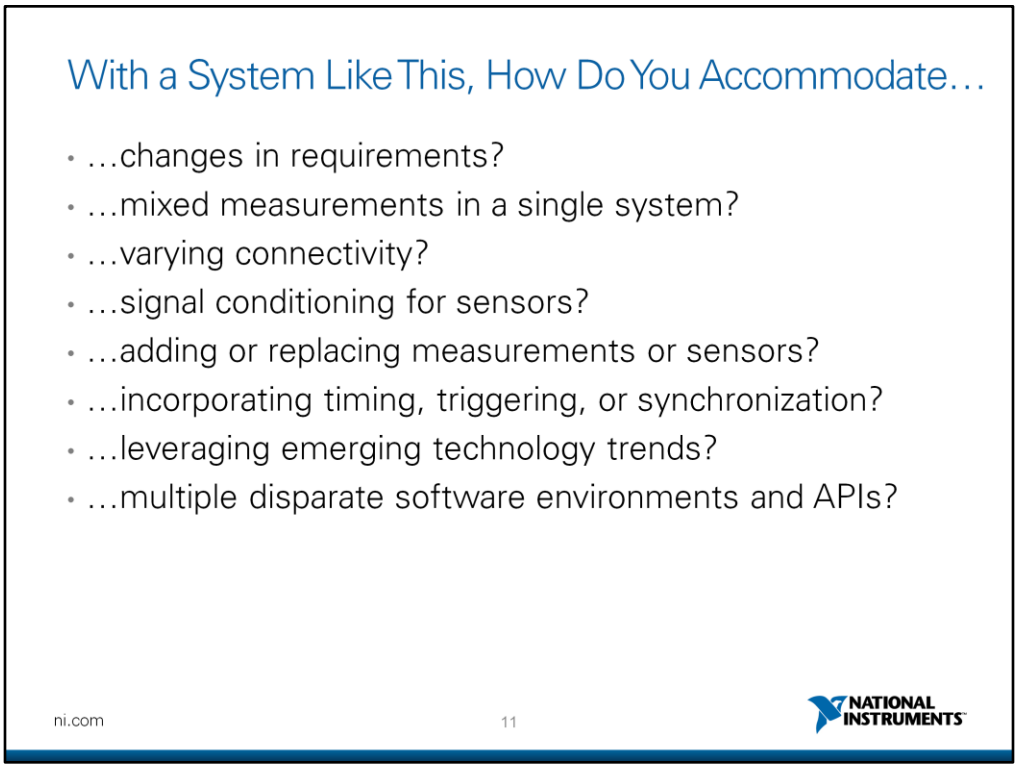
Sensor		Software
GPS		
Attitude, Altitude		
LiDAR		
Temperature		
O_2, CO_2, O_3, NH_3		<No Software Provided>

ni.com 10 NATIONAL INSTRUMENTS

Sensor vendors are rarely in business of creating software. That said, most sensor vendors feel obligated to provide some form of simplistic software interface that, at the very least, will allow you to interactively take a measurement from their sensor. Additionally, the majority of sensor vendors will provide some type of application programming interface (API) – typically in the form of a distributed function library – that will allow you to interface with the sensor programmatically.

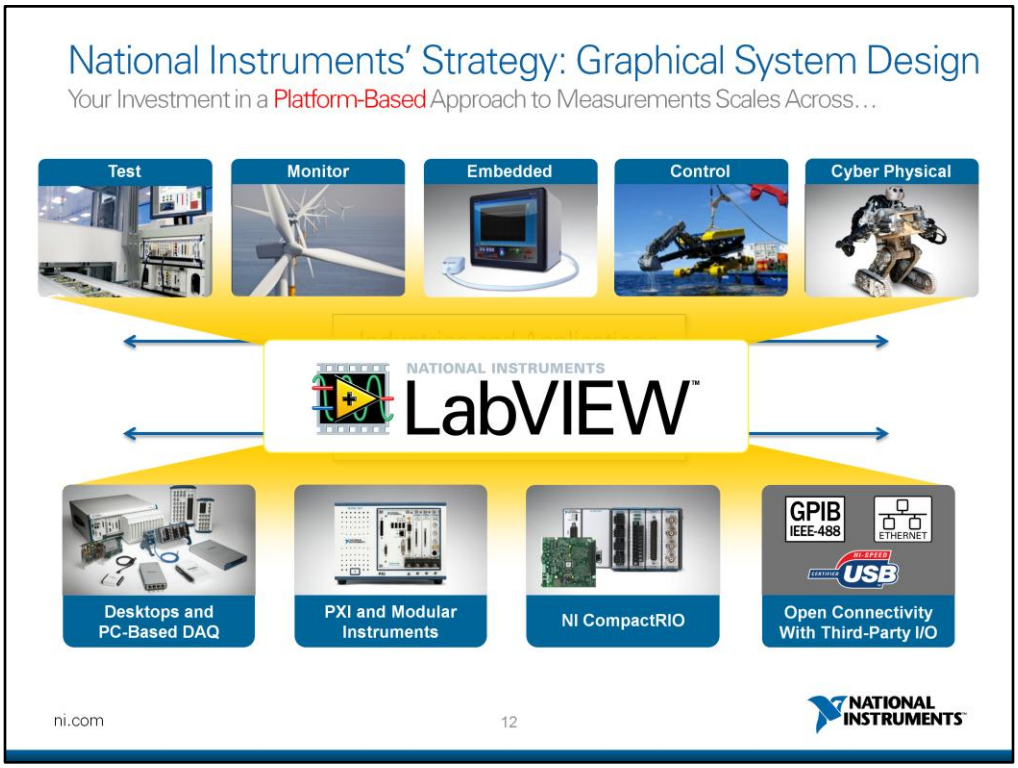
Since we would want to use each one of the sensors in an automated fashion by incorporating the sensors into a single, integrated system, the interactive software tools are all but useless to us aside from being applicable as quick test applications that would allow us to make sure the devices are functioning. After all, we wouldn't want to have to interface with five separate pieces of software (not to mention doing so interactively) just to take a measurement. Instead, we would rely solely on the API to communicate with the sensor via code.

Unfortunately, not all APIs are created equal. It is highly likely that each shared library we receive could take a different form (for example, a .NET assembly versus a C DLL), which means we may or may not be able to even use them from our chosen software environment if we hadn't considered this beforehand. Furthermore, each API will have a different extent of documentation (some providing none at all), expect different syntax, and come accompanied with varying amounts of examples (if at all).



After acknowledging just a few of the challenges associated with our hypothetical system – including but not limited to disparate connectivity, signal conditioning, and software interfaces – you can begin to understand how quickly the difficulties of a mixed-measurement data acquisition system add up. If you don't carefully consider the effect of each on your system, complexity could spiral beyond our control.

What if the requirements of our example application changed in a way that we hadn't accounted for – such as needing to incorporate additional measurements in the future, or take advantage of industry trends such as wireless communication – would our system be able to accommodate those obstacles, or will our chosen architecture become a dead end?

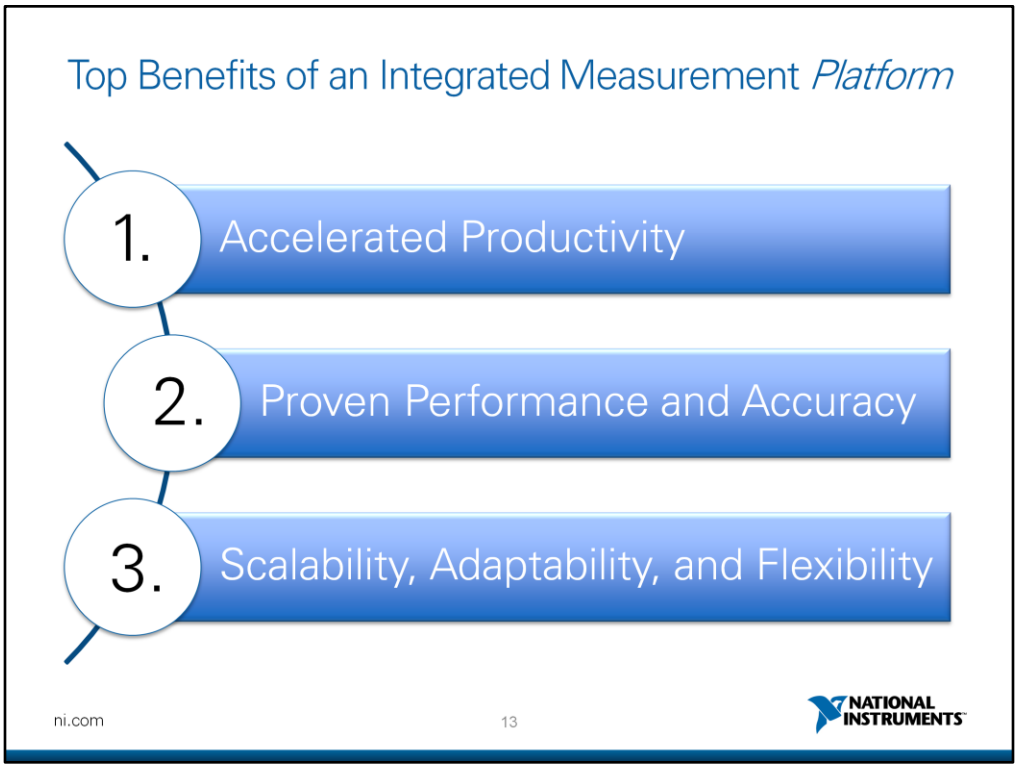


National Instruments calls this approach to measurements *graphical system design*. Your LabVIEW-based measurement systems will move you from measurement to decision faster thanks to the graphical system design approach, which provides reuse of a software-centered platform targeting multiple hardware footprints that share a reconfigurable architecture. Once you've applied the platform to a single problem, you can use it to scale to multiple problems, even if they involve future requirements or technologies.

Our goal in providing this platform is to equip engineers and scientists with all of the tools necessary to build absolute any measurement or control system, which spans a wide array of industries and application areas.

The integrated software and hardware nature of this platform makes it easier to visualize and implement all of the technology, I/O, math and models needed for a system. Within LabVIEW you can use different models of computation to describe the way the system works, and then compile to the best target to run your system. With graphical system design, you can close quickly on not just the visualization, but the implementation, of the systems you need.

ni.com/labview
ni.com/daq



The graphical system design approach to automated measurements will provide you three key advantages over traditional measurement systems.

ni.com/labview
ni.com/daq

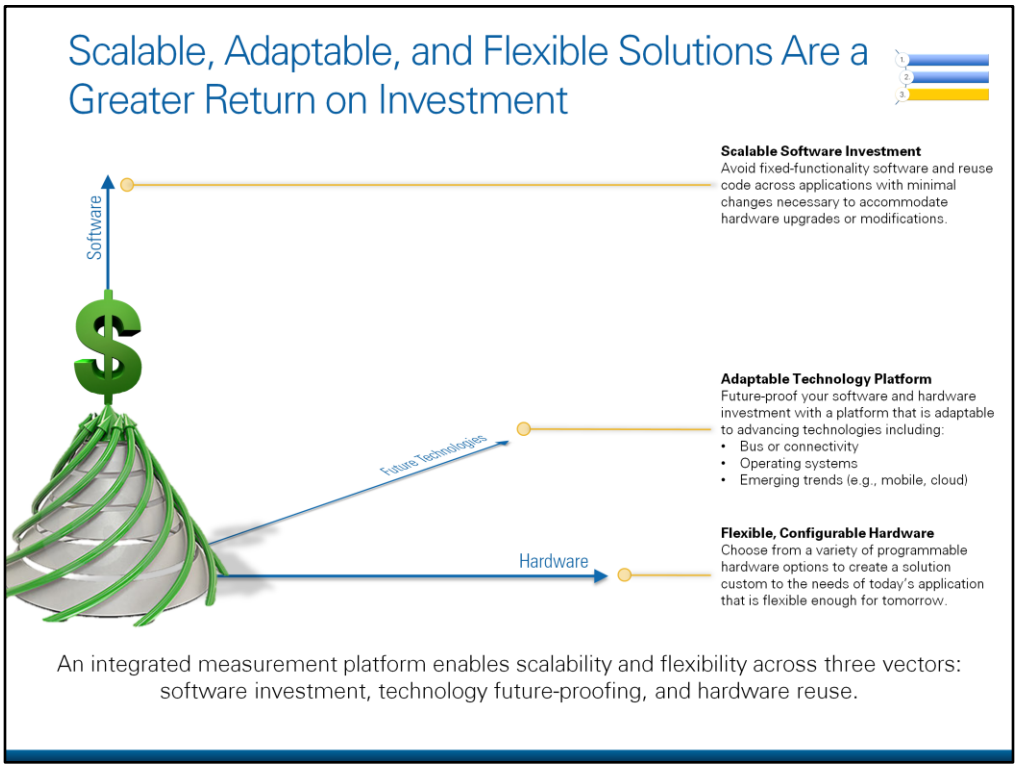
With LabVIEW, you'll be more productive because you can quickly integrate and take full advantage of an incredibly diverse range of NI and third party hardware products within a single software environment that also includes all of the analysis and UI capabilities necessary to build any measurement or control system. In addition, the graphical approach to programming provides an abstraction from low-level software complexity (such as threads and pointers) that the majority of engineers don't have time to manage, yet is powerful enough to assimilate this low-level code from traditional text-based languages in which you may have previously made a software development investment.

Each new version of LabVIEW is continually enhanced with features to further enhance and accelerate productivity – something greatly appreciated by the extensive worldwide user base.

ni.com/labview
ni.com/daq

While you can use LabVIEW to integrate third-party hardware – and most mixed-measurement systems do - there's a reason why NI DAQ hardware is the most trusted measurement solution for engineers and scientists. NI DAQ hardware delivers a wide-ranging catalog of I/O for any sensor and any bus, which is the primary reason it is ideal for systems that need to combine mixed-measurements. Industry-leading innovative hardware and driver software technologies – perfected over several decades of experience deliver absolute accuracy and proven performance that is unrivaled in the measurement industry. Lastly, NI DAQ hardware is backed up by the National Instruments brand, and features a complete product solution including extended warranties, calibration services, and technical support.

ni.com/labview
ni.com/daq



The graphical system design approach will deliver measurement solution flexibility and scalability across three vectors that are important to consider.

First, because LabVIEW integrates such a large variety of hardware platforms, your software investment scales across industries, applications, and hardware targets. Additionally, identical LabVIEW code can often be used across a number of different measurement devices with minimal or no changes, making it easy to upgrade or modify your hardware system as necessary.

Second, NI offers many measurement hardware options that are modular, making it possible to customize and configure them specific to the needs of your current application, yet reconfigure them to meet the unforeseen requirements of future applications without having to reinvest in an entirely new hardware system.

Lastly, NI measurement hardware delivers more value, performance and adaptability because it allows you to better leverage standard, commercially available PC technologies and more rapidly take advantage of the innovations in the PC industry. As advancements such as new hardware buses, PC operating systems, or technology trends occur, the platform allows you to quickly utilize and incorporate any advantages those progressions may yield.

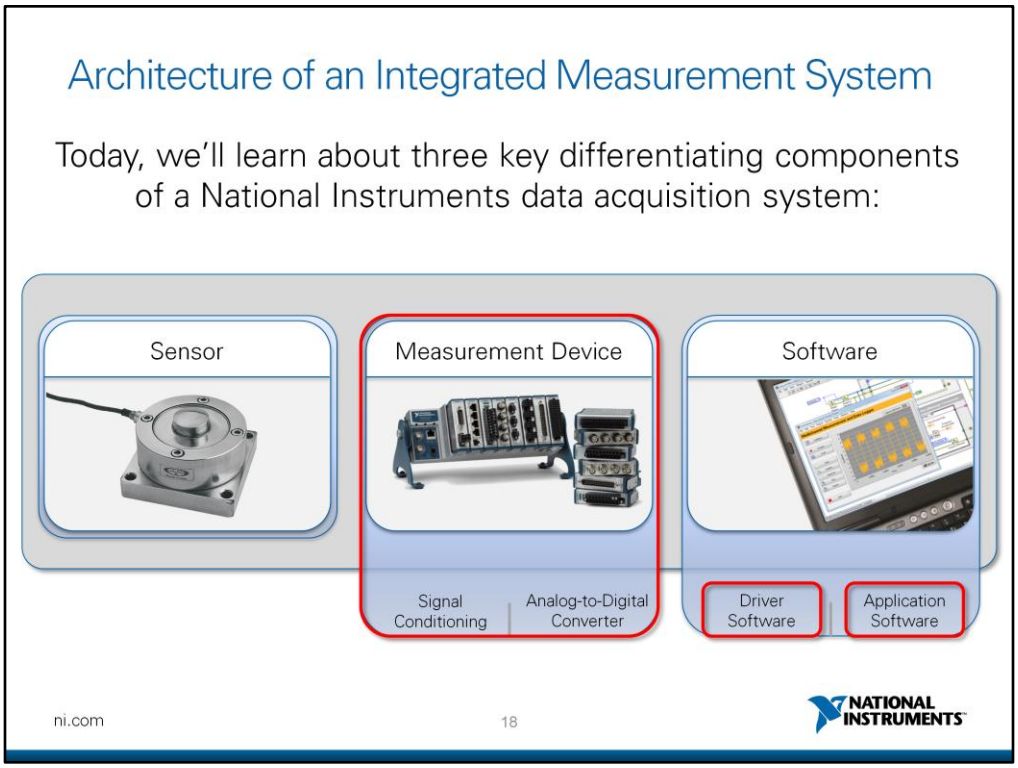
The tight integration of hardware and software is paramount to the graphical system design approach and is key to enabling the aforementioned benefits.

ni.com/labview
ni.com/daq

With the proliferation of mobile devices and tablets, there is a trending need to visualize and interact with measurement systems using mobile devices. Engineers and scientists who have built their systems around an integrated measurement platform are able to quickly and easily integrate this industry trend into their systems with the Data Dashboard for LabVIEW application, which supports creating mobile dashboards that access data from any LabVIEW application hosting network-published shared variables or web services, and can be used to augment a wide range of data acquisition, embedded monitoring, and test applications.

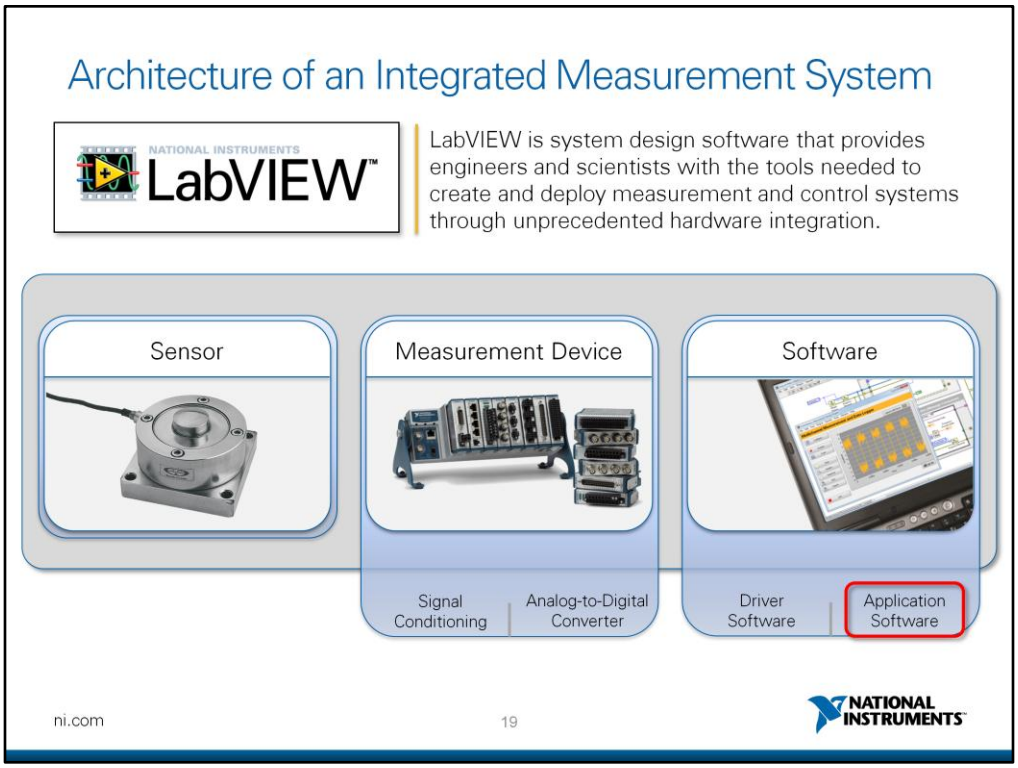
This is just one example of the scalability and flexibility that a platform-based approach to measurements provides.

ni.com/labview
ni.com/daq



The three components of a highly integrated measurement system that we'll explore for the remaining duration of today's seminar are the measurement hardware, application software, and driver software.

ni.com/labview
ni.com/daq



Before we explore the fundamentals of data acquisition, let's begin with the application software – NI LabVIEW.

ni.com/labview
ni.com/daq


Coffee Break
Enjoy Coffee and Networking
With Industry Peers

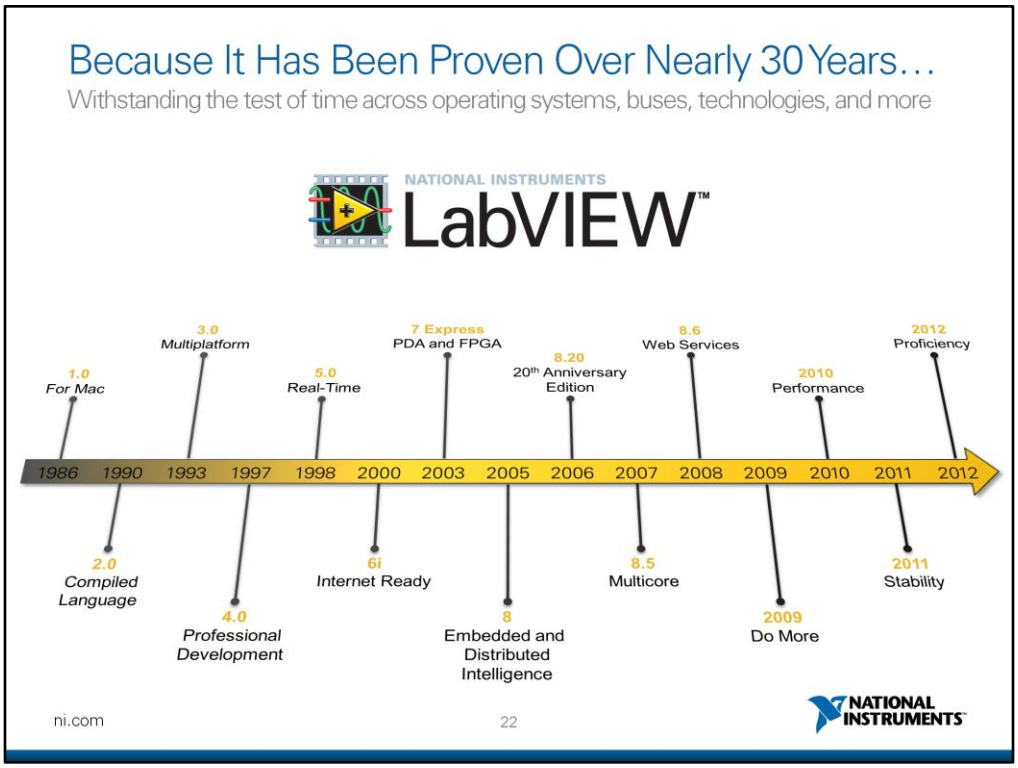
Introduction to LabVIEW

System Design Software for Any Measurement Application

ni.com

NATIONAL INSTRUMENTS

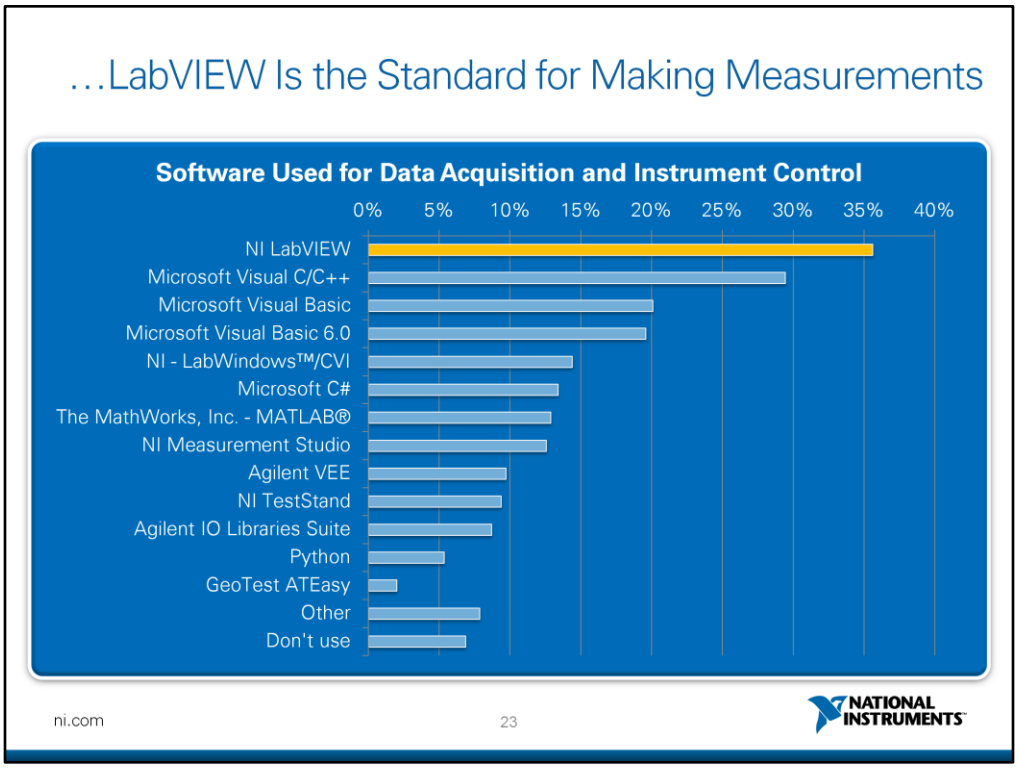
ni.com/labview
ni.com/daq



The computing industry is ever-changing. Operating systems change, processor architectures evolve, and over time, even programming languages come and go. From Fortran to Pascal, C to C++ and now C#, each new language is eventually superseded by the next.

Standing the test of time, however, is LabVIEW, which was originally created for the Macintosh nearly 30 years ago for the explicit purpose of providing engineers and scientists a better software solution for communicating with laboratory instrumentation. Throughout its history, the capabilities of LabVIEW have expanded to include a wide range of additional hardware targets and technologies, but these original tenets upon which LabVIEW was founded remain a cornerstone of the software to this day.

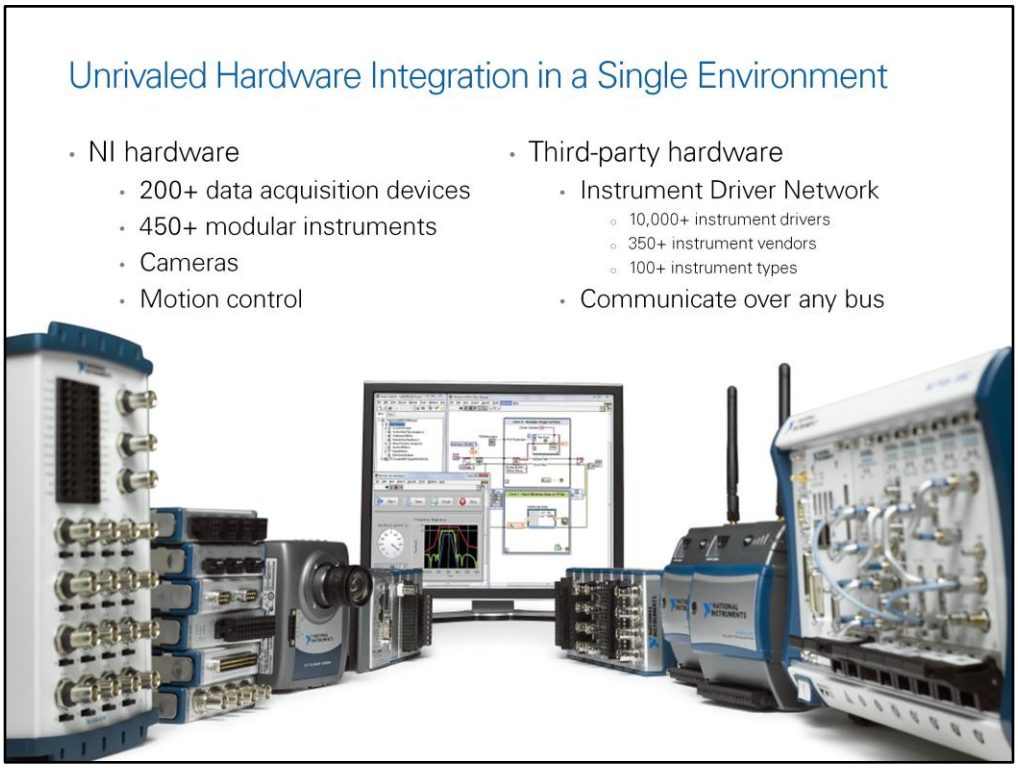
ni.com/labview
ni.com/daq



Because of its proven history, LabVIEW is literally the industry standard system design software for building automated measurement solutions, as validated by several recurring industry surveys. The vast LabVIEW user base includes engineers, scientists, researchers and students spanning across dozens of industries, with applications as simple as the LEGO® MINDSTORMS® NXT robots and as complex as the CERN Large Hadron Collider (LHC).

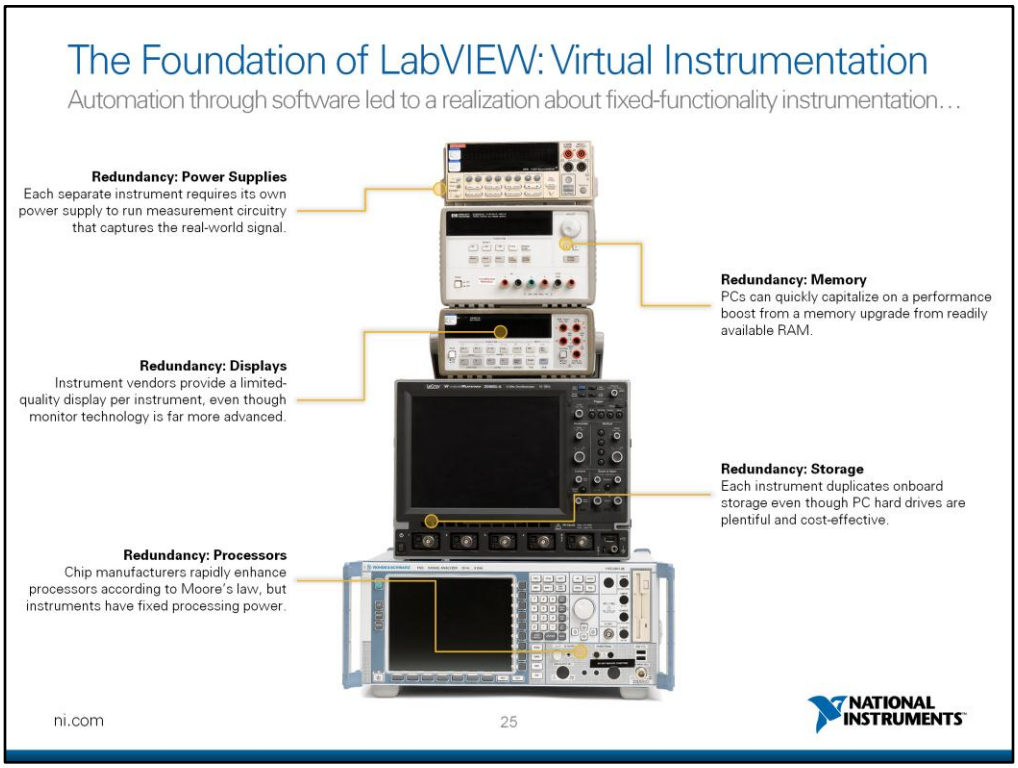
Source:
2011 US Platform Awareness Study. Question Text: "Which of the following software packages/programming languages do you use to control or acquire data from your instruments? (Multiple response question)." Note: For multiple response questions, the sum of the percentage of respondents may equal more than 100%.

ni.com/labview
ni.com/daq



The biggest reason for LabVIEW's widespread adoption is its unrivaled integration with NI and third party hardware. With each new release of LabVIEW, significant engineering investment is focused on ensuring that users have access to the latest technologies and hardware platforms. In addition to native integration with all NI hardware (including hundreds of devices for measurement, automation, control, image capture, and motion), LabVIEW supports the industry's largest and most exhaustive network of free instrument drivers for communicating with traditional hardware.

ni.com/labview
ni.com/daq

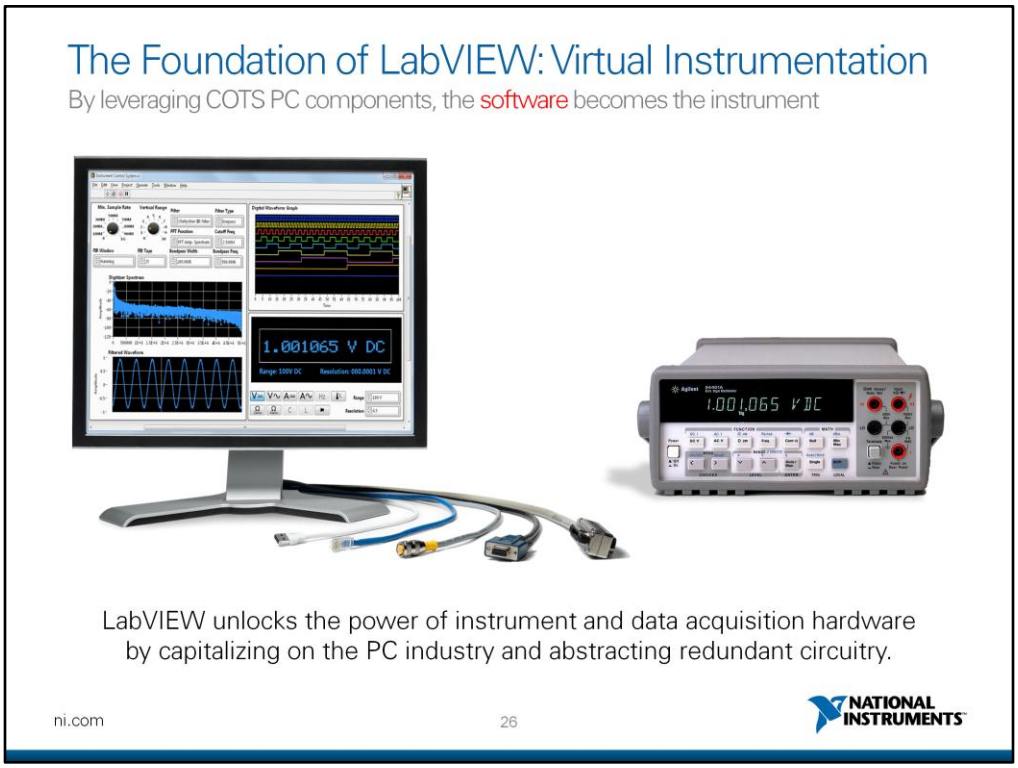


LabVIEW was originally created to communicate with traditional boxed-instruments over the general purpose interface bus (GPIB). Many of these traditional instruments are still in use today.

If you were to open the box of a traditional instrument like an oscilloscope, you would see measurement circuitry that captures the real-world signal in digital form, as well a processor, memory, power supply, display with buttons, knobs, and so on. Also inside the instrument is software that is pre-written by the vendor to make a specific measurement. This software is embedded in ROM or a hard disk, and the user cannot change it. It is a closed, fixed-function device that has limited flexibility and cannot be easily upgraded or enhanced. The essence of virtual instrumentation is the realization that the computer functionality inside these boxes is readily available in standard PCs – and is much more powerful and lower cost.

Virtual instrumentation leverages the PC to eliminate the redundant duplication of computer components inside traditional boxes, giving users a more powerful, lower cost, and more flexible solution.

ni.com/labview
ni.com/daq

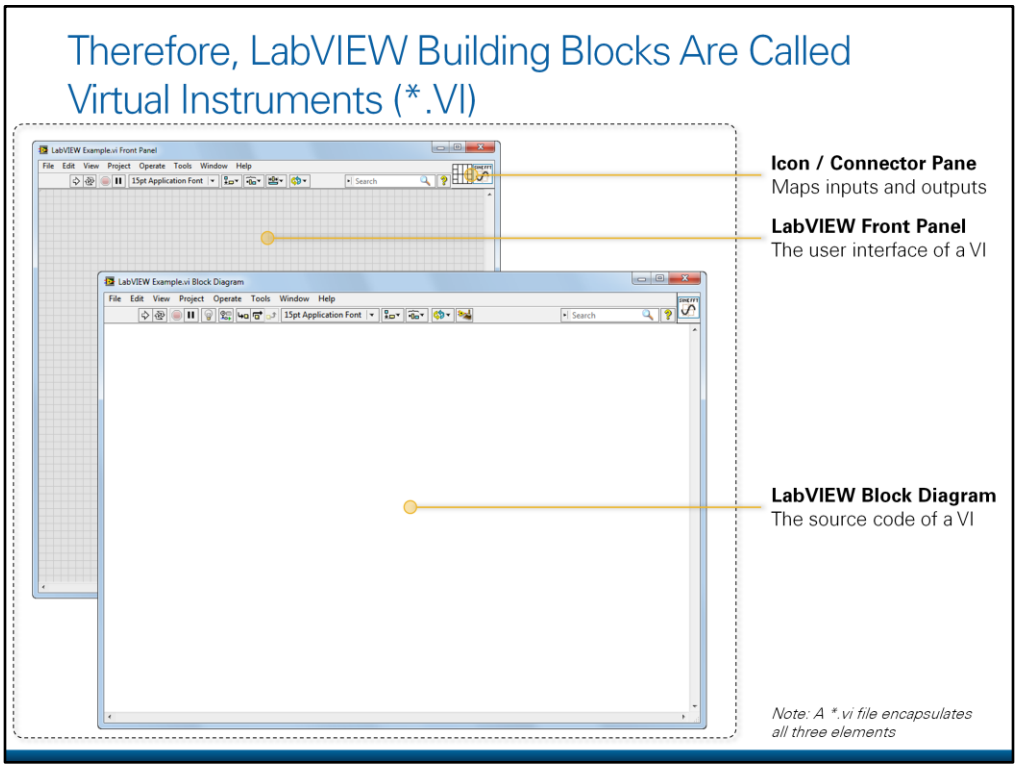


A virtual instrument consists of an industry-standard computer or workstation equipped with powerful application software, cost-effective modular hardware such as plug-in boards with appropriate driver software, the unit under test (UUT), and sensors, all of which work together to perform the functions of traditional instruments. Virtual instrumentation leverages the power of commercially available PC technology such as processors, memory, and I/O to create instrumentation tools. The virtual instrumentation paradigm transformed test, measurement, and automation applications from loosely coupled and often incompatible stand-alone instruments and devices to tightly integrated, high-performance measurement and automation systems.

More than just building the equivalent of a traditional instrument around a PC, virtual instrumentation is about *redefining what an instrument is* and empowering users to build powerful instruments and flexible measurement systems never before possible.

Virtual instruments represent a fundamental shift from traditional hardware-centered instrumentation systems to software-centered systems that exploit the computing power, productivity, display, and connectivity capabilities of ubiquitous desktop computers and workstations. Although the PC and integrated circuit technology have experienced significant advances in the last two decades, it is software that provides the leverage to build on this powerful hardware foundation to create virtual instruments, providing better ways to innovate and significantly reduce cost. With virtual instruments, engineers and scientists build measurement and automation systems that suit their needs exactly (user-defined) instead of being limited by traditional fixed-function instruments (vendor-defined).

ni.com/labview
ni.com/daq



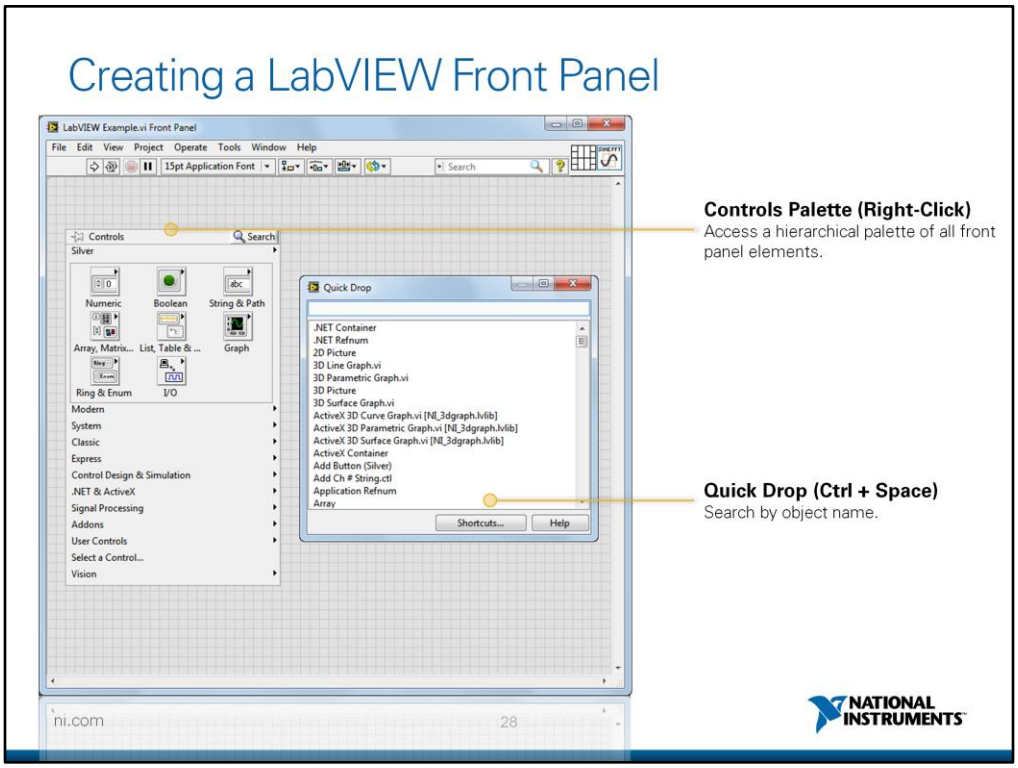
LabVIEW, specifically is the software cornerstone of a virtual instrumentation system. Its flexibility, combined with powerful modular hardware solutions, creates the ultimate in user-defined, scalable instrumentation systems. Virtual instrumentation is so fundamental to LabVIEW that the basic building block (program of code) in LabVIEW is called a Virtual Instrument, or *.vi file.

There are three important things encapsulated within the *.vi file on disk:

The LabVIEW Front Panel is the user interface of the VI.
The LabVIEW Block Diagram contains the source-code of a VI.
The Icon/Connector Pane defines the input and output parameters of the VI.

Insider Tip!

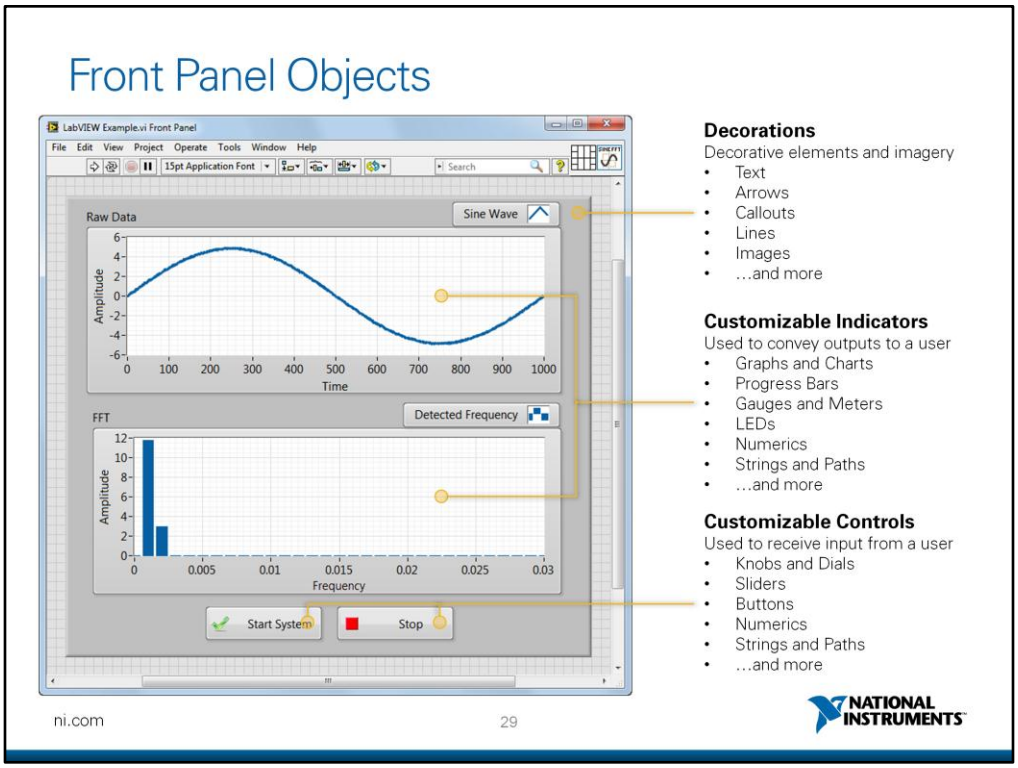
- You can arrange the front panel and block diagram to stand side-by-side by using the shortcut **<Ctrl + T>**.
- You can toggle between the front panel and block diagram by using the shortcut **<Ctrl + E>**.



There are two ways to populate a LabVIEW front panel with user interface elements.

First, by **right-clicking** anywhere on the front panel, you can access the *Controls Palette* – a hierarchically organized palette of all components that you can use to populate your user interface. Browse the Controls Palette hierarchy until you find the UI element you'd like to use, then drag-and-drop it to the appropriate location on the front panel. The Controls Palette supports customization (so that you can decide which sub-palettes are shown and the order in which they're shown) as well as pinning in order to make the palette remain visible.

If you already know the name of the UI element you're looking for, you can use *LabVIEW Quick Drop* by pressing the **<Ctrl+Space>** shortcut. This is often much faster than navigating through a series of sub-palettes to find a UI element that is nested several layers deep.



The LabVIEW front panel includes over 300 controls and indicators designed specifically for measurement applications. Each object is configurable, enabling you to create professional graphical user interfaces.

A *Control* is a front panel object for user input. Simple examples of controls include buttons, slides, dials, and text boxes.

An *Indicator* is a front panel object that displays data to the user. Examples of indicators are graphs, thermometers, and gauges.

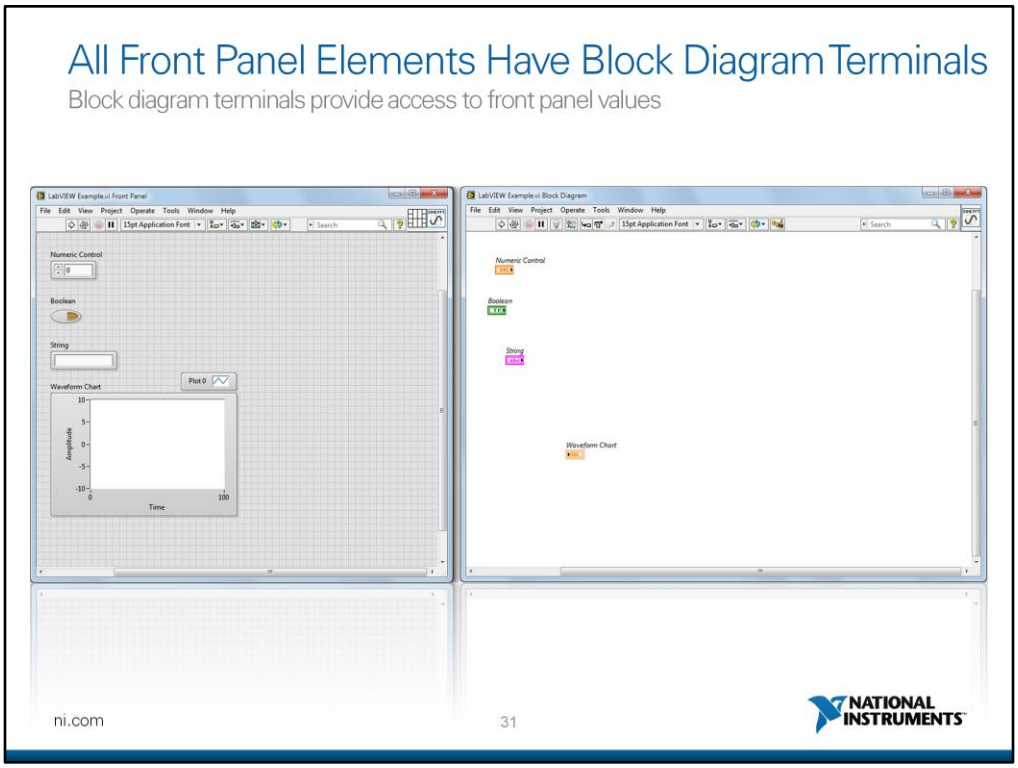
LabVIEW Front Panels in Action

Dozens of LabVIEW front panels at SpaceX Mission Control during successful launch of Dragon
Photo Credit: Elon Musk

All of the front panels above were contributed for sharing and reuse by members of the global LabVIEW community.

LabVIEW controls and indicators can be customized in order to build advanced elements for highly sophisticated, modern user interfaces. This slide shows a series impressive LabVIEW user interfaces directly from the LabVIEW user community as well as a real-world example of LabVIEW in action during the successful launch of the Dragon spacecraft by SpaceX.

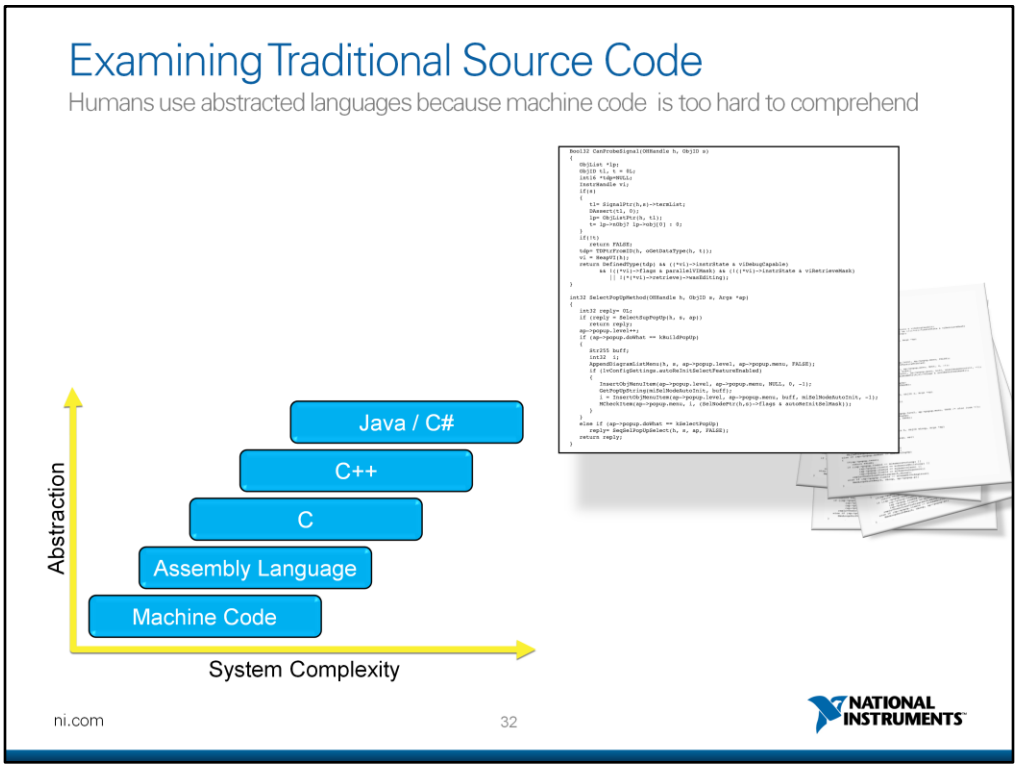
ni.com/labview
ni.com/daq



Whenever you place user interface controls or indicators on the LabVIEW front panel, you'll notice that corresponding terminals get added to the LabVIEW block diagram. Similarly, if you create controls or indicators via the block diagram, corresponding elements get added to the user interface.

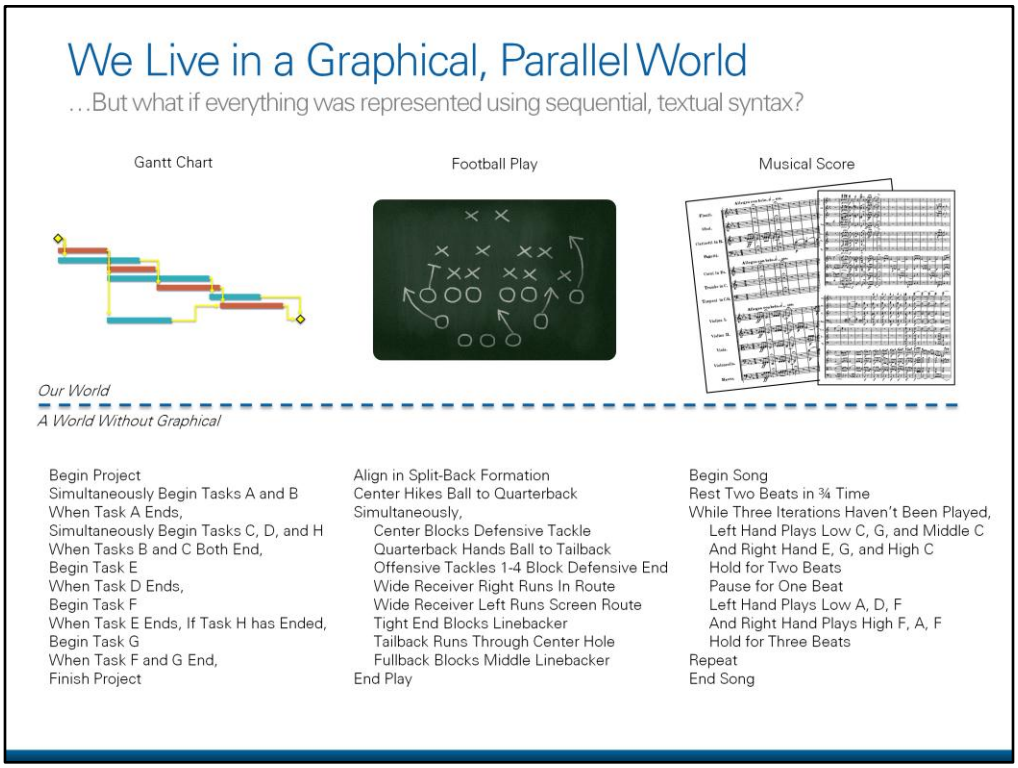
This allows you to use the source code to gain programmatic access to the data that the user has input on the user interface, or conversely to update the user interface via code.

ni.com/labview
ni.com/daq



At their lowest level, computers understand what is called machine code – a way of representing software using instruction steps written in the most basic language computers can interpret. However, it is extremely rare that you'll ever see machine code when working with modern day computers, even when writing computer software yourself. Why?

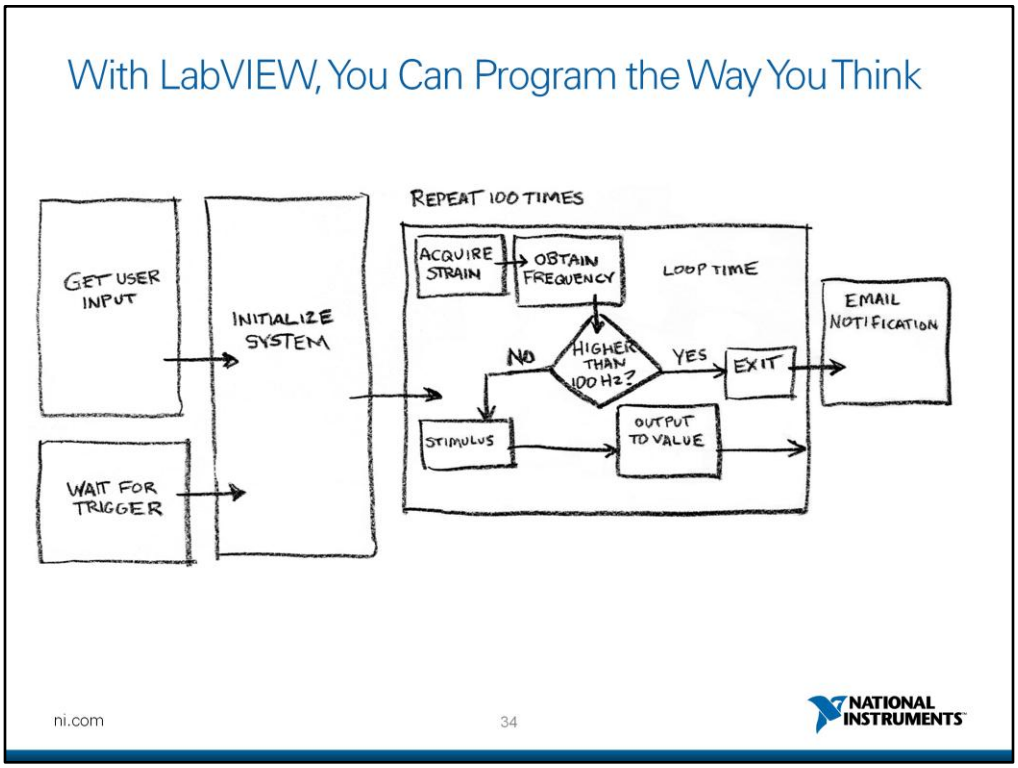
Though computers fundamentally operate in a digital manner using binary (a series of 1s and 0s), it is incredibly inefficient for humans to attempt to represent software in the same way, as humans have a much harder time interpreting binary than do the logic gates that comprise modern day processors. As a result, over time, a series of software abstraction layers have been developed that allow humans to represent software instructions in the way that makes most sense to them, yet still create lower-level code during the compilation process. It is precisely this compromise that has opened up a world of accessibility and allowed the complexity of systems to increase over time without requiring that the creators of those systems be the world's foremost computer science experts.



The abstraction of computer programming languages has been paramount to the accomplishments in modern day technology. However, even today's common abstracted programming languages like C, C#, or Java continue to represent software with a sequential, textual syntax.

If the point of software abstraction is to allow humans to better interpret and represent computer software in a way that they can understand – else we would all be writing code using machine language – then why do we continue to expect programmers to use a sequential textual representation of software even though we live in a graphical, parallel world?

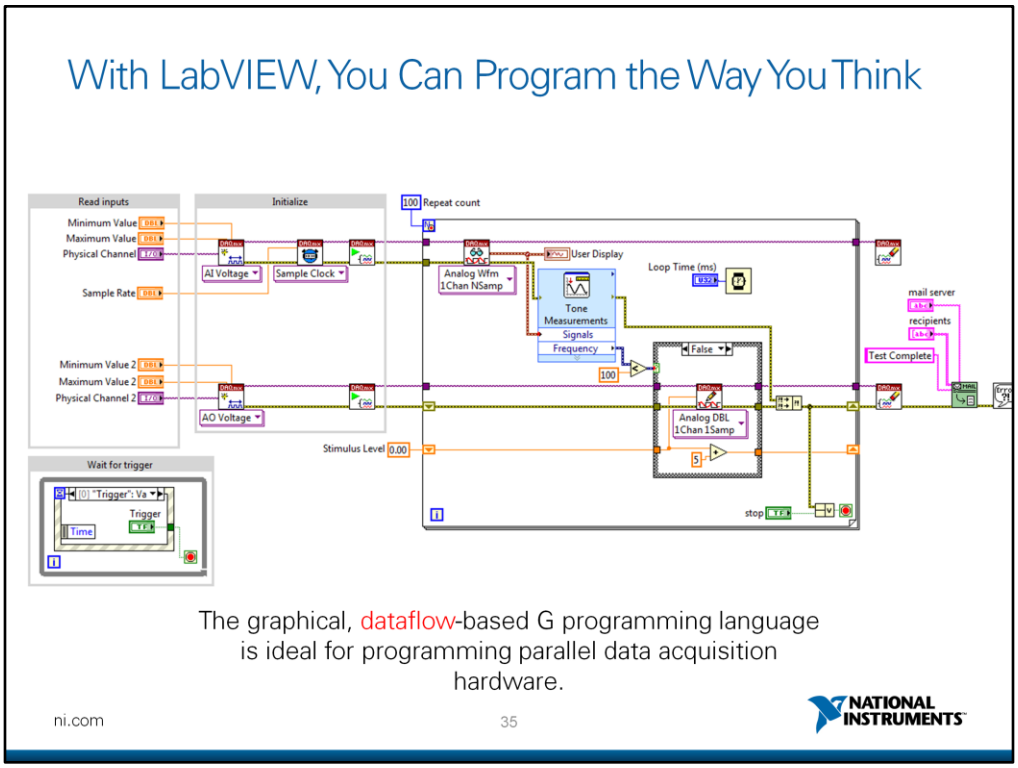
Consider the way we commonly represent the flow of everyday actions in our lives. Each are graphical in nature, which allows us to implicitly specify parallelism without having to make any extra effort. Gantt charts are timing diagrams used by project managers to represent the execution of project tasks, football coaches represent the coordination of a particular play using graphics, and even the language of music allows us to harmonize sounds from multiple instruments at once, complete with timing, synchronization, and a construct for specifying repetition! The sequential, textual equivalents of these graphical representations are far more difficult to understand.



Programming in LabVIEW is fundamentally built off of exactly this ideology. If we as engineers and scientists can program the way we think, we'll be able to more easily create complex systems without requiring the vertical expertise of a computer scientist. The majority of LabVIEW users are domain experts that prefer to focus on *what* they're trying to measure, not *how* they're going to measure it. The tools that comprise the measurement system should not stand in your way.

If we were to quickly diagram a flowchart that represents the steps of a system, the object of which was to measure strain, apply a stimulus, and email test results at the conclusion of the measurement, it might look something like this.

ni.com/labview
ni.com/daq



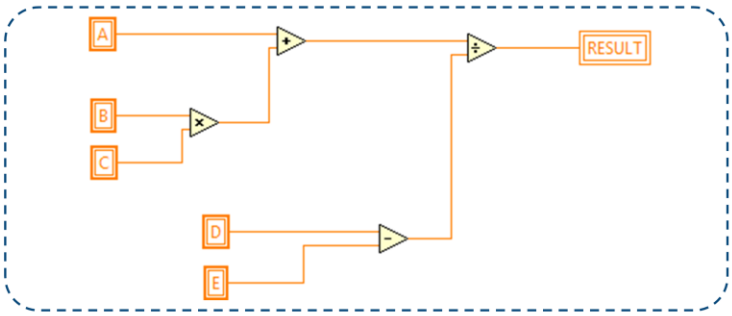
The first thing you'll notice when examining LabVIEW source code on the block diagram is that it is graphical in nature. This allows you to program the way you think. In fact, the exact LabVIEW code required to implement the system from our previous flowchart is displayed on the slide. Suitably, the graphical language used within LabVIEW is called *G*.

Just like a flowchart represents the flow of data through a system, representing code graphically makes use of this concept as well. In fact, it is called precisely that – dataflow.

ni.com/labview
ni.com/daq

What Is Data Flow?

- Each block diagram node executes only when it receives all inputs
- Each node produces output data after execution
- Data flows along a path defined by wires
- The movement of data determines execution order



Formula: Result = (A+B*C) / (D-E)

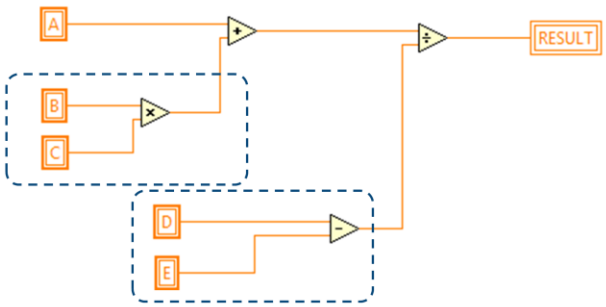
There are four fundamental tenets to dataflow.

Every item that you see on the block diagram cannot execute until it receives all of its required input data. After any node successfully receives all inputs, it will execute and produce output data. Typically, data will flow through a LabVIEW block diagram from left to right; therefore, LabVIEW nodes represent their input terminals on their left and their output terminals on their right.

As data flows through a block diagram, it does so using wires – much like real-world wires allow electricity to flow through a circuit. Taking into account each of these facts, it is a logical step to understand that the movement of data will be the determining factor in the order of execution of the source code.

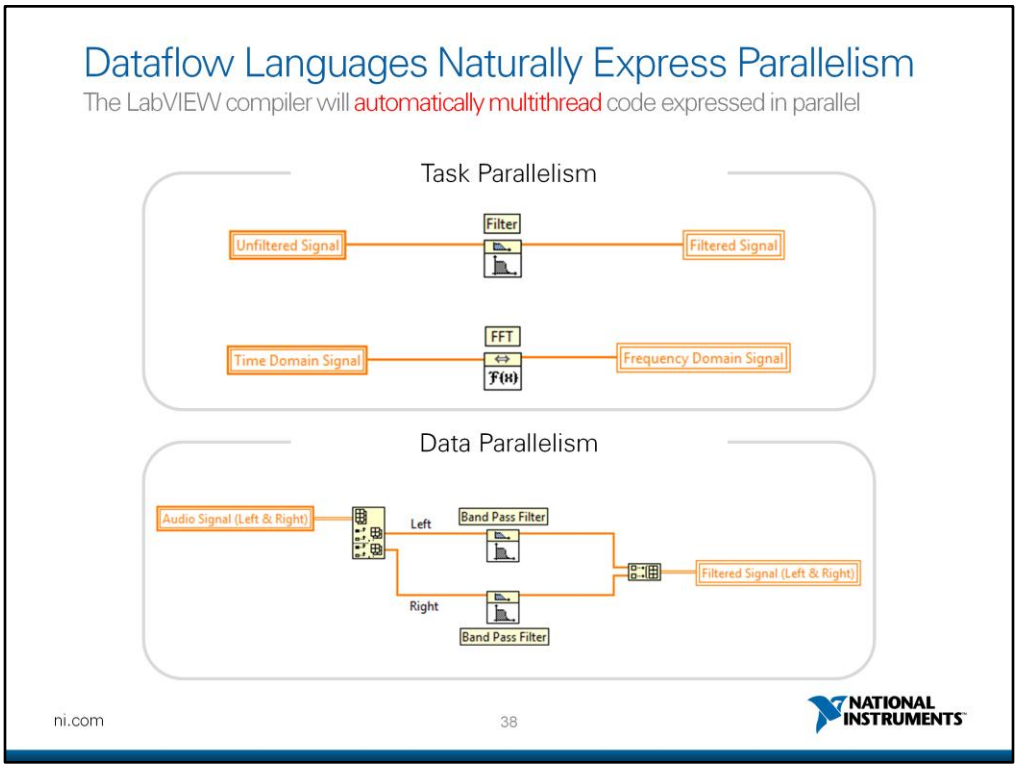
What Is Data Flow?

- Each block diagram node executes only when it receives all inputs
- Each node produces output data after execution
- Data flows along a path defined by wires
- The movement of data determines execution order



The [Multiply] and [Subtract] operations can execute at the same time since they don't have any data dependencies.

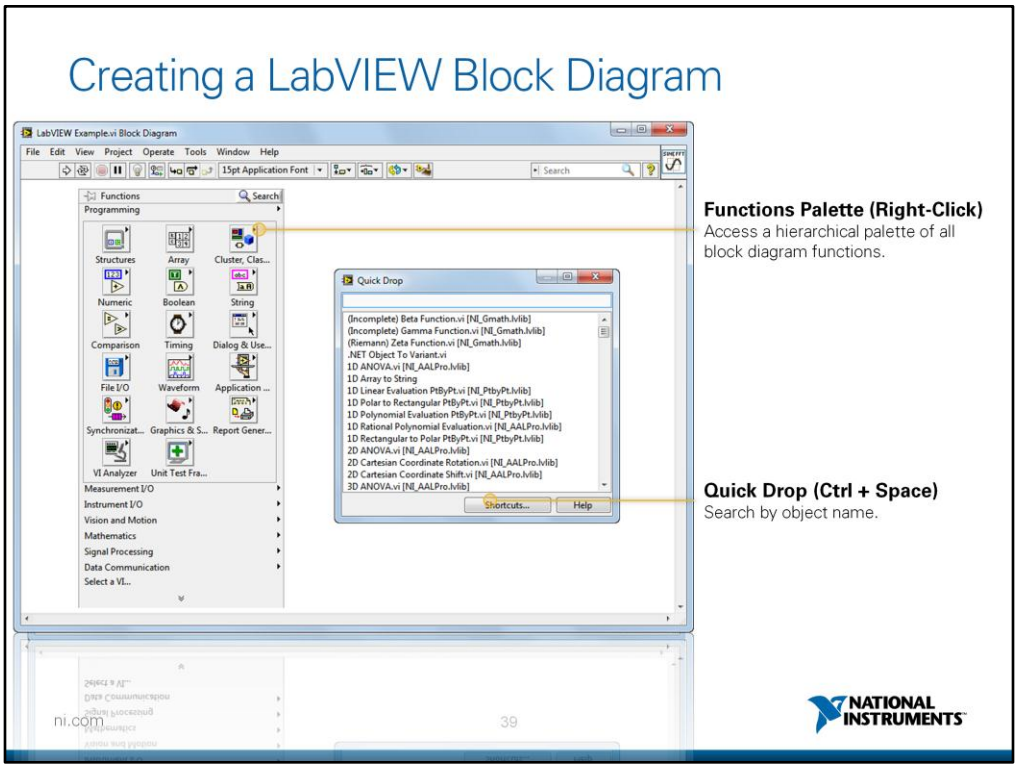
Given the fact that a node can execute as soon as it receives all of its required inputs, multiple operations in LabVIEW can be concurrently executed so long as there are no data dependencies between the nodes. This exceptionally advantageous trait of graphical programming means that we can express parallelism in code naturally and logically, without having to be experts in in thread spawning from thread pools like traditional computer scientists have to be.



G is a compiled language, which can be surprising because during typical LabVIEW development, there is no explicit "compile" step. Instead, you make a change to your VI and simply press the 'run' button to execute it. As we discussed earlier, humans understand G code but the computer doesn't, so LabVIEW has to send your code through the compiler, which interprets it, optimizes it, and generates the machine code that your CPU can understand and act upon.

One of the many benefits of using LabVIEW is that much of the administrative complexity of computer programming such as memory management, language syntax, and multithreading of parallel operations is automatically abstracted away by the compiler. In fact, LabVIEW abstracts a great deal of artificial and unnecessary complexity created by other programming tools, enabling higher productivity and faster development for automated measurement applications.

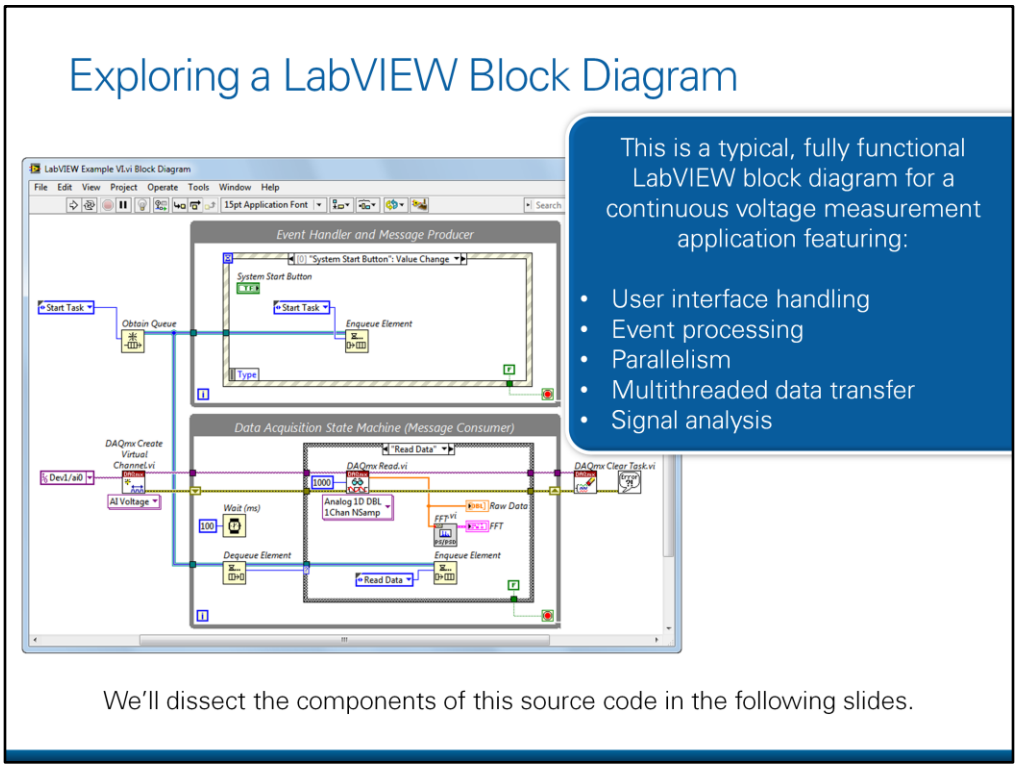
ni.com/labview
ni.com/daq



There are two ways to populate a LabVIEW block diagram with code.

First, by **right-clicking** anywhere on the block diagram, you can access the *Functions Palette* – a hierarchically organized palette of all components that you can use to populate your block diagram. Browse the Functions Palette hierarchy until you find the function you'd like to use, then drag-and-drop it to the appropriate location on the block diagram. The Functions Palette supports customization (so that you can decide which sub-palettes are shown and the order in which they're shown) as well as pinning in order to make the palette remain visible.

If you already know the name of the function you're looking for, you can use *LabVIEW Quick Drop* by pressing the **<Ctrl+Space>** shortcut. This is often much faster than navigating through a series of sub-palettes to find a function that is nested several layers deep.

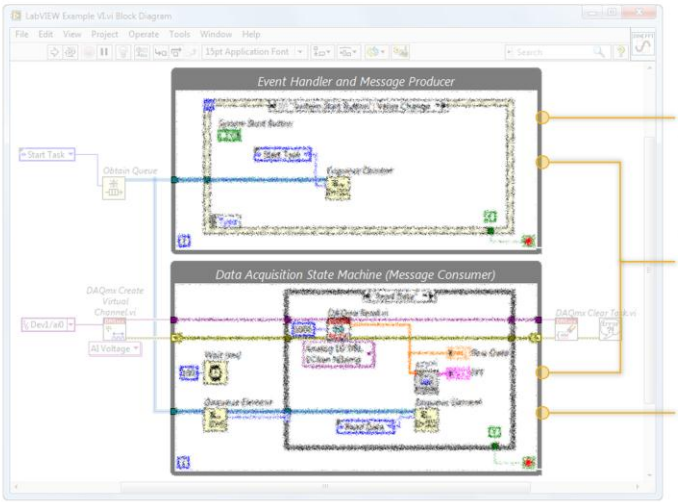


Now that we both understand the fundamental ways in which source code on a LabVIEW block diagram differs from traditional text-based code and understand how to populate a block diagram with functions, let's take a look at the elements of a block diagram by reverse-engineering the source code for a very common continuous measurement application.

The application on the slide is built using a common LabVIEW architecture called a Producer-Consumer pattern, a popular design applicable to many standard measurement applications.

ni.com/labview
ni.com/daq

Exploring a LabVIEW Block Diagram



UI Thread
This thread handles events from the front panel (user interface).

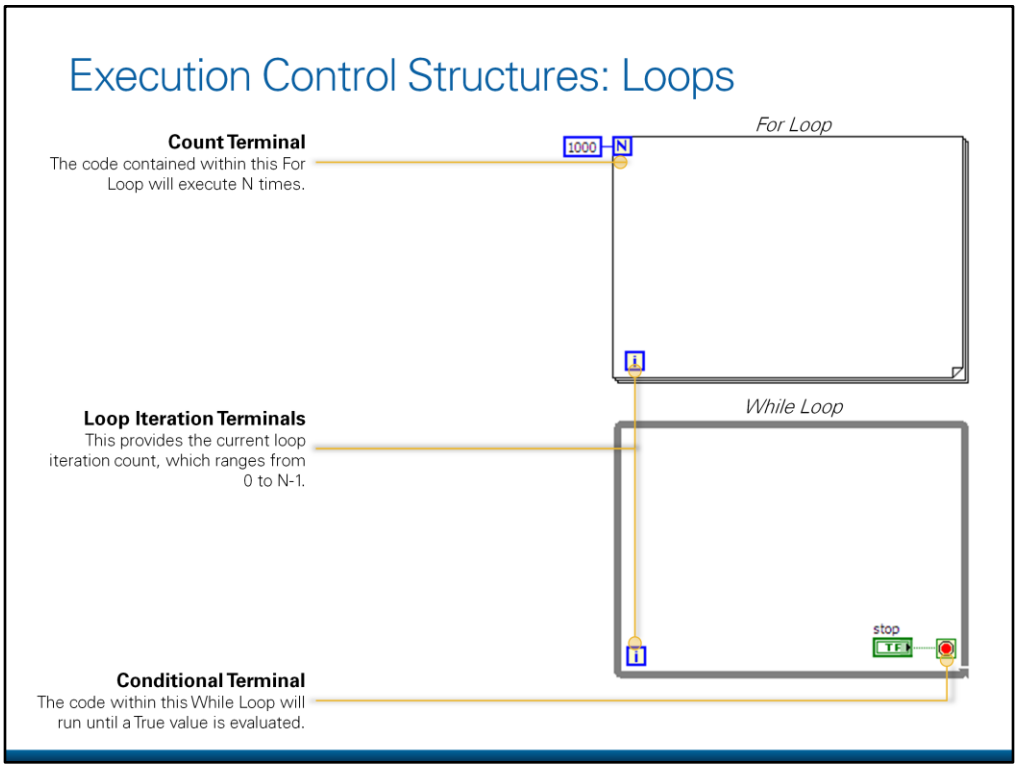
Two Asynchronous Threads
Because no data is passed between entities, they execute in parallel.

DAQ Thread
This thread interacts with data acquisition hardware.

Any block diagram entity that can contain code within it is called a structure.

If we momentarily abstract all of the lower-level details, the first thing to notice about this source code at a higher-level is that it has two concurrently running processes. We can determine this because the larger "structures" on the slide have no data dependencies – in other words, wires never exit one structure (as an output) and enter another (as an input). As we already learned, this means that the LabVIEW compiler will automatically multithread the execution of whatever code is within these elements.

The Producer-Consumer architecture is aptly named, because one process typically "produces" information and one "consumes" information. In the design of this application, the producer process reacts to interactions with the user interface. This generates commands – for example, when a button is clicked to start the acquisition of data – for the consumer process to handle.

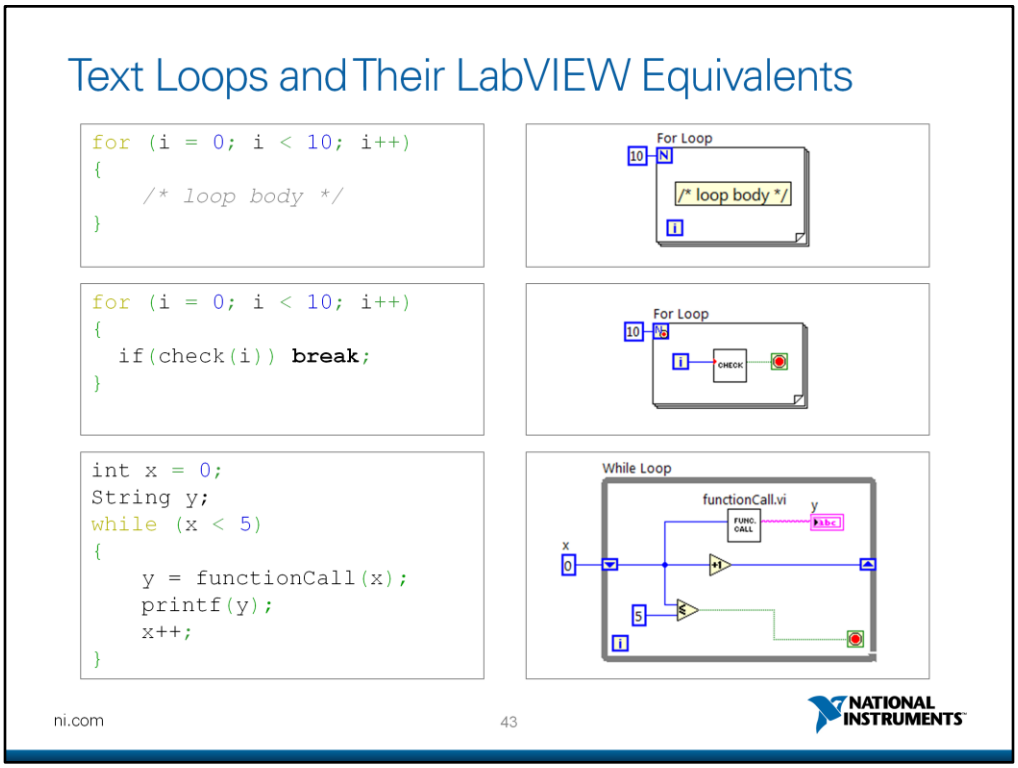


The concurrently executing processes we examined on the previous slide contain special structures called *While Loops*. A While Loop is a standard programming construct that allows a developer to specify that a section of code should run repeatedly while a certain condition has not been met. For example, if you wanted to measure a signal until that signal's measured value exceeds a threshold, you would like use a While Loop, because you wouldn't know how long it would take for the signal to exceed the specified threshold (if ever).

LabVIEW includes several execution control structures. Occasionally, you want to run a section of code a specified number of times. This is the perfect application of a traditional *For Loop*.

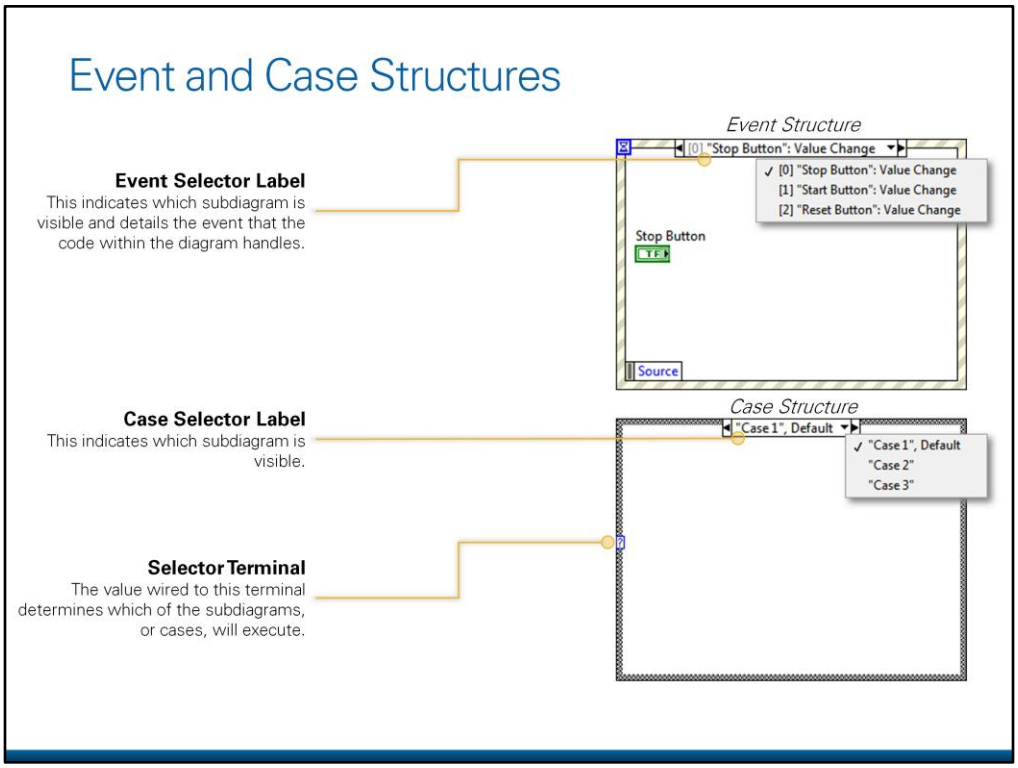
Data can flow into and out of any LabVIEW structure via a wire, which will create an input or output terminal on the border of the structure.

You can find the While Loop and For Loop structures on the block diagram Functions Palette under **Programming » Structures**.



If you have text-based programming experience, you're probably already comfortable with the concept of For and While Loops. Since G is a programming language, each of the traditional loops you'll find in text-based languages can be represented using LabVIEW.

One interesting aspect of graphical programming is that loops actually exist as structures on the block diagram that encapsulate the code that will be repeated. When the complexity of the code executed within a loop increases, this clear delineation comes in handy, particularly when contrasted with text-based equivalents.

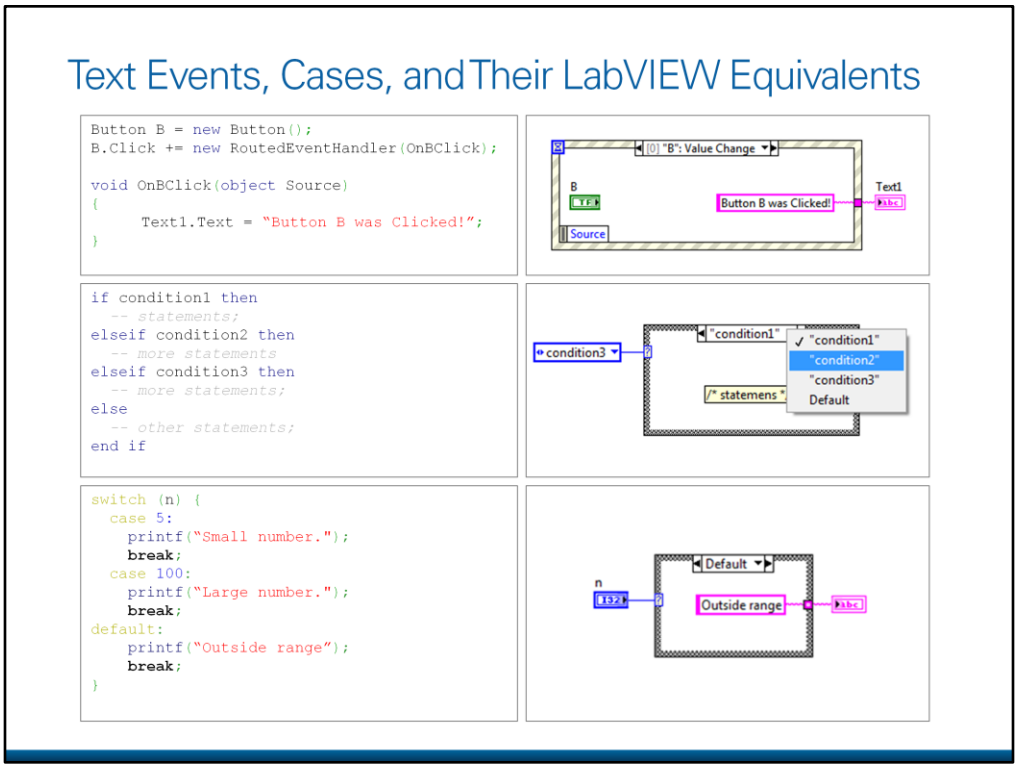


Additional LabVIEW structures allow you to react to the occurrence of certain events or conditionally execute code based upon a particular condition. The *Event Structure* is regularly used within most LabVIEW applications that feature an interactive user interface. With the Event Structure, you can programmatically react to events that your application may generate. Example events may include a user clicking a start or stop button or right-clicking on a graph.

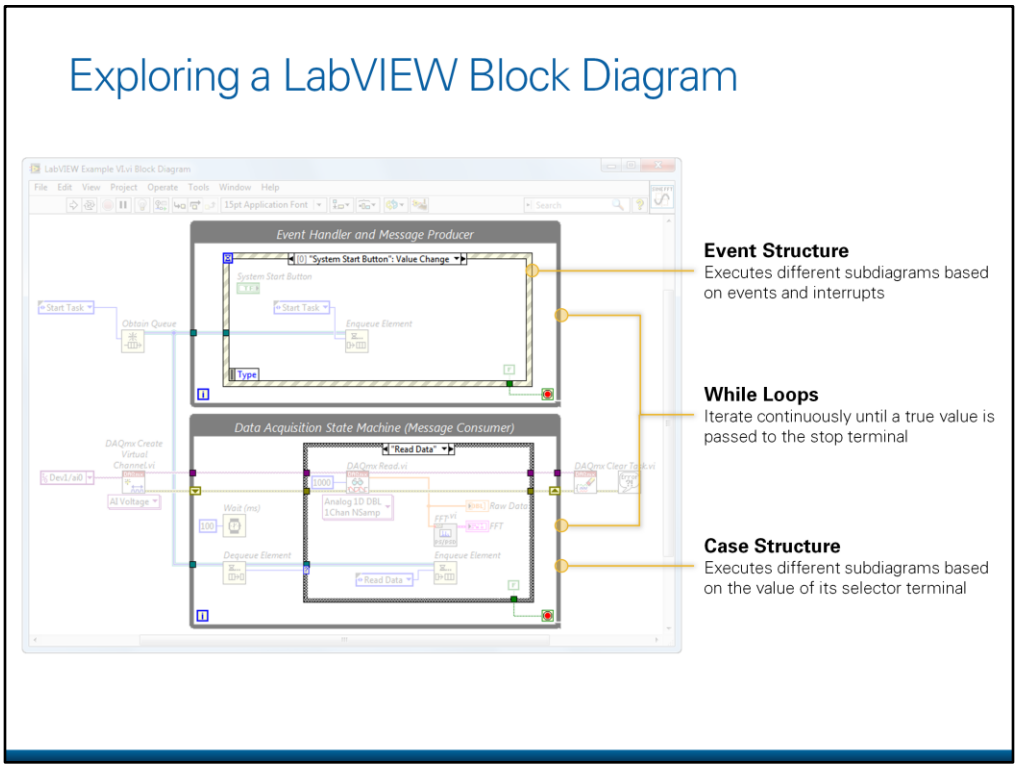
You can use the *Case Structure* to specify multiple source code alternatives and programmatically determine which selection is executed at run-time based upon a condition of your choosing. For example, if you want to execute a diagnostics test depending on whether or not an error is detected during execution, you might use a Case Structure.

Data can flow into and out of any LabVIEW structure via a wire, which will create an input or output terminal on the border of the structure.

You can find the Event Structure and Case Structure on the block diagram Functions Palette under **Programming » Structures**.

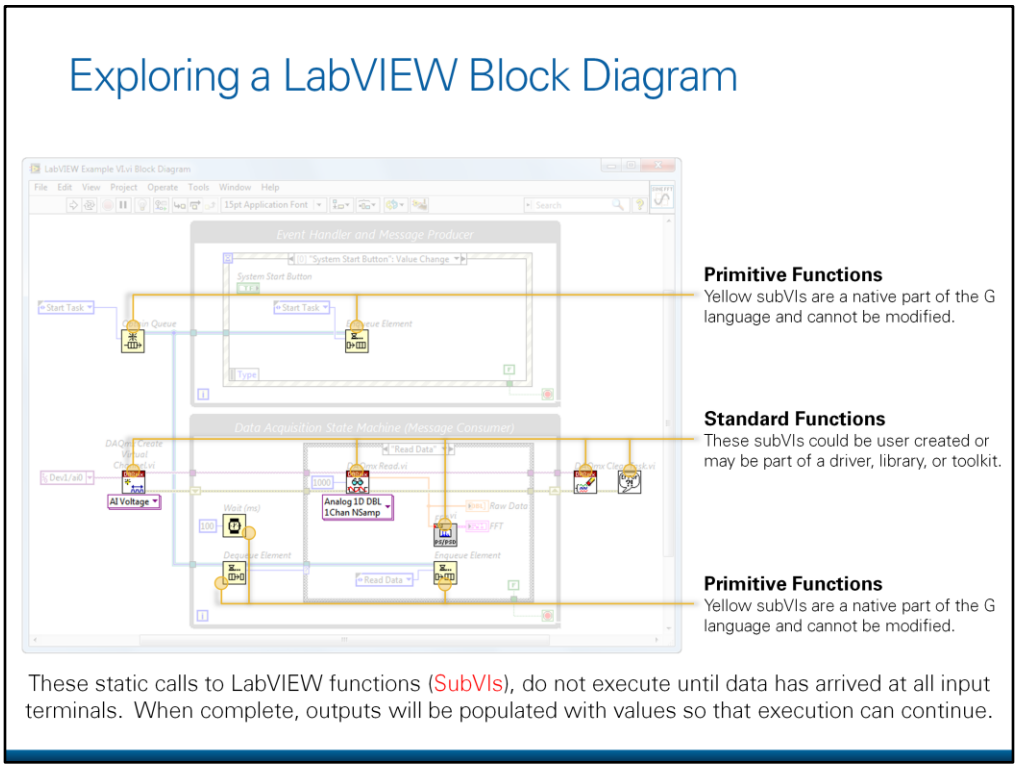


If you have text-based programming experience, you're probably already comfortable with the concept of event-based programming and logical branching. In many text-based languages, there are several alternatives to creating logical branches, including the *if* statement, the *if…then* statement, the *if…then…else* statement, and the *switch case*.



On our example block diagram, we can now understand that each of the concurrently executing processes is built using LabVIEW programming structures. Both processes contain a While Loop, because we want to execute each process continuously until the user presses a stop button.

The producer process contains an Event Structure, because it is designed to react to the user's interactions with the front panel in order to command the consumer process. The consumer process subsequently contains a Case Structure because it needs to execute different code depending on what commands it is sent by the producer process.

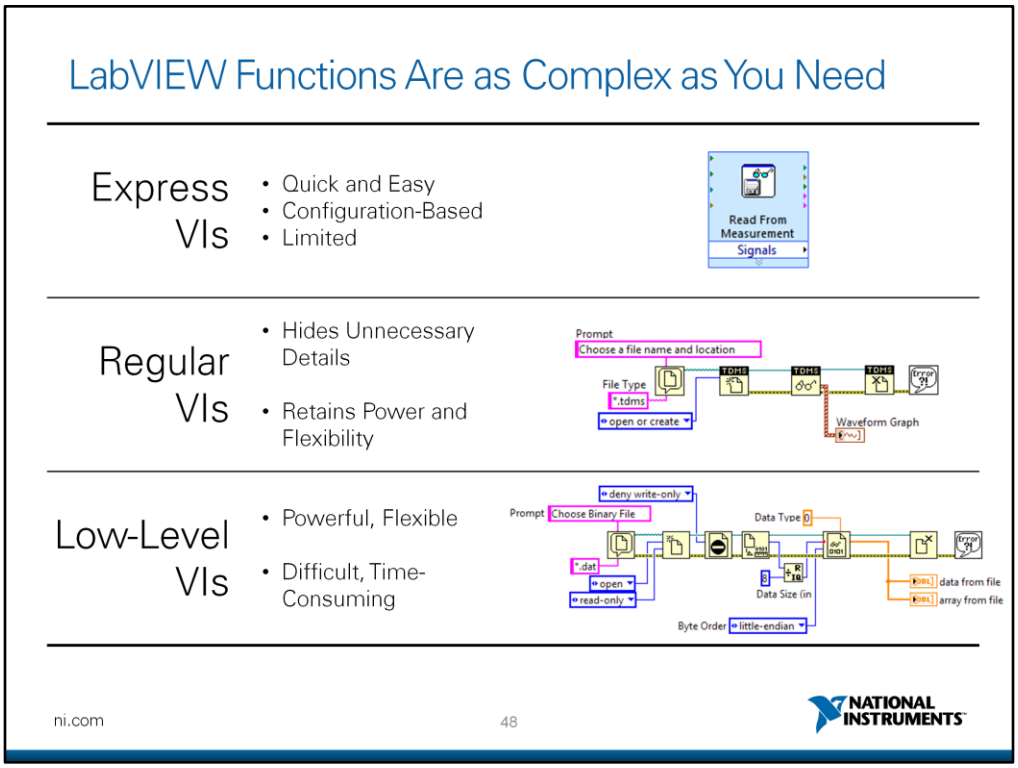


Traditional text-based programming languages encapsulate units of code within functions (or methods), and each function can call other functions (or methods) while passing that function data parameters and potentially receiving data back in return. Naturally, this concept applies to LabVIEW as well; the only difference is that a piece of LabVIEW code that is called from a LabVIEW Virtual Instrument (VI) is called a SubVI.

There are two different categories of LabVIEW SubVIs:

Primitive SubVIs – a native part of the G programming language that cannot be modified or edited by the user and therefore do not have their own front panels and block diagrams – are easy to recognize because of their yellow hue.

Standard SubVIs typically will not appear yellow in color, but are difficult to aesthetically describe because they are simply VIs in and of themselves and therefore contain fully customizable icons. These functions also have their own front panels and block diagrams, which can be accessed by **double-clicking** on the SubVI. You'll encounter a plethora of standard SubVIs within LabVIEW, as they are often user-created or used as part of a driver, library, or toolkit.



In order to ensure your productivity as a LabVIEW developer without sacrificing flexibility, LabVIEW often contains VIs with varying levels of complexity that can be mixed-and-matched to complete an application.

Depending on your requirements and level of skill, you can build a fully-functional measurement application in LabVIEW using configuration-based Express VIs, which allow you to configure the pertinent details of their operation with easy-to-use interactive dialogs, but may not expose all of the lower-level detail you need.

Low-Level VIs, which are at the opposite end of the spectrum as Express VIs, allow you complete control and granularity over all aspects of their execution.

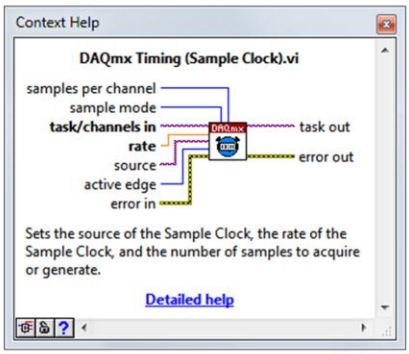
Regular VIs – where LabVIEW developers often spend most of their time – are a compromise between Express VIs and Low-Level VIs in both usability and functionality.

There is absolutely nothing wrong with building an application entirely out of Express VIs, as long as they meet your needs.

ni.com/labview
ni.com/daq

Understanding SubVI (Function) Behavior

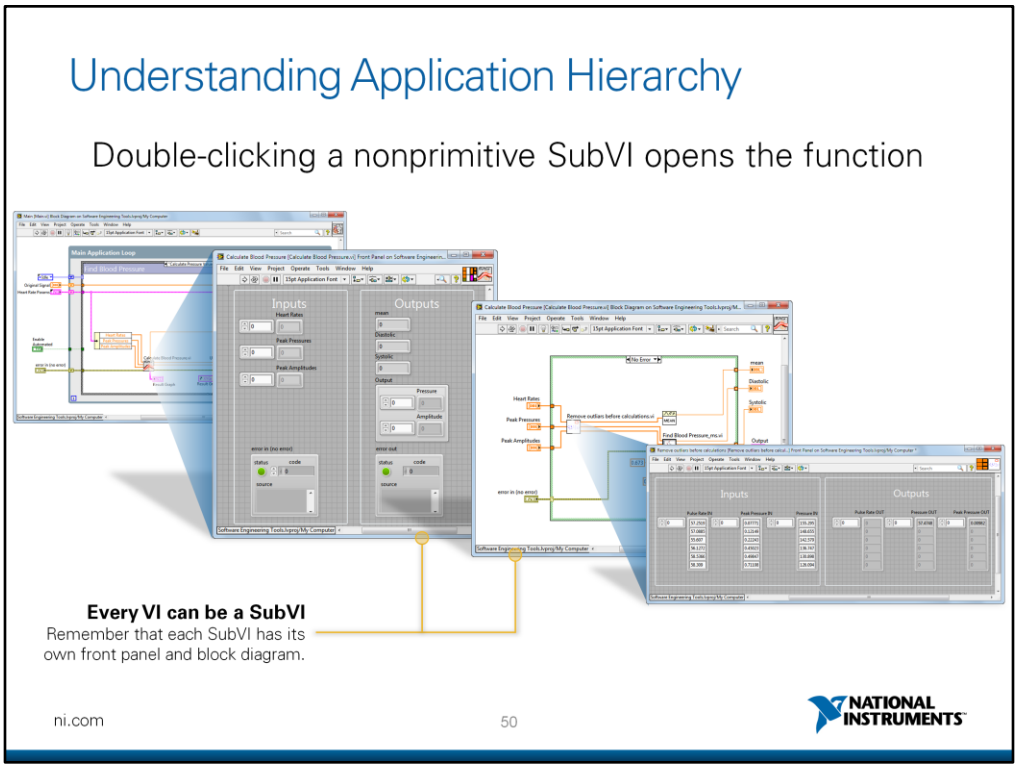
- Code will only compile if required inputs are wired
- Required inputs are **Bold**
- If an optional input is not supplied, a default value will be used for execution



*Tip: Access the Context Help using **Ctrl+H***

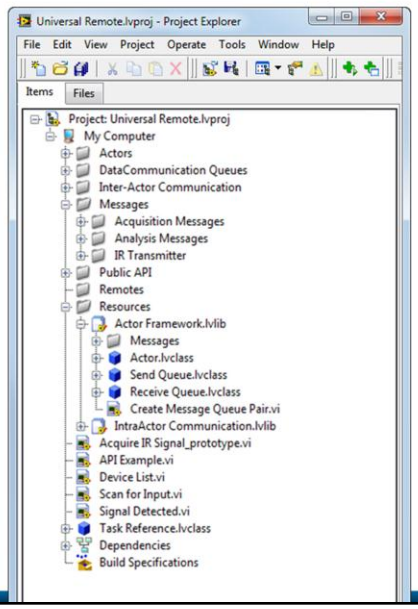
Data can be passed into SubVIs as input parameters and returned from SubVIs as output parameters. Following dataflow, inputs are typically defined on the left side of a SubVI, whereas outputs return on the right. You can configure the inputs and outputs of your own VIs by using the Icon/Connector Pane.

Not all inputs are required; to determine which inputs are and are not mandatory, access the Context Help window by pressing **<Ctrl + H>** and hover over the SubVI's icon on the block diagram. The Context Help window will contextually update depending on the position of your mouse, and will also describe the high-level purpose of the SubVI, give you an overview of all inputs and outputs, and link you directly to the LabVIEW Help for more detailed information.



Unless a LabVIEW application is particularly simple in its design, there's a good chance that it is constructed using a series of VIs and SubVIs. To traverse an application's call chain and access the underlying code or user interfaces of called SubVIs, double-click the SubVI's icon on the block diagram. Each SubVI can call other SubVIs, so you may find that your application consists of a number of layers of VI hierarchy.

Managing Application Resources in Larger Applications



LabVIEW Project Explorer

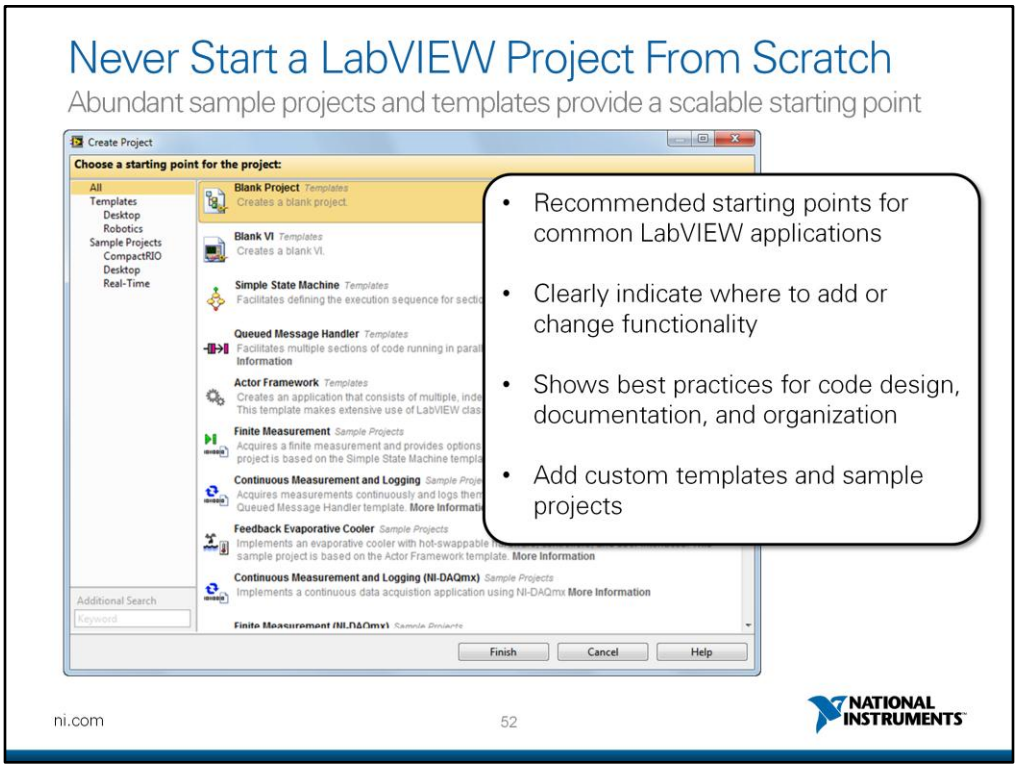
- Organize application resources
- Create classes and libraries
- Build executables and installers
- Define access scope of components
- Create and manage hardware deployment targets

51

NATIONAL INSTRUMENTS

Because complex applications could contain dozens or hundreds of VIs in addition to help documentation, imagery, custom user interface components, hardware deployment targets and numerous other dependencies, LabVIEW includes a *Project Explorer* window that will help you organize and maintain all application resources.

Insider Tip!
Even if you expect that your LabVIEW application will be simple, you should always start by creating a LabVIEW Project. You never know when your application will scale in complexity later, in which case you'll be glad you organized your work from the start.



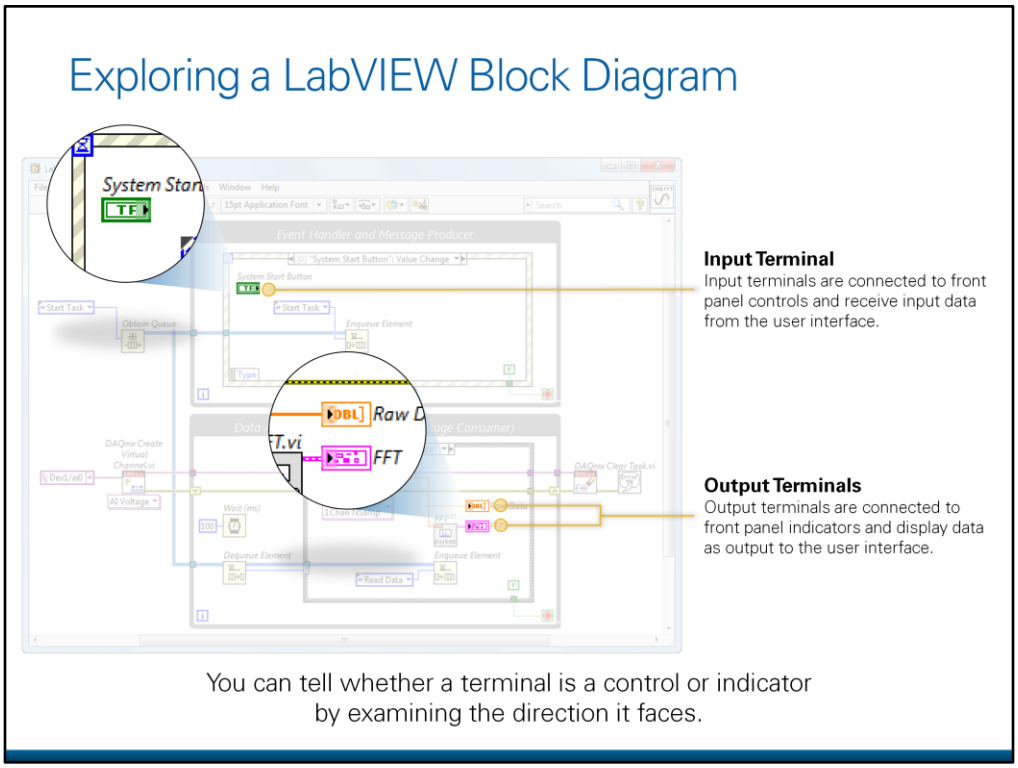
LabVIEW includes a variety of templates and sample projects, which provide recommended starting points designed to ensure the quality and scalability of a system. All of the templates and sample projects are open-source and include extensive documentation designed to clearly indicate how the code works and the best practices for adding or modifying functionality. In addition to demonstrating recommended architectures, these projects also illustrate best practices for documenting and organizing code.

What are the differences between Templates, Sample Projects, and Examples?

A LabVIEW **Template** provides a commonly used project, code, documentation, and folder structure. It may or may not also demonstrate a common LabVIEW design pattern.

A LabVIEW **Sample Project** is a working application that is a starting point for customization. Most often, sample projects are based upon templates.

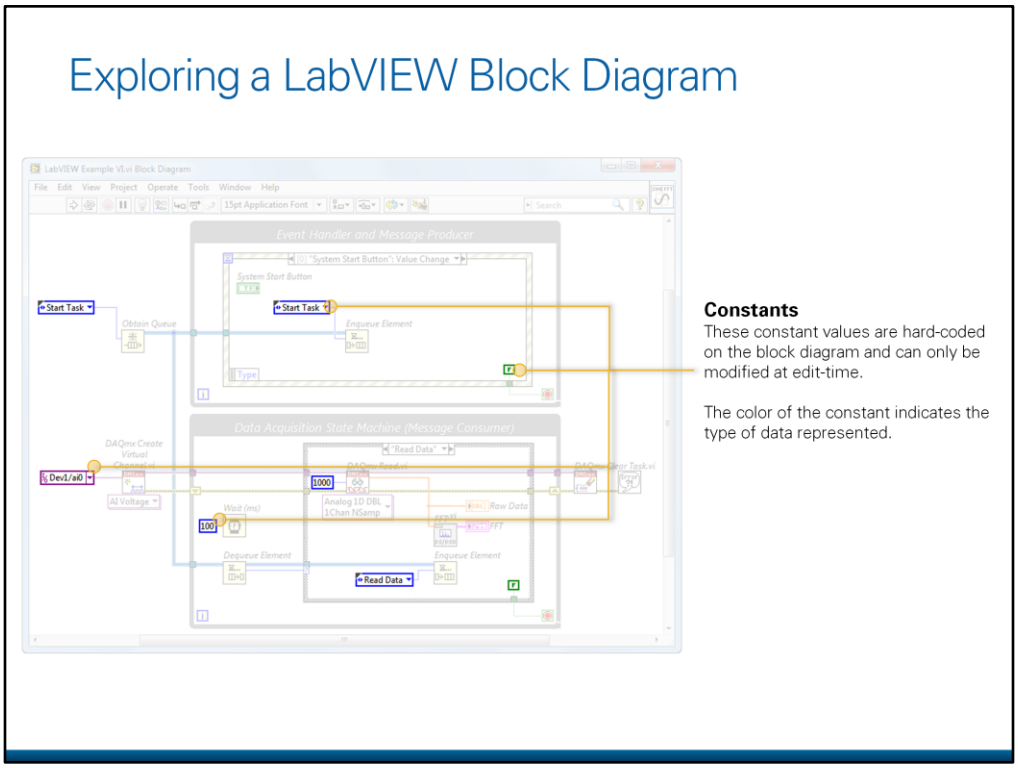
A LabVIEW **Example** (in Example Finder) is a conceptual illustration demonstrating how to use a low-level API or technology within LabVIEW



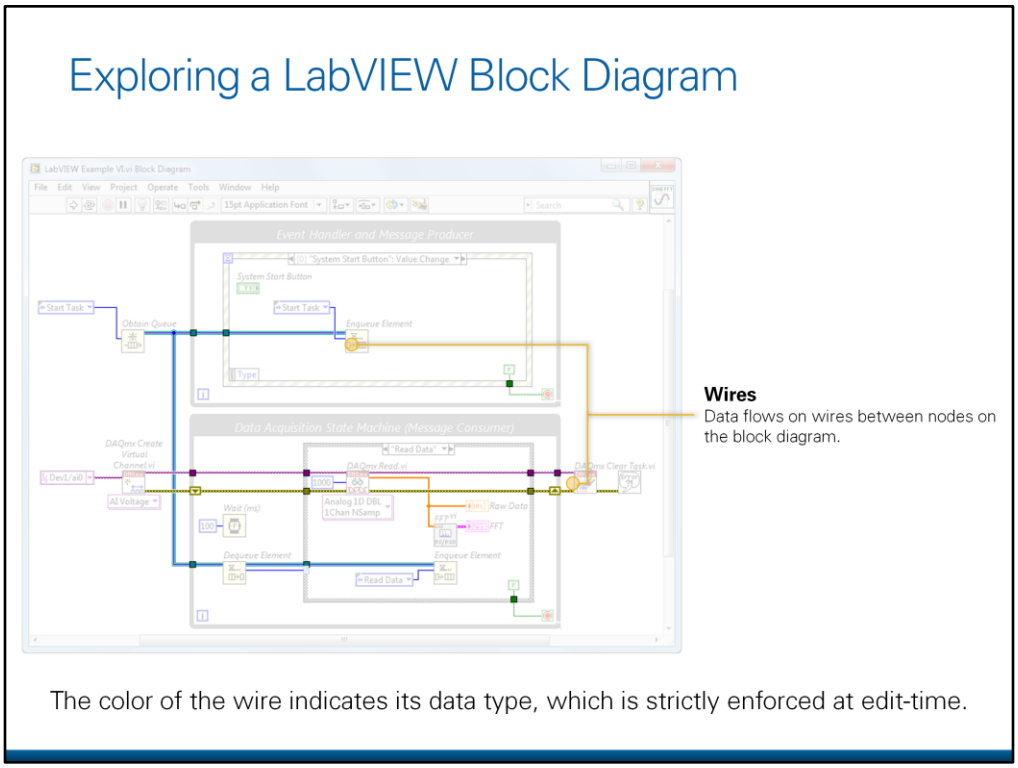
We already learned that front panel elements have corresponding block diagram terminals so that you can use the source code to programmatically access values from the user interface or update the values represented on the user interface.

If data flows from the wire to the terminal (on its left), you know your source code is outputting a value to the user interface via a front panel indicator.

Conversely, if data flows from the terminal to the wire (on its right), you know your source code is receiving input from control elements contained on the user interface.



Sometimes we need to represent data on the block diagram but we don't want the user to have control over the value of the data at run-time. In this case we can use block diagram *constants*, which can be defined statically at edit-time and cannot be changed by the user of the application. Constants can exist in any data type, including but not limited to numeric constants, Boolean constants, string constants, timestamp constants, array constants, and more.



The final components of our block diagram source code are the *wires*, which are paths that data use to travel through the code between nodes on the block diagram. Notice that there are several different wire colors, sizes, and patterns represented in this example code. Each aspect of the wire's appearance means something about the type of data that flows on that wire.

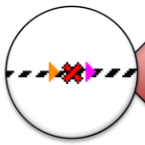
Wire color is a representation of a wire's data type, which is strictly enforced at edit-time. For example, the blue wires in the diagram above carry a reference to a queue.

Wire thickness is a representation of the dimension of the data that flows across the wire. For example, data could be scalar in nature or exist as a one- or multi-dimensional array.

Finally, *wire pattern* is a general indication of the complexity of a wire's data. Primitive data types such as numerics, strings, or Booleans are all represented using solid-color wires. Complex data types such as objects, references, clusters (structures), or errors are typically represented with a non-solid pattern to distinguish them on the block diagram.

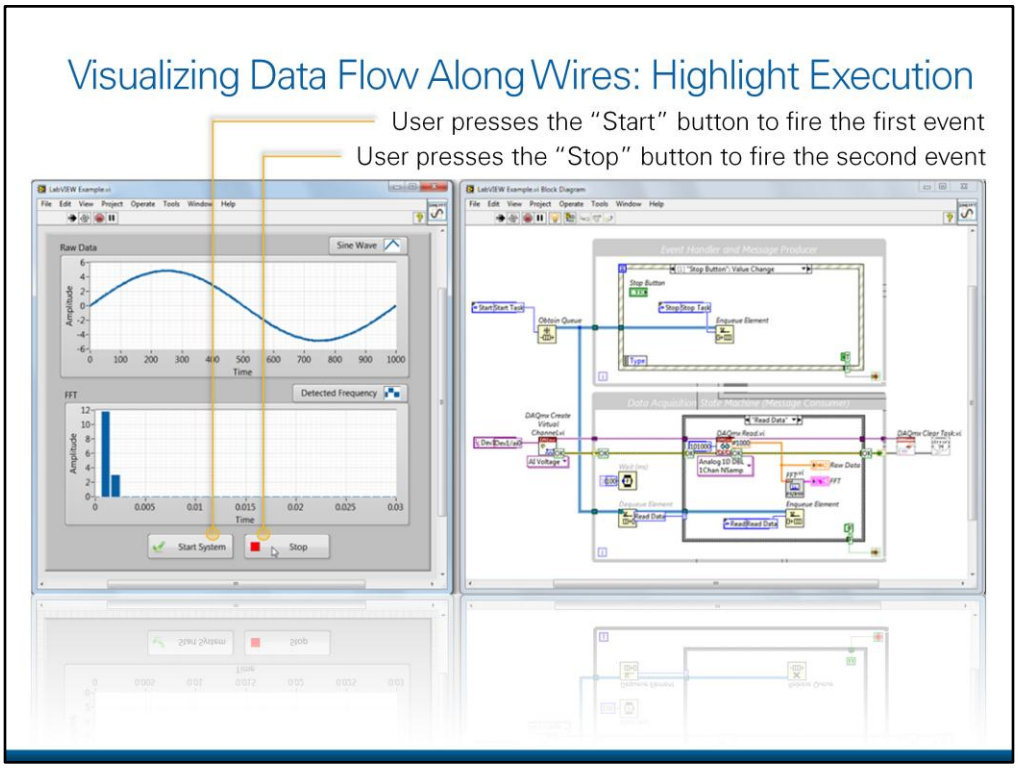
The Color, Style, and Thickness of Common Wires

Wire Type	Scalar	1D Array	2D Array	Color
Floating Point				Orange
Integer				Blue
Boolean				Green
String				Pink
Error				Yellow



A "broken wire" represents a data type conflict that LabVIEW cannot automatically resolve. Fix it, or your code won't run!

As you create and modify your LabVIEW source code, the LabVIEW compiler constantly checks the composition of your application to ensure that no syntactical programming errors have been made. The compiler will alert you of any errors by disallowing the execution of the code. A common programming mistake involves attempting to connect a wire between nodes that expect data of two different types or dimensions. In this case, unless LabVIEW can automatically resolve the difference in data types (for example, the conversion of an integer to a floating point value), the LabVIEW compiler will "break" the wire connecting the elements.

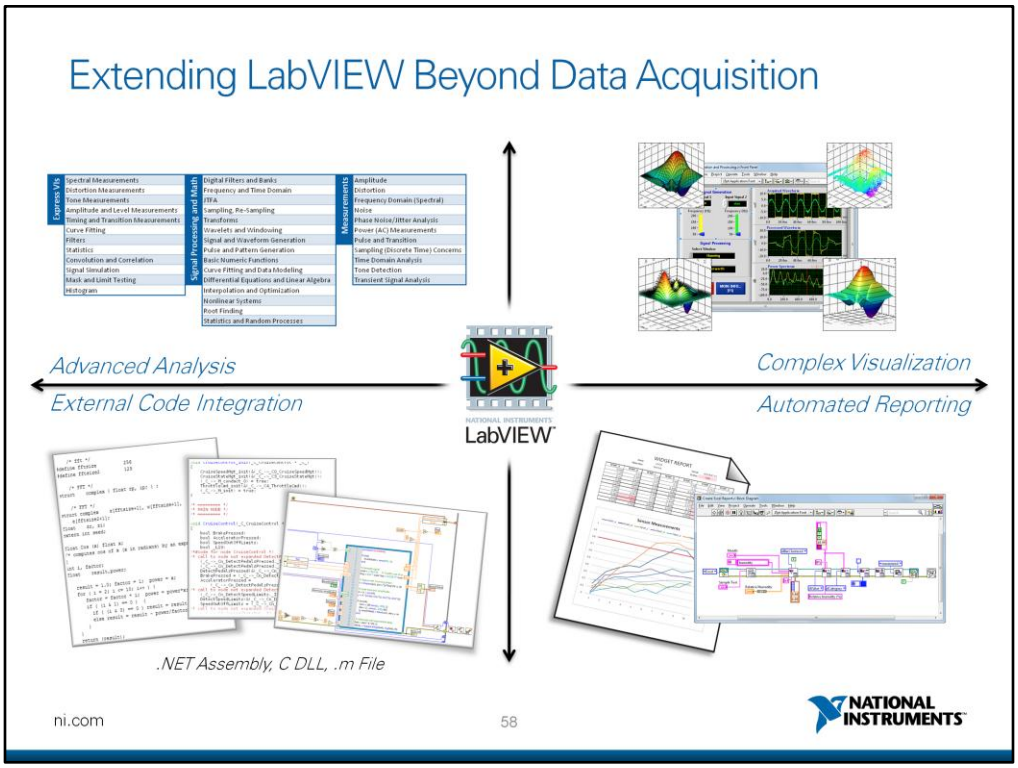


Now that we understand all of the fundamental building blocks of our example code, let's watch the way in which each of the pieces interact to contribute toward the measurement application solution.

When compiled, traditional software – including that written in both LabVIEW and text-based languages – executes as fast as it can, assuming no explicit timing or control elements have been used to slow or delay the execution of code. This makes it extremely difficult to visualize the execution of code to determine where errors or bugs may be occurring. For this reason, most modern development environments (including LabVIEW) contain constructs for adding breakpoints to code in order to pause execution at a particular point in the code. After execution has paused, developers can *step into*, *step out of*, and *step over* sections of code in order to isolate the occurrence and source of erroneous behavior.

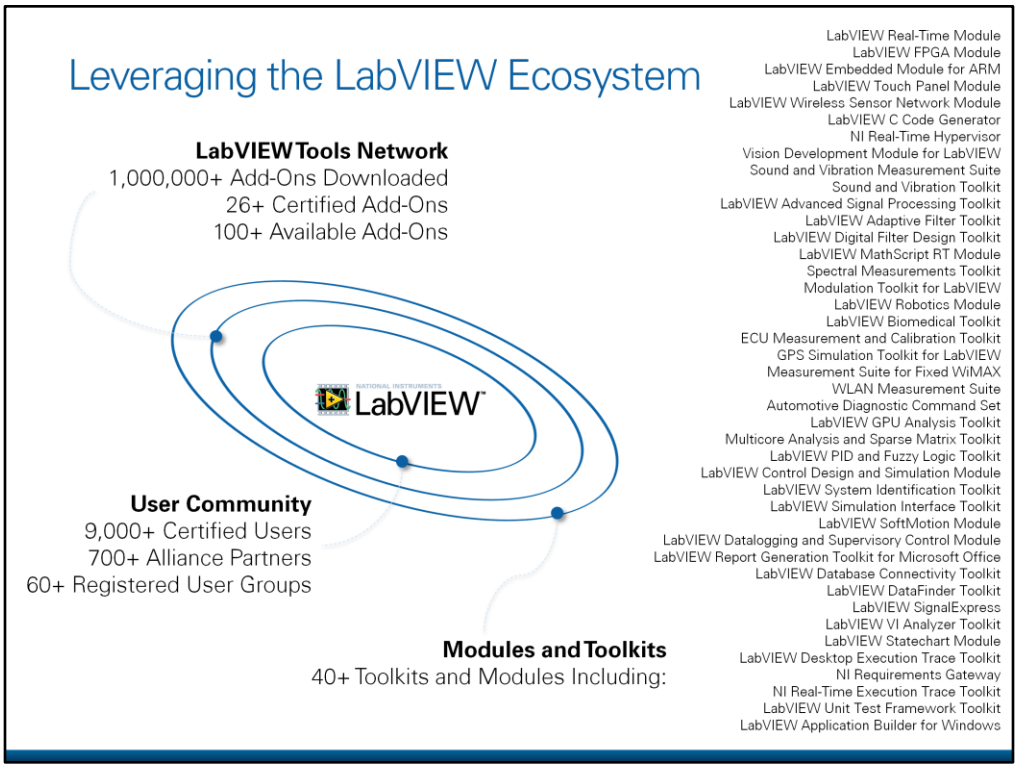
The graphical nature of G enables LabVIEW to include an additional and extremely powerful debugging and visualization technique called *Highlight Execution*, which slows the execution of code and displays the flow of data directly on the block diagram.

Insider Tip!
While Highlight Execution is a terrific debugging tactic, it is not a catch-all solution for hunting bugs. The most pesky of timing and synchronization errors, for example, only occur when code runs at full speed. Make sure you use a combination of techniques to identify bugs in the most efficient way.



Thought it isn't a focus of this particular seminar, after you acquire data using LabVIEW, you'll certainly need to do something with it. Rest assured that LabVIEW contains all the functionality you'll need for signal processing and analysis, visualization, automated reporting, and even the integration of external code. With LabVIEW, you'll be able to re-use any existing software investment you've made in external environments or languages or use the extensive capabilities built into the software to turn your measurement data into actionable information.

ni.com/labview
ni.com/daq

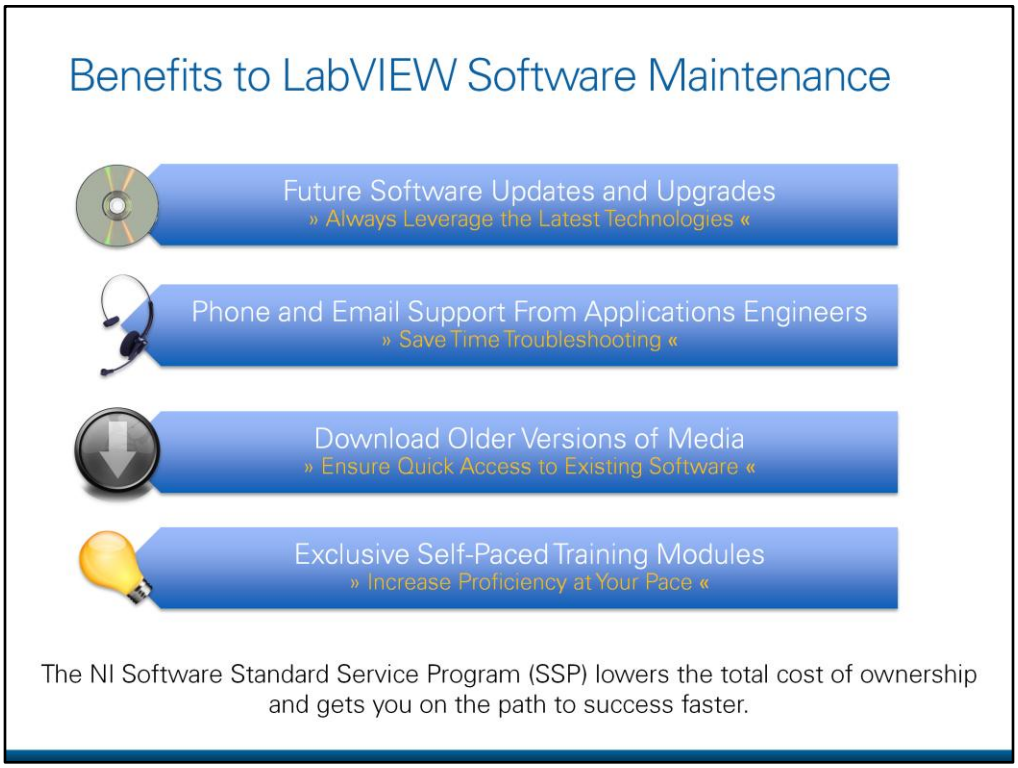


An investment in LabVIEW is really an investment in an entire software platform and ecosystem, which is comprised of the users and the various add-ons that are available.

A platform is more than features and technology; it's that plus the infrastructure and the community. Everything about the platform's ecosystem is important for the long-term viability and support of a product – the services, support structure, networks and more. Over the years, the LabVIEW community – the users – have built up an extensive ecosystem to fortify the platform and make it the vibrant global ecosystem that it is today.

To be a platform you also have to have a network of users and partners building complimentary products – such as the LabVIEW Tools Network containing hundreds of available LabVIEW add-ons. However, it's more than being connected to products and technology, it's also about connecting the people. LabVIEW users communicate daily via user groups, online forums and blogs.

To learn more about LabVIEW add-ons, modules and toolkits, visit **ni.com/labview/products**.



With the purchase of LabVIEW, you'll automatically receive a year of access to the NI Standard Service Program (SSP). Membership in National Instruments Standard Service Program can help you maximize your software investments by offering the latest software upgrades, technical support, self-paced training, and access to older versions of software.

For more information on the benefits of software maintenance, visit **ni.com/ssp**.


Coffee Break
Enjoy Coffee and Networking
With Industry Peers

The Fundamentals of Data Acquisition (DAQ)

The Basics of Making PC-Based Measurements

ni.com

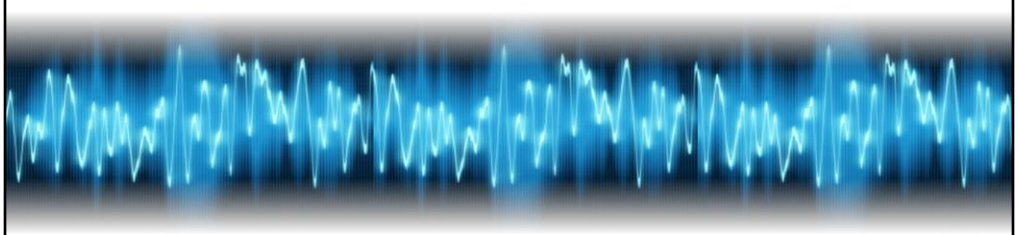
NATIONAL INSTRUMENTS

ni.com/labview
ni.com/daq

What Is Data Acquisition (DAQ)?

Data acquisition (DAQ) is the process of measuring an electrical or physical phenomenon such as voltage, current, temperature, pressure, or sound with a computer.

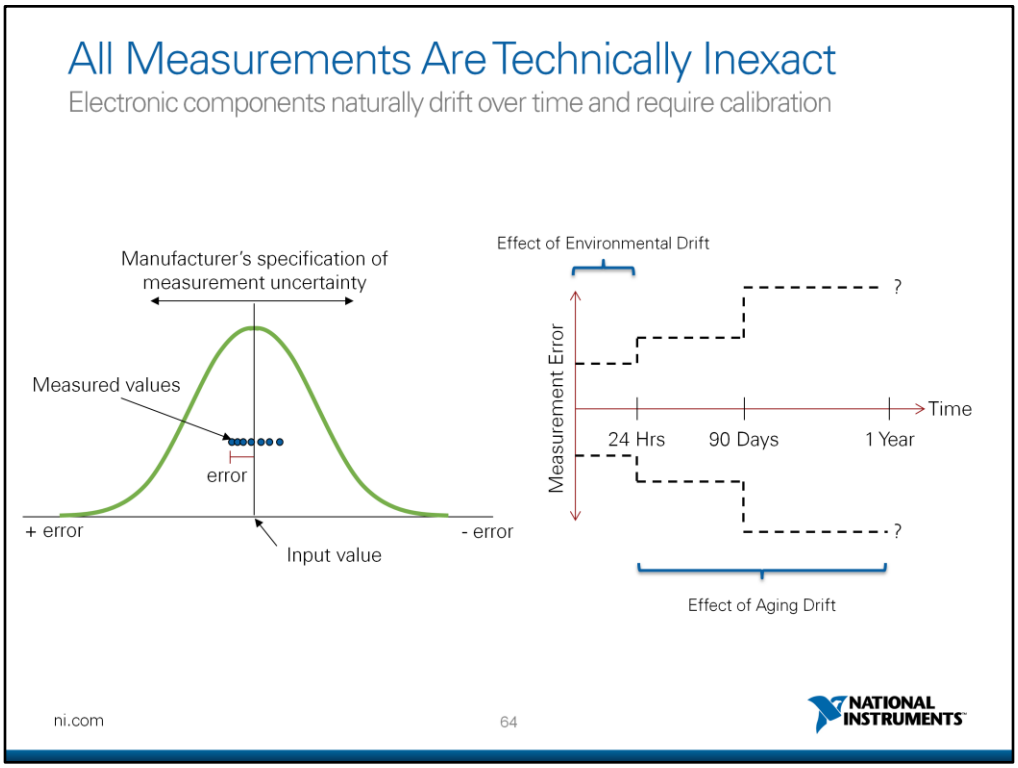
Compared to traditional measurement systems, PC-based DAQ systems exploit the processing power, productivity, display, and connectivity of industry-standard computers providing a more powerful, flexible, and cost-effective measurement solution.



Data acquisition hardware adds inputs and outputs (I/O) to standard laptop and desktop computers, turning a general-purpose PC into a flexible, high-performance measurement system. By taking advantage of the fast pace of standard PC technology improvements, you can achieve progressively faster processing, more memory, larger hard drives, and superior graphics displays for system visualization at increasingly lower costs. Standard PC buses like USB, PCI, and PCI Express provide a path for sensor and voltage measurements to be streamed directly into your computer for processing, analysis and visualization.

The purpose of data acquisition is to measure an electrical or physical phenomenon such as voltage, current, temperature, pressure, or sound.

PC-based data acquisition uses a combination of modular hardware, application software, and a computer to take measurements. While each data acquisition system is defined by its application requirements, every system shares a common goal of acquiring, analyzing, and presenting information. Data acquisition systems incorporate signals, sensors, actuators, signal conditioning, data acquisition devices, and application software.



The entire portfolio of National Instruments data acquisition hardware is calibrated to a known standard of absolute accuracy, but it is important to know that no matter what hardware you use and what vendor it came from, you can never measure an actual value; instead, you're measuring an estimate. All measurements should be within the manufacturers specification and in many cases may be bounded by a distribution (e.g. Gaussian, rectangular), but each measurement of the same actual value may be slightly different due to measurement error – most often caused by drift.

NI measurement hardware features self-calibration functionality that can compensate for small drift that may occur due to temperature and environment, but it is important to note that as your measurement hardware ages, the onboard reference will drift as well. Coupled with the drift of other analog components, this can result in significant error. Environmental changes and aging means the manufacturer can no longer predict the uncertainty of your measurements over time.

This can result in:

- Increased measurement uncertainty
- Reduced measurement consistency
- Lower production yields
- Failure to meet ISO-9000 requirements

These risks and effects can be mitigated with calibration.

NI Offers a Range of Hardware Calibration Services

	Traceable Calibration	*Compliant Calibration	*Accredited Calibration
Verify and adjust measurement performance using NI-approved calibration procedures	✓	✓	✓
Detailed measurement data for all channels	✓	✓	✓
Available at point of sale	✓	✓	✓
Uncertainty evaluation		✓	✓
Performed at laboratory accredited to ISO 17025		✓	✓
Calculated measurement uncertainties (includes ISO 17025 accreditation body logo)			✓

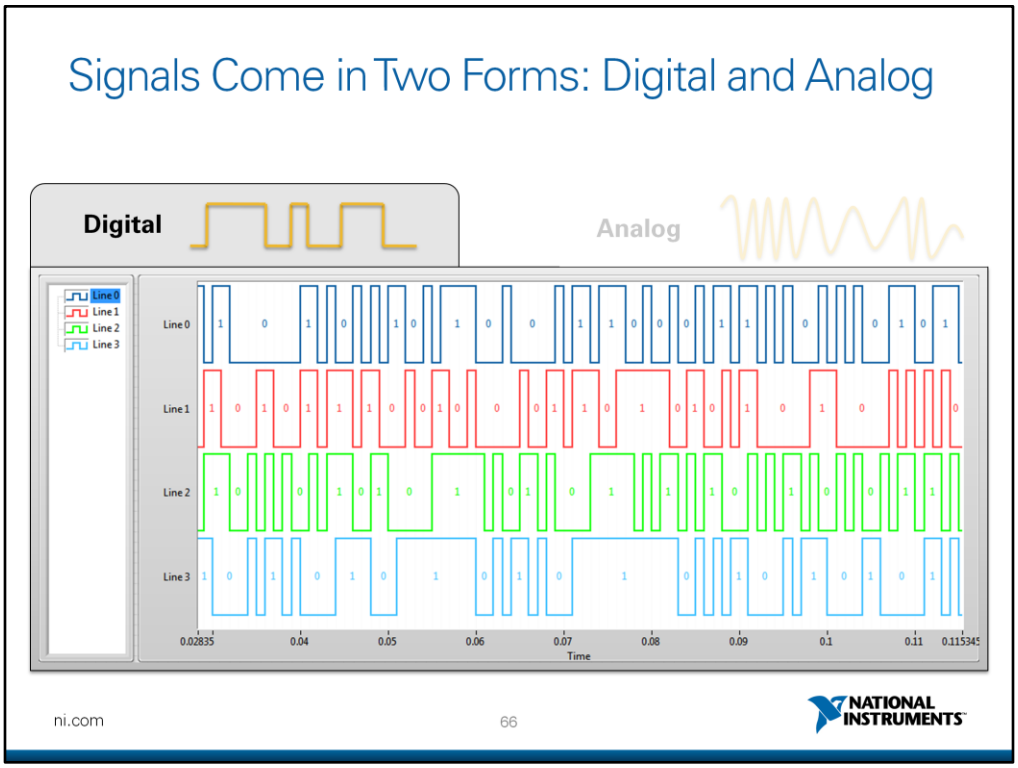
**May be performed at NI Certified Calibration Center operated by third-party provider.*

It is necessary to calibrate instruments at regular intervals as defined by the manufacturer. Calibration quantifies and improves the measurement performance of an instrument. Benefits of maintaining properly calibrated equipment include reduced measurement errors, consistency between measurements, increased production yields, and the assurance of accurate measurements. To help you meet your calibration needs, NI provides calibration support and services that include manual calibration procedures, services to calibrate your products, and automated calibration software specifically designed for use by metrology laboratories.

All calibration services from NI are traceable to national and international standards and help you comply with quality programs like ISO 9001. In addition to these basic requirements, NI offers a number of calibration service levels to meet your more advanced needs for compliance.

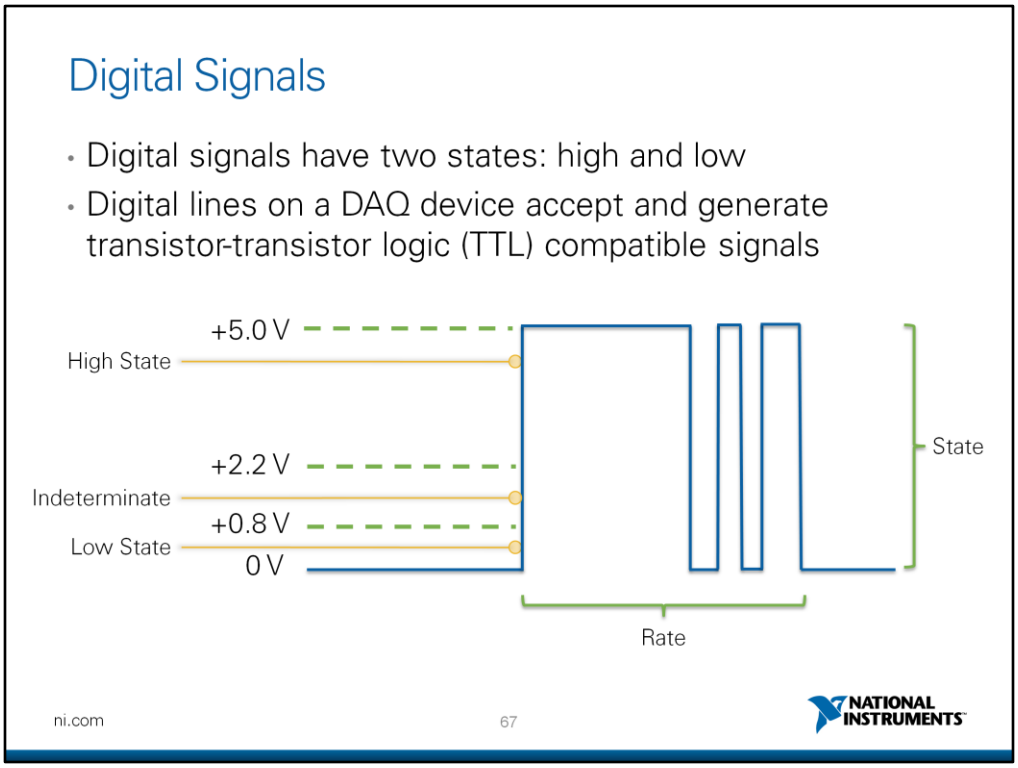
NI has an expansive network of service centers that work in conjunction with regional NI Certified Calibration Centers to meet your global needs for calibration service. These centers comply with global standards to ensure that you are getting the highest quality possible.

For more information on calibration and a variety of other global services that NI offers to ensure complete system success, visit **ni.com/services**.



A signal can fall into one of two categories: *digital* or *analog*. The type of signal we're trying to measure will affect the way in which we will measure the signal, so understanding the differentiation between them is important.

First, let's examine characteristics that classify a signal as digital.

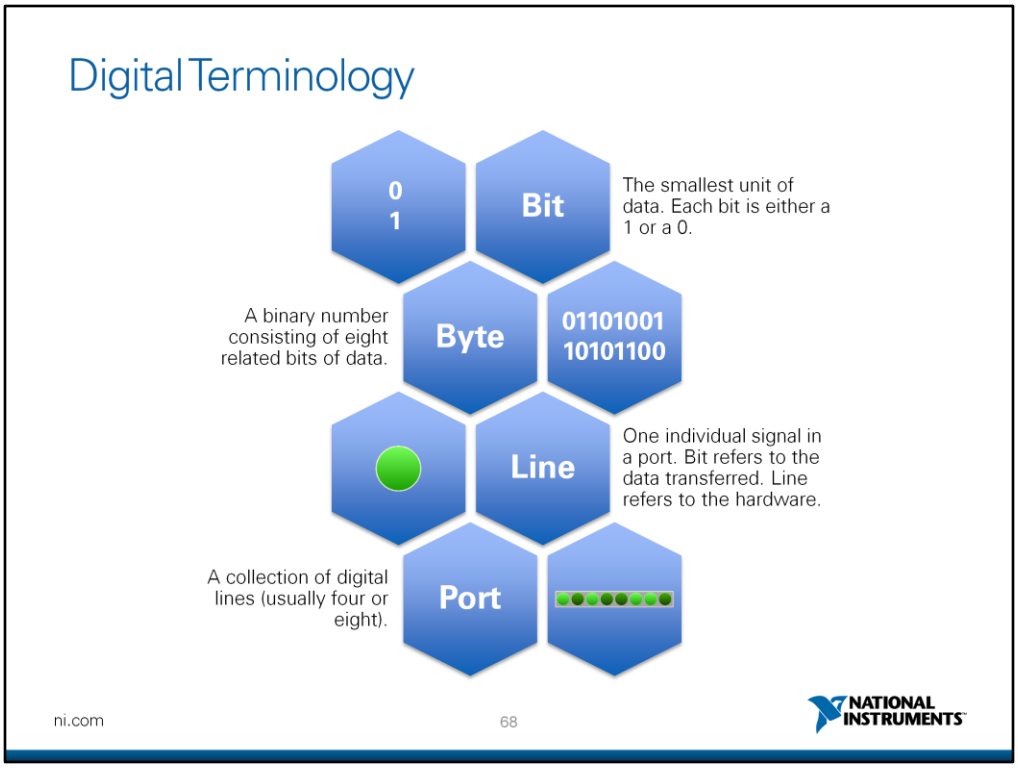


A digital signal is one in which the signal has only two possible states: *ON* or *OFF*. ON is also called logic high and OFF is also called logic low.

An example of a digital signal is a TTL (Transistor-to-Transistor Logic) signal. The specifications for a TTL signal state that a voltage level between 0 – 0.8 volts it is considered logic low, and a voltage level between 2.2 – 5.0 volts is considered logic high. The area between 0.8 V and 2.2 V is indeterminate and could be interpreted as either a logic high or a logic low. To ensure that the digital lines measure your signal correctly make sure that the voltage level of your signal is never between 0.8 V and 2.2 V.

Most digital devices in industry accept a TTL compatible signal; this includes the digital lines on most DAQ devices, which will accept and generate TTL compatible signals.

Since a digital signal only has two states, we can only measure two quantities of a digital signal: state (high or low) and rate (the number of state transitions in a given amount of time).



Digital input and output is quite different from analog input or output, therefore we must learn some of the common terminology used with digital I/O operations.

- **Bit**

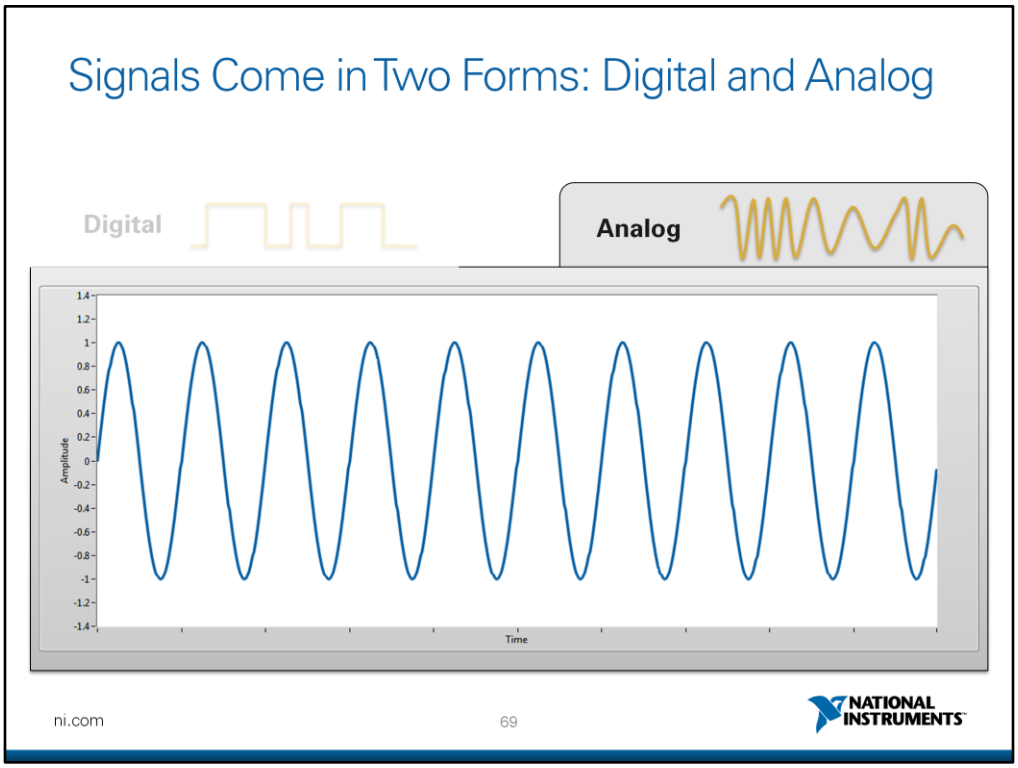
  A bit is the smallest unit of data used in a digital operation. Bits are binary so they can either be a 1 or a 0.
- **Byte**

  A binary number consisting of 8 related bits of data. Also used to denote the amount of memory required to store one byte of data.
- **Line**

  A line is an individual signal in a digital port. The difference between a bit and a line is that the bit refers to the actual data transferred, and the line refers to the hardware the bit is transferred on. However, the term line and bit are fairly interchangeable. For instance, an 8-bit port is the same as a port with 8 lines.
- **Port**

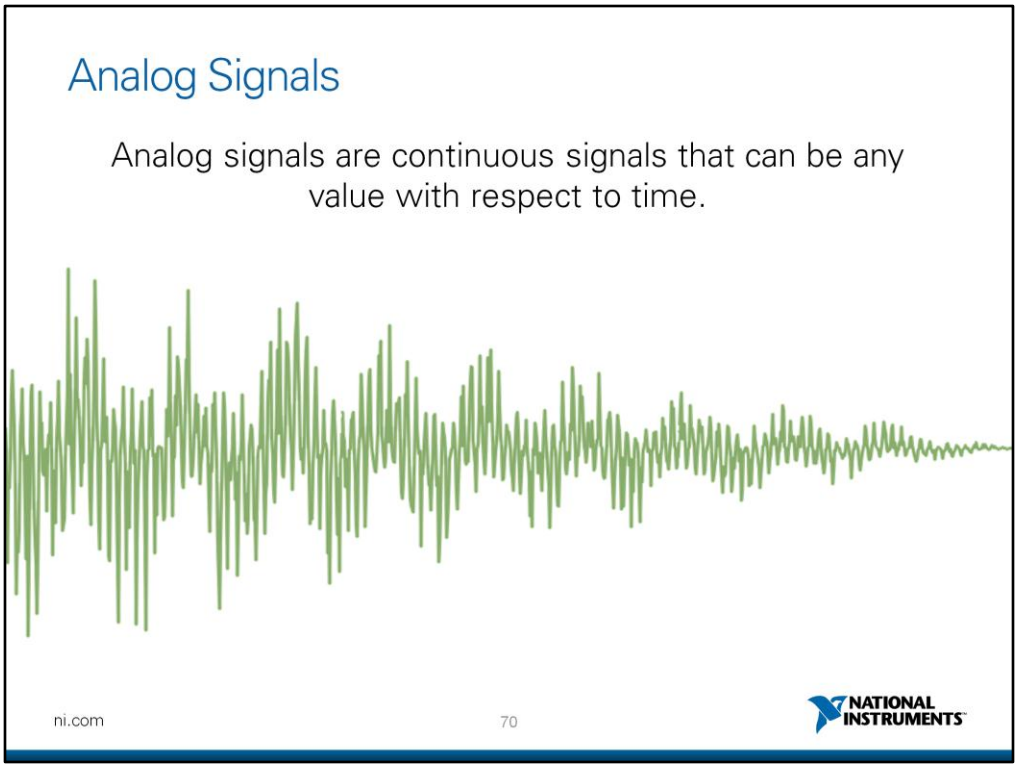
  A port is a collection of digital lines. Usually the lines are grouped into either a 8-bit, 16-bit, or 32-bit port.

National Instruments DAQ devices make use of this terminology to allow you to address a single line or an entire port with one identical function call.

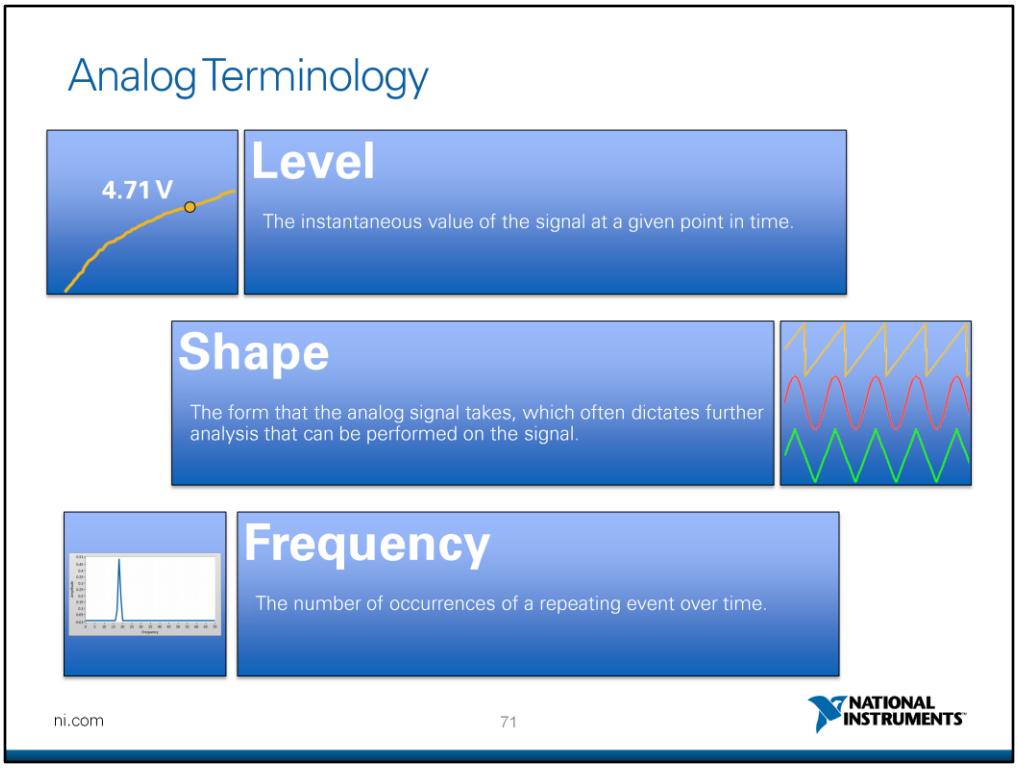


A signal can fall into one of two categories: *digital* or *analog*. The type of signal we're trying to measure will affect the way in which we will measure the signal, so understanding the differentiation between them is important.

Next, let's examine characteristics that classify a signal as analog.

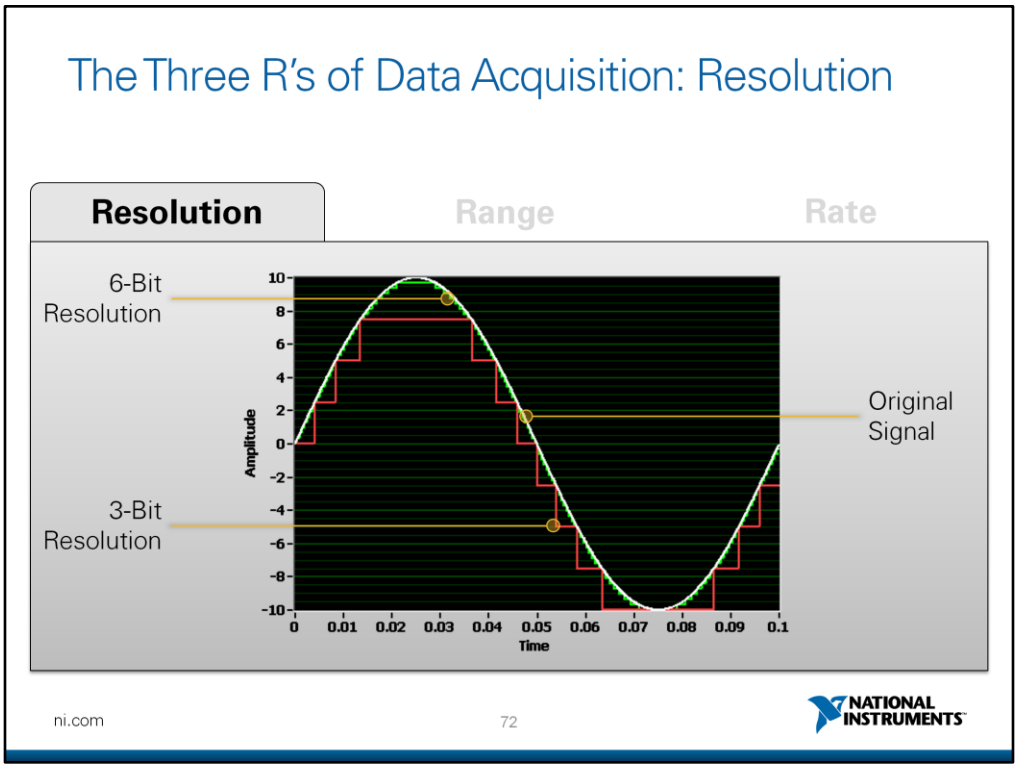


Unlike a digital signal, which contains only two possible states, an analog signal can be at any voltage level with respect to time. Since an analog signal can be at any state at any time, the physical quantities we want to measure differ from those of a digital signal.



When measuring and examining analog signals, there are three quantities that may be of value: we can measure the level, shape, or frequency of an analog signal.

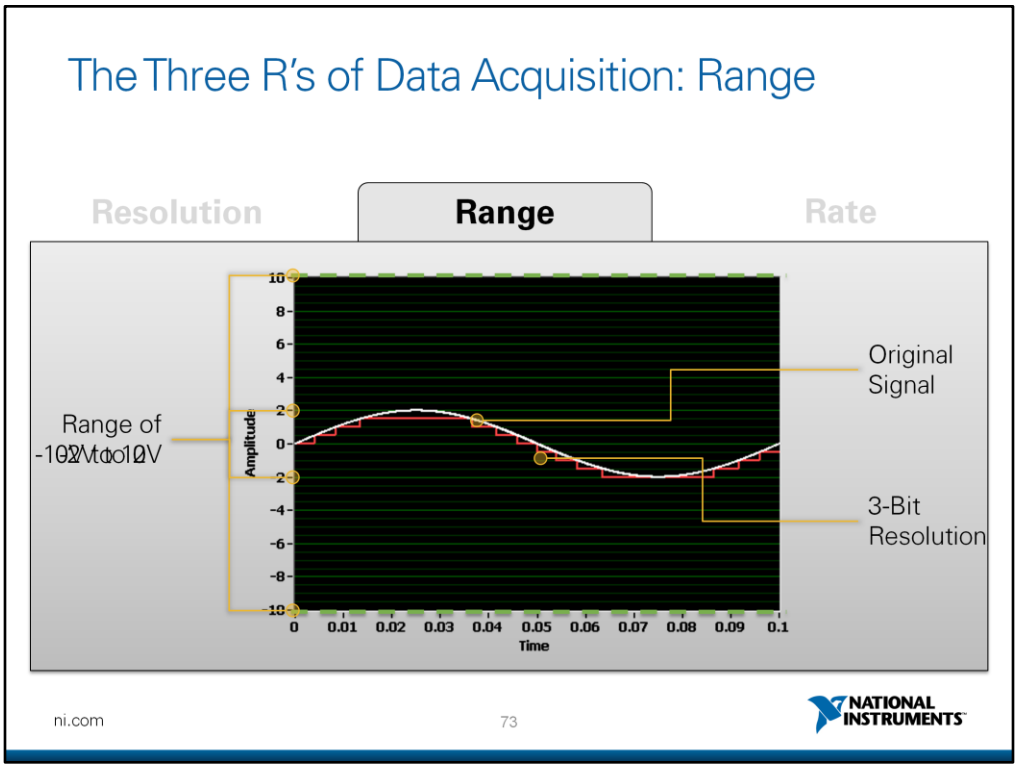
- **Level**
  Measuring the level of an analog signal is similar to measuring the state of a digital signal. The only difference is that an analog signal can be at any voltage state, whereas a digital signal can only be at one of two states.
- **Shape**
  Because analog signals can be at any state with respect to time, the shape of the signal is often important. For instance, a sine wave has a different shape than a sawtooth wave. Measuring the shape of a signal opens the door to further analysis on the signal itself such as peak values, slope, integration, and more.
- **Frequency**
  Measuring the frequency of an analog signal is similar to measuring the rate of a digital signal. However, you cannot directly measure the frequency of an analog signal. Software analysis of the signal is required to extract the frequency information. The analysis is usually done by an algorithm called a Fourier Transform.



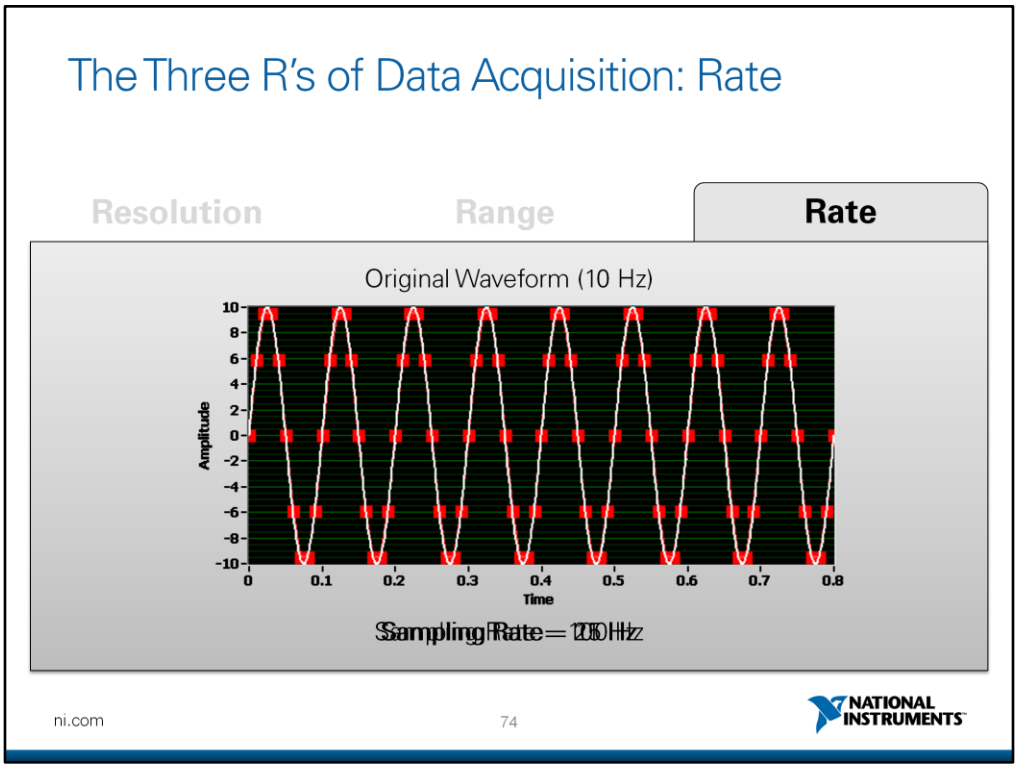
Analog signals can be at any state at any time, however, the process of capturing a continuously variable analog signal – which theoretically has an infinite number of possible states depending on the level of granularity at which you view the transitions of the signal – involves translating the signal into terms your computer can understand. Since computers are digital in nature, a fundamental component of the data acquisition process is analog-to-digital conversion, which is performed by a hardware component called an analog-to-digital converter, or ADC.

An ADC takes an analog signal and turns it into a binary number. Therefore, each binary number from the ADC represents a certain voltage level. The ADC returns the highest possible level without going over the actual voltage level of the analog signal. An ADC's *resolution* describes the number of discrete binary levels the ADC can use to represent an analog signal, or the granularity with which changes in the signal are measured. To figure out the number of binary levels available based on the resolution, simply calculate $2^{\text{Resolution}}$. Therefore, the higher the resolution, the more levels you will have to represent your signal. For instance, an ADC with 3-bit resolution can measure 2^3 (or 8) voltage levels, while an ADC with 16-bit resolution can measure 2^{16} (or 65,536) voltage levels.

Even though ADCs are not made with only 3-bit resolution, if they were, the lowest voltage level would correspond to 000, the next highest to 001, and so on all the way up to 111.



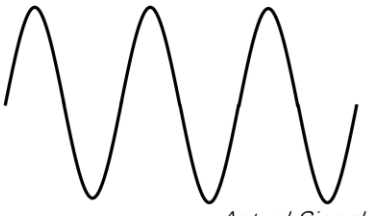
Imagine we have an ADC with an input *range* of -10.0 to 10.0 V. With an example 3-bit resolution, this would allow us to represent 8 possible voltage values, which means we could detect changes in our analog signal of a minimum of 2.5 V. If our original signal varies between -2.0 and 2.0 V, it would only take up part of the range of the ADC and we would only be able to detect and represent two possible values because our signal only utilizes 20% of the range of the ADC. Therefore, we either have to choose a device with a range appropriate for the signal we intend to measure, or manipulate our input signal (via amplification) to better utilize the range of the ADC. We'll learn more about amplification in a moment.



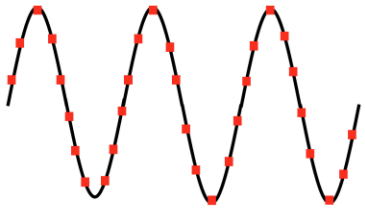
A sampling *rate* is the frequency at which we measure the original signal. Notice how the rate at which we sample drastically affects the accuracy of the depiction of the original signal we end up with. Let's explore this important concept in further detail.

Sampling Rate Considerations

- An analog input signal is continuous with respect to time.
- Sampled signal is series of discrete samples acquired at a specified sampling rate.
- The faster we sample, the more our sampled signal will look like our actual signal.
- If not sampled fast enough, a problem known as aliasing will occur.

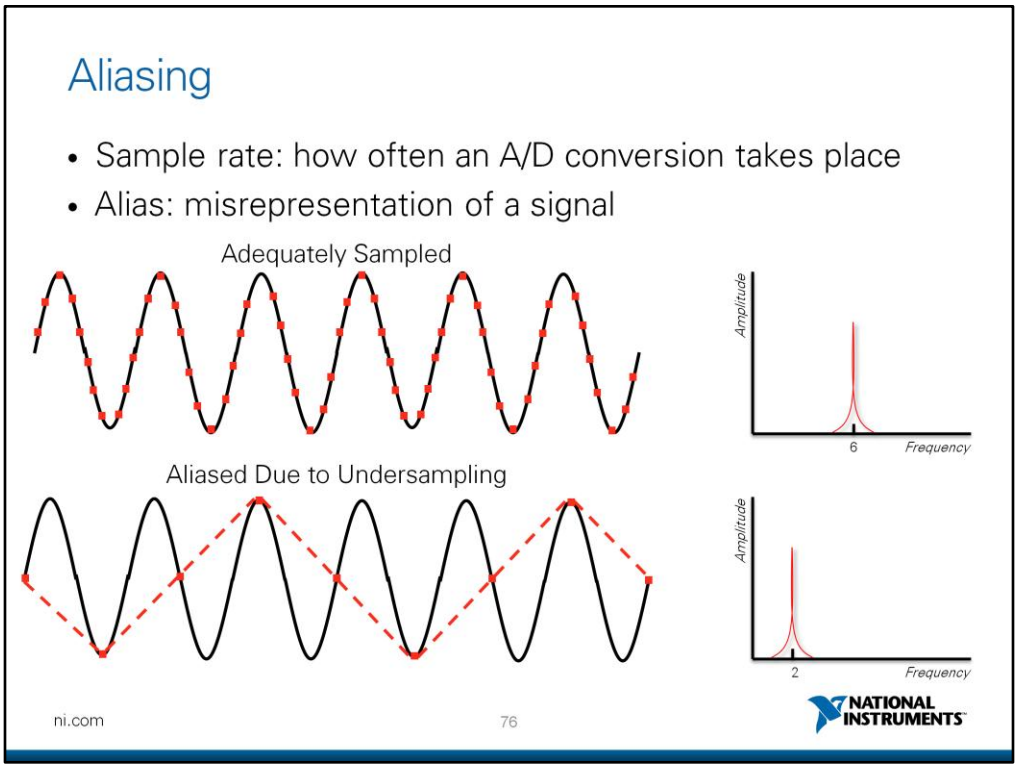


Actual Signal

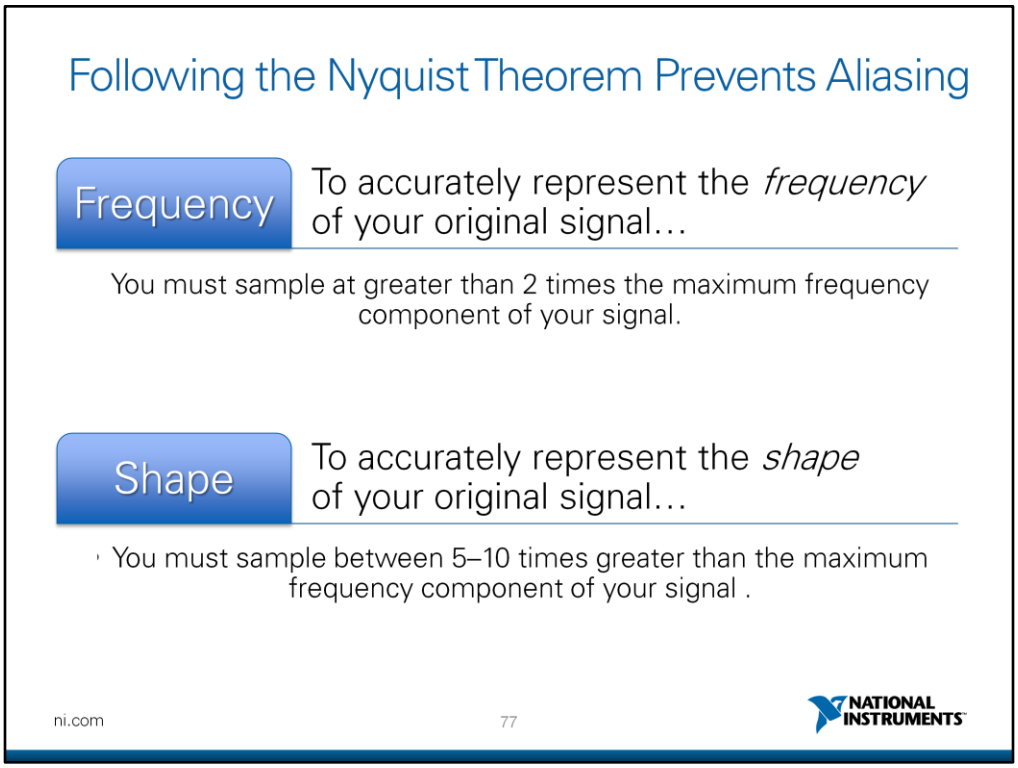


Sampled Signal

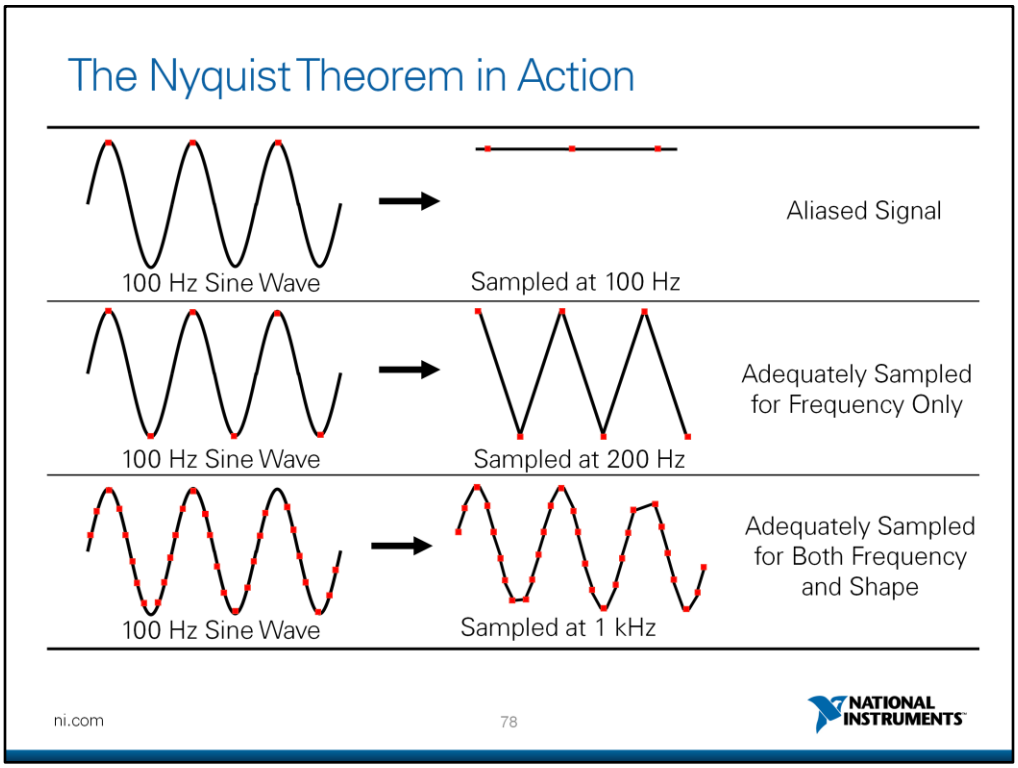
This phenomenon happens because when we are acquiring an analog signal, we are taking a signal that is continuous with respect to time (an infinite amount of points) and converting it into a series of discrete samples (a finite amount of points). The faster we sample, the more points we will acquire, and therefore the better our representation of the signal will be. If we don't sample fast enough we will experience a problem known as *aliasing*.



The sampling rate is the rate at which we acquire samples, but it is also the rate at which the analog-to-digital conversion takes place. As you can see in the top graph, our signal is adequately sampled—we see the shape **and** frequency of the signal. When a signal is not adequately sampled, aliasing can occur. That is, the signal can be misrepresented, as seen in the bottom graph. Simply put, with a sampling rate that is too low, we will not be able to adequately represent our signal.



The Nyquist Theorem is a rule we can follow to prevent aliasing our signal. The Nyquist Theorem states that you must sample at greater than 2 times the maximum frequency component of your signal to accurately represent the frequency of the signal. Notice that the Nyquist Theorem only deals with accurately representing the *frequency* of the signal; it doesn't mention anything about properly representing the shape of our signal. In order to properly represent the shape of your signal you must sample between 5–10 times greater than the maximum frequency component of your signal. Next, we will illustrate the Nyquist Theorem with various examples.



Assume we are trying to measure a 100 Hz sine wave.

First we will try sampling our signal at exactly 100 Hz. According to the Nyquist Theorem 100 Hz is not fast enough to correctly represent the frequency of our signal. If our signal frequency is exactly 100 Hz and we are measuring at exactly 100 Hz we will get a straight line. We are obviously incorrectly representing both the shape and the frequency of our signal; therefore the Nyquist Theorem and our guideline for shape both hold true.

Next, let's sample our signal at 200 Hz. Note that this is exactly twice the frequency of our signal, so according to the Nyquist Theorem this is just fast enough to correctly represent the frequency of our signal. However, it is not fast enough to correctly represent the shape of our signal. If our signal is exactly 100 Hz and if we sample at exactly 200 Hz we will get the triangle wave shown above. Notice that the 100 Hz sine wave and the triangle wave have different shapes, but the same frequency; therefore the Nyquist Theorem and our guideline for shape still hold true.

Finally, let's sample our signal at 1 kHz. Since we are sampling at 10 times our frequency, we should be able to accurately represent both the frequency and shape of our signal. As you would expect, our sampled signal does look like a sine wave, and it has the same frequency as our measured signal; therefore the Nyquist Theorem and our guideline for representing the shape of our signal still hold true.

Conditioning Signals for Quality Measurements

- Signal conditioning improves a signal that is difficult for your DAQ device to measure
- Signal conditioning is not always required

Noisy, Low-Level Signal

Filtered, Amplified Signal

Most sensors need some sort of external hardware in order to perform their job. For instance, RTDs need an excitation current, and strain gages need a configuration of resistors called a Wheatstone bridge.

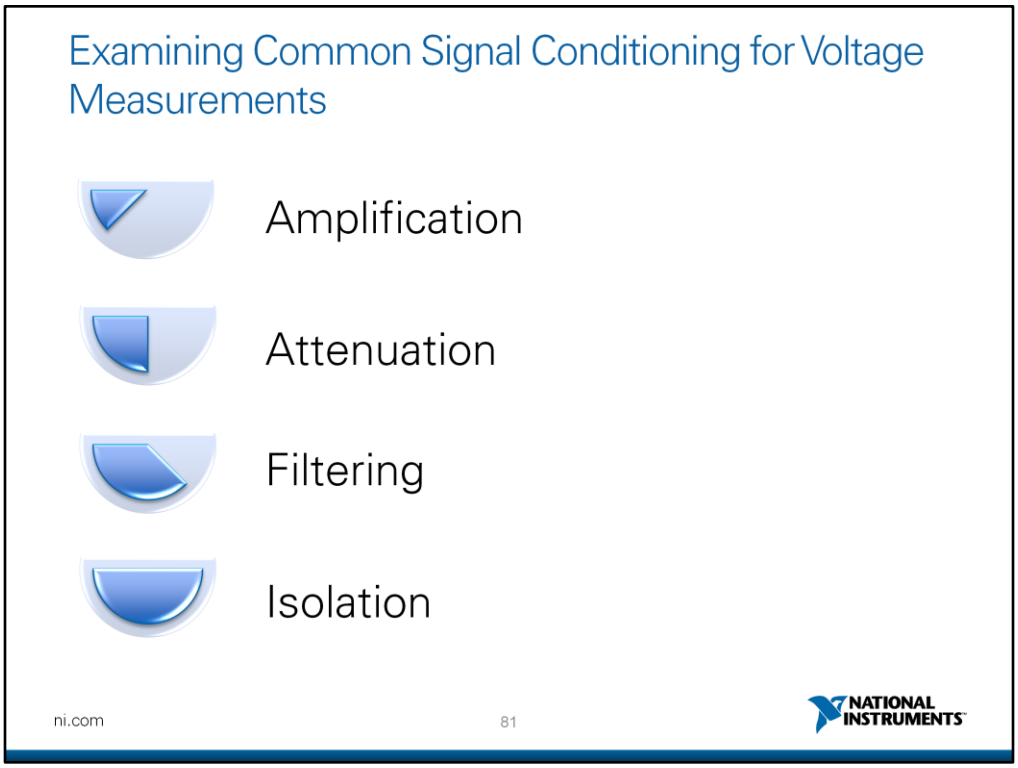
In addition to needing external hardware, not all transducers produce a perfect voltage for our data acquisition hardware to measure. The signal from the transducer could be noisy, or if could be too small or too large for the range of our DAQ device. For instance, thermocouples, strain gages, and microphones all produce a voltage in the millivolt range making it hard to detect changes in the signal due to range and resolution of the DAQ device's ADC.

The external hardware required by most sensors in order for a DAQ device to appropriately read the signal is called signal conditioning.

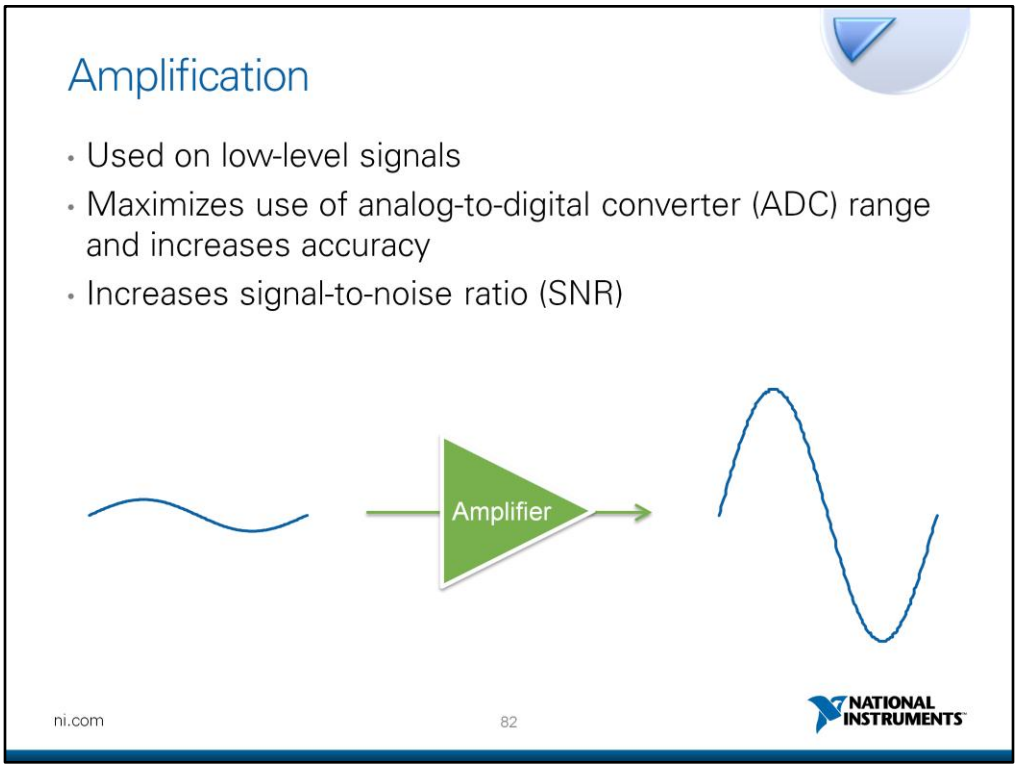
Common Signal Conditioning Examples

Transducer/Signals	Signal Conditioning
Thermocouples	Amplification, Linearization, Cold-Junction Compensation
RTD (Resistance Temperature Detector)	Current Excitation, Linearization
Strain Gage	Voltage Excitation, Bridge Configuration, Linearization
Common Mode or High Voltage	Isolation Amplifier
Loads Requiring AC Switching or Large Current Flow	Electromechanical Relays or Solid-State Relays
High-Frequency Noise	Low-Pass Filters

There are a number of types of signal conditioning techniques, depending on the sensor you're using and the type of signal conditioning (if any) it requires. Without a high-quality sensor and DAQ device, you'll be responsible for building and integrating the external signal conditioning circuitry yourself.

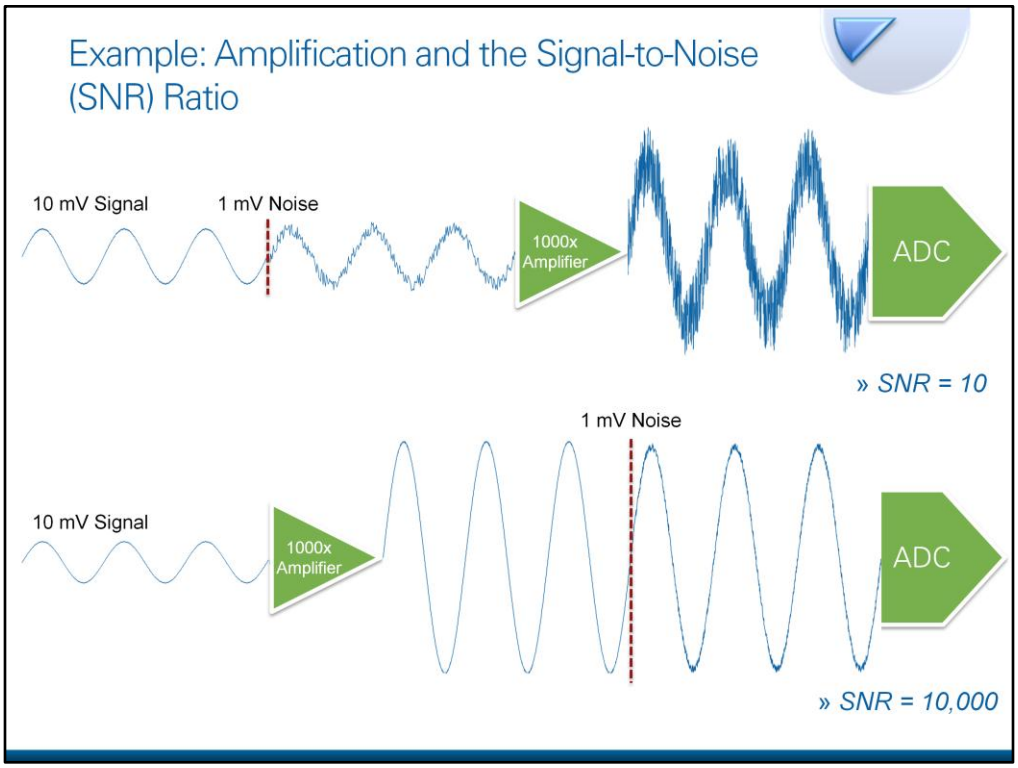


Four of the most common types of signal conditioning performed are amplification, attenuation, filtering, and isolation. Let's explore each in a bit more detail.



Amplification is a way of increasing a signal from a sensor that is too small for your DAQ device to accurately measure. A common example of a sensor that requires amplification is a thermocouple. Thermocouples output a voltage in the millivolt range; if you were to send the signal from your thermocouple straight to your DAQ device, it is feasible that a change of a degree or two in temperature would not be detected by your system. However, if we amplify the signal, we will be measuring a signal that is better suited to the range of our DAQ device.

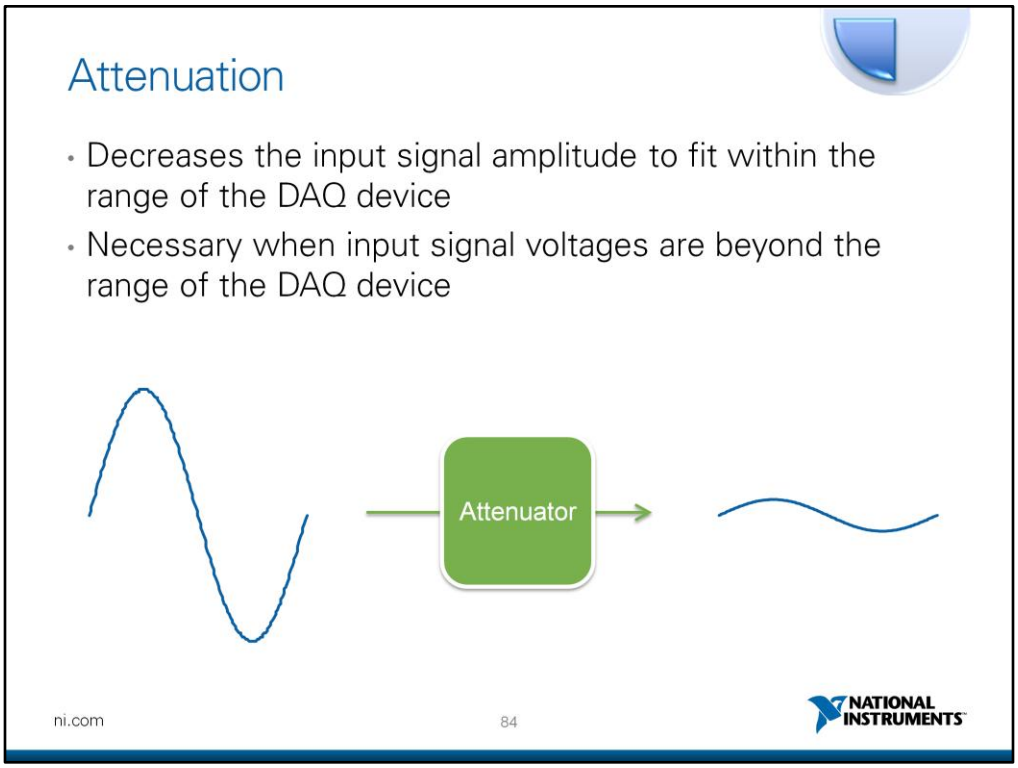
Your signal can either be amplified on the DAQ device or externally. The problem with amplifying the signal on the DAQ device is that we also amplify the noise the signal has picked up on its way to the DAQ device. In order to minimize the amount of noise that is amplified it is best to place the amplifier as close to the signal source as possible. Thus it is usually best to use some form of external amplification. As we will see next, we can show the benefit of external amplification with an index called the signal-to-noise ratio.



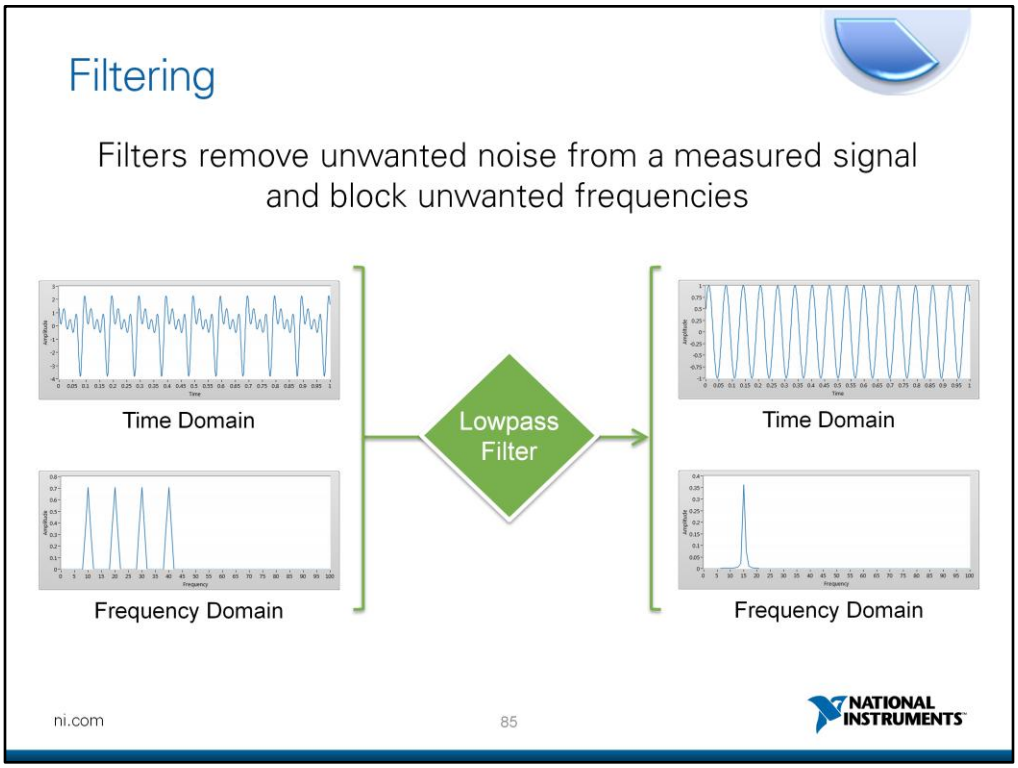
In this example, our 10 mV signal is faced with 1 mV of noise before being measured. As soon as the 1 mV of noise affects the 10 mV signal, 10% of our signal is erroneous. If we amplify the signal *after* the noise has already affected it, we'll be amplifying the noise as well; by adding an amplifier into the system that introduces a gain of 1000, the signal becomes a 10 V signal, but our signal-to-noise ratio remains 10.

If we instead apply the same amplification to the signal *before* the same 1 mV of noise has had a chance to affect the signal, the original signal (now 10 V) is still affected by the 1 mV of noise, but that noise has much less effect on the quality of the measured signal because the signal-to-noise ratio, in that case, is 10,000.

Because of this, if at all possible it is best to amplify your signal close to your signal source before any potential noise can be induced.



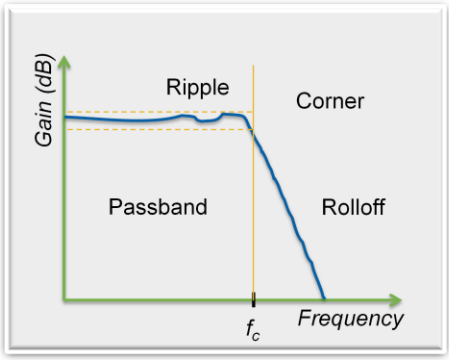
Attenuation – the opposite of amplification – is necessary when voltages to be digitized are beyond the digitizer input range. This form of signal conditioning decreases the input signal amplitude so that the conditioned signal is within ADC range. Attenuation is necessary for measuring high voltages.



Since noise is inevitable in the real world, it is often necessary to filter a signal in order to remove unwanted noise from the measurement. There are many different types of filters; one of the most common is the lowpass filter, which allows all frequency components lower than a specified cutoff frequency to pass while filtering all components higher than the cutoff frequency.

Filtering

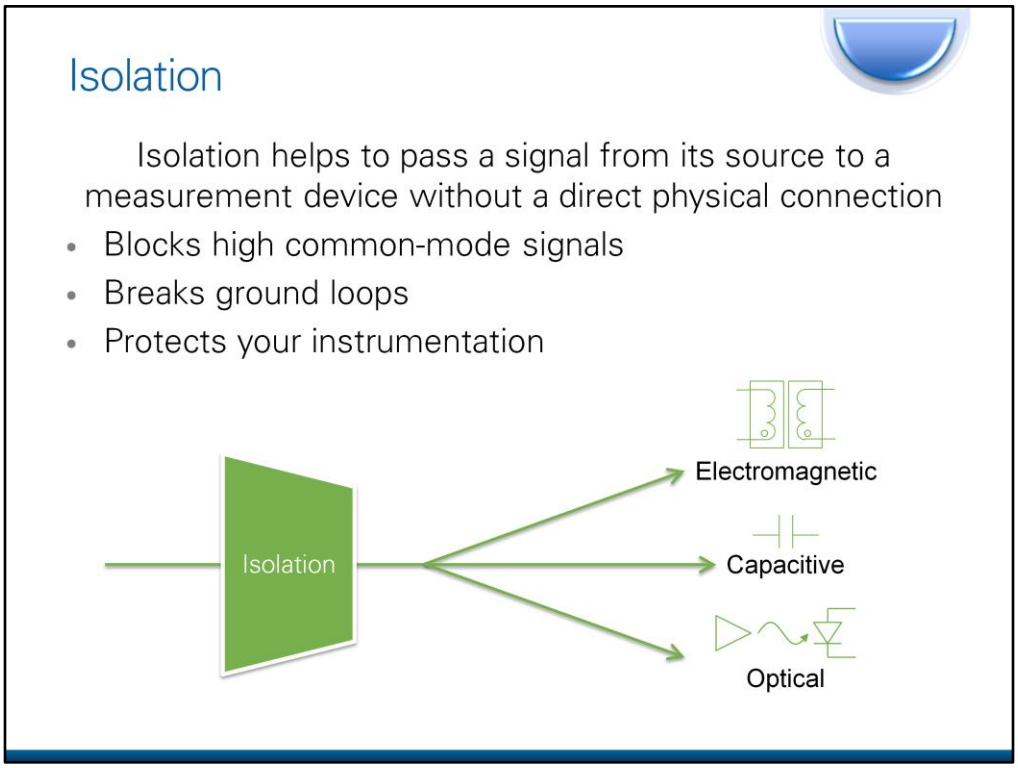
- Passband
  - Frequencies the filter lets pass
- Ripple
  - Filter's effect on the signal's amplitude
- Corner
  - Frequency where the filter begins blocking the signal
- Rolloff
  - How sharply the filter cuts off unwanted frequencies



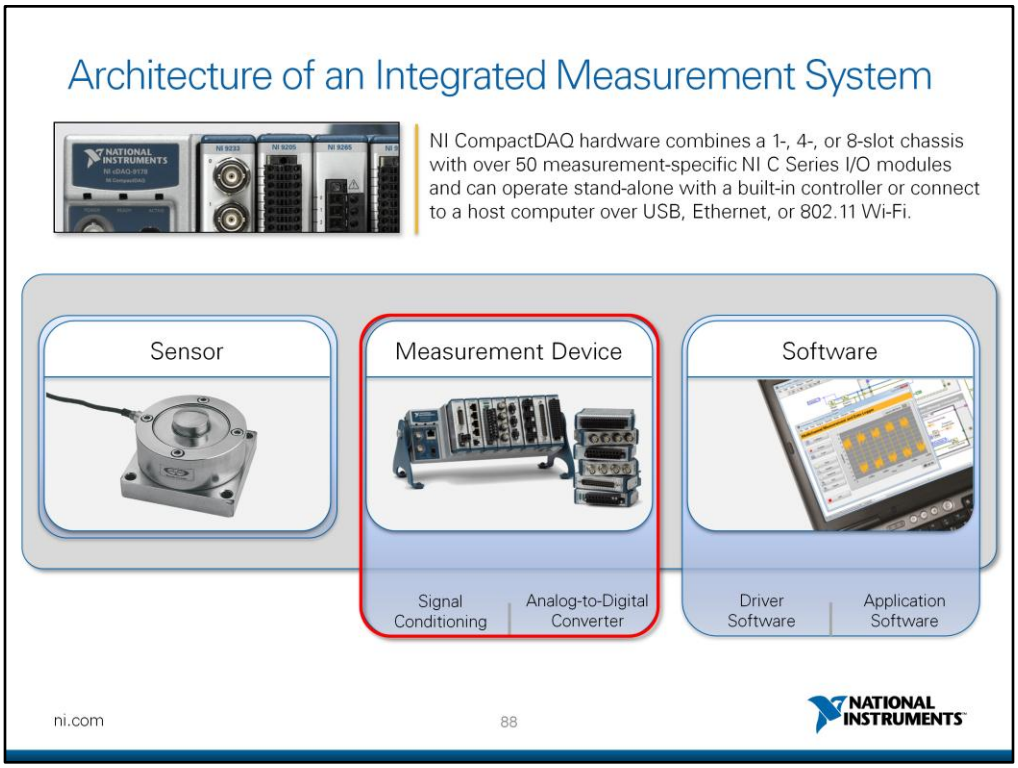
Example Bode Plot

A filter's attributes are typically described using Bode Plots

Different filters have different attributes to describe their performance. A filter's attributes are typically described in Bode Plots, like the example seen on the slide, which allow you to graphically visualize a filter's behavior.

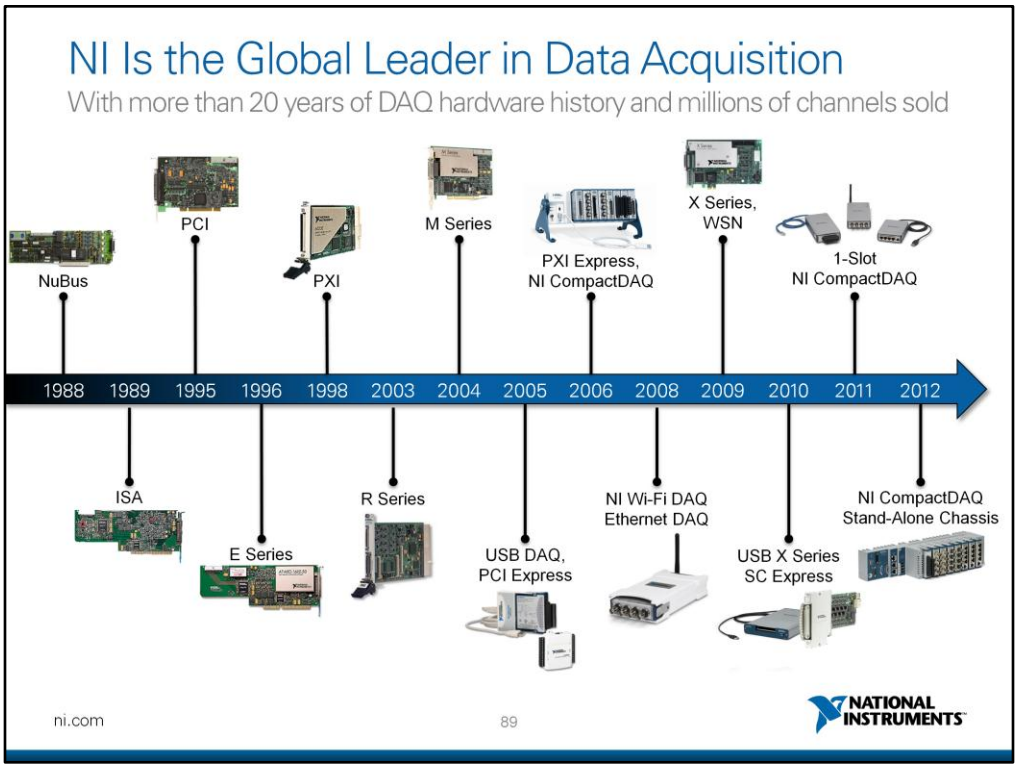


Isolated signal conditioning devices pass the signal from its source to the measurement device without a physical connection by using transformer, optical, or capacitive coupling techniques. In addition to breaking ground loops, isolation blocks high-voltage surges and rejects high common-mode voltage and thus protects both the operators and expensive measurement equipment.



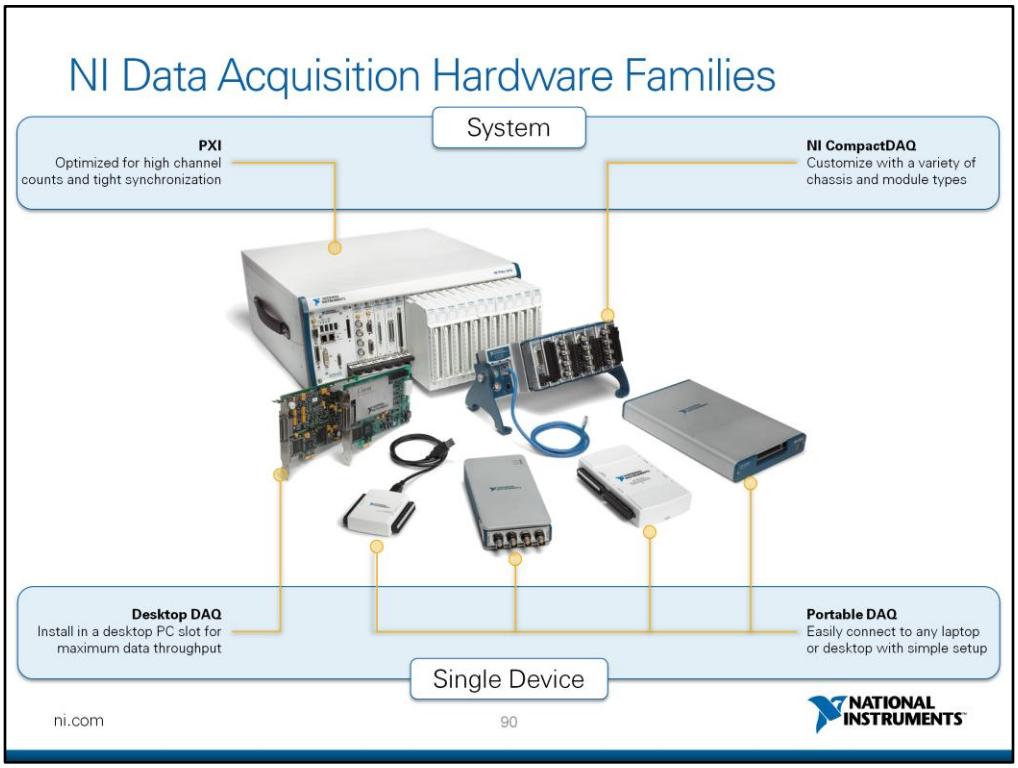
Now that we understand the complexity and fundamentals of data acquisition, let's explore an example NI hardware platform that drastically simplifies the automated measurement process – NI CompactDAQ.

ni.com/labview
ni.com/daq



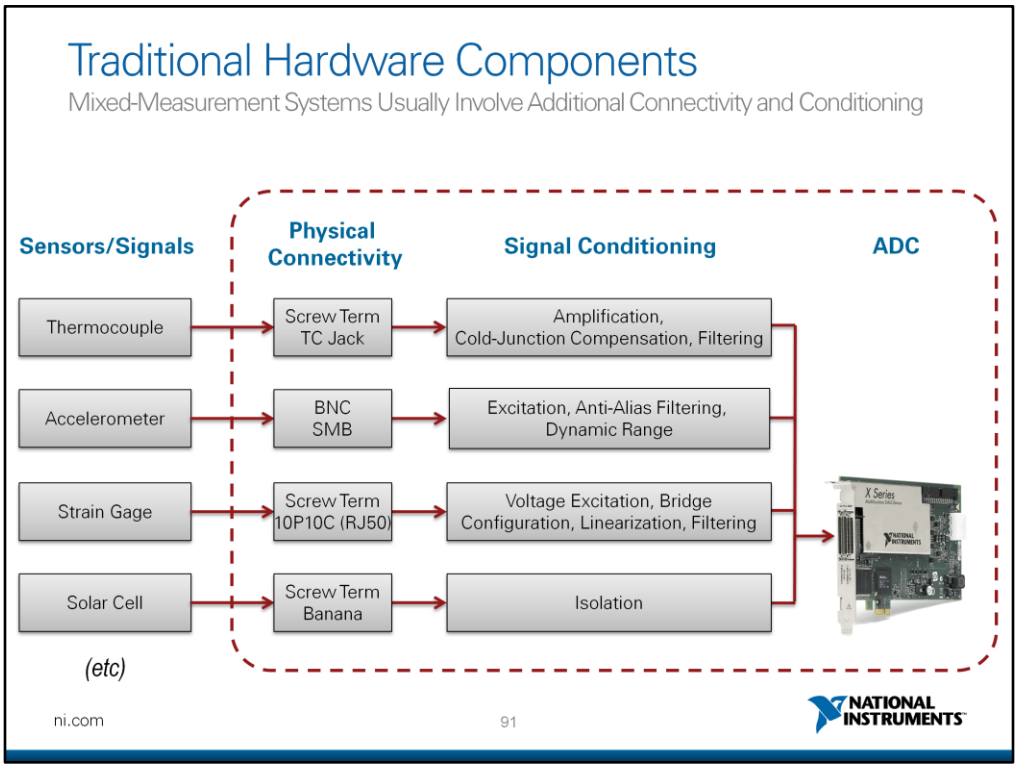
Following initial hardware releases focused on controlling traditional boxed-instruments with LabVIEW using the General Purpose Interface Bus (GPIB), National Instruments began to develop multifunction data acquisition hardware. Even as bus technologies have evolved over the last twenty years, NI hardware has remained the industry standard hardware for data acquisition.

ni.com/labview
ni.com/daq

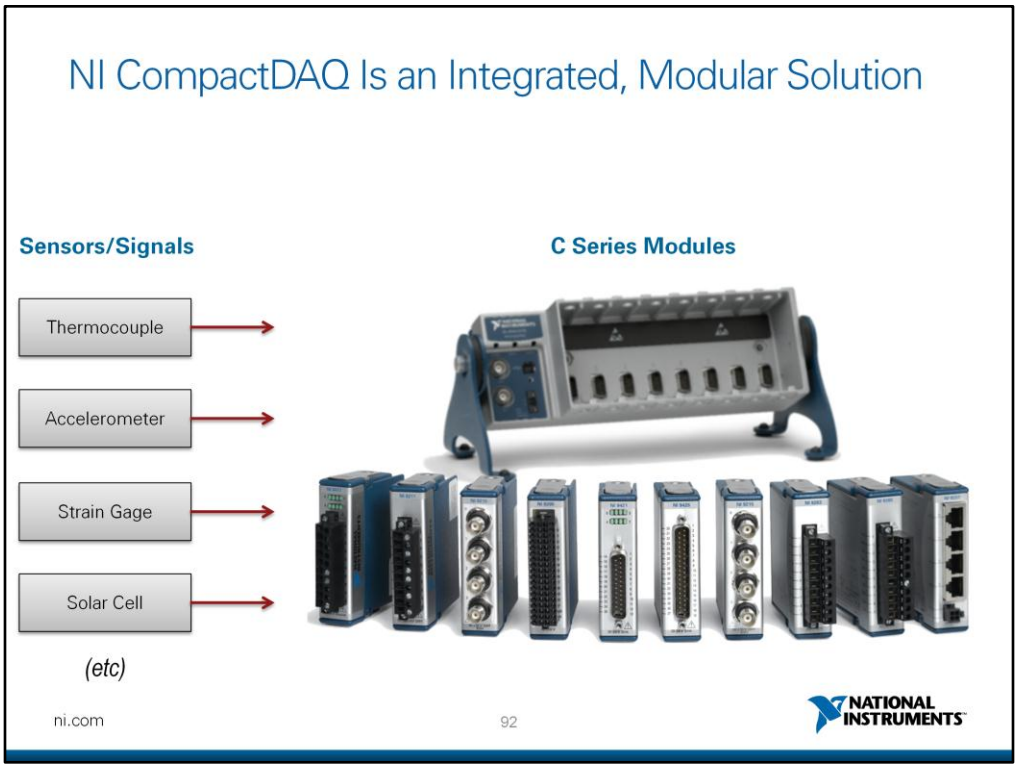


Today, NI's data acquisition hardware portfolio contains hundreds of devices across a spectrum of price points and functionality. NI offers a range of individual multifunction DAQ devices in several form factors, but for guaranteed scalability and flexibility, consider one of National Instruments' modular hardware platforms such as PXI and NI CompactDAQ, which comes in available USB, Ethernet, and Wi-Fi communication buses.

ni.com/labview
ni.com/daq

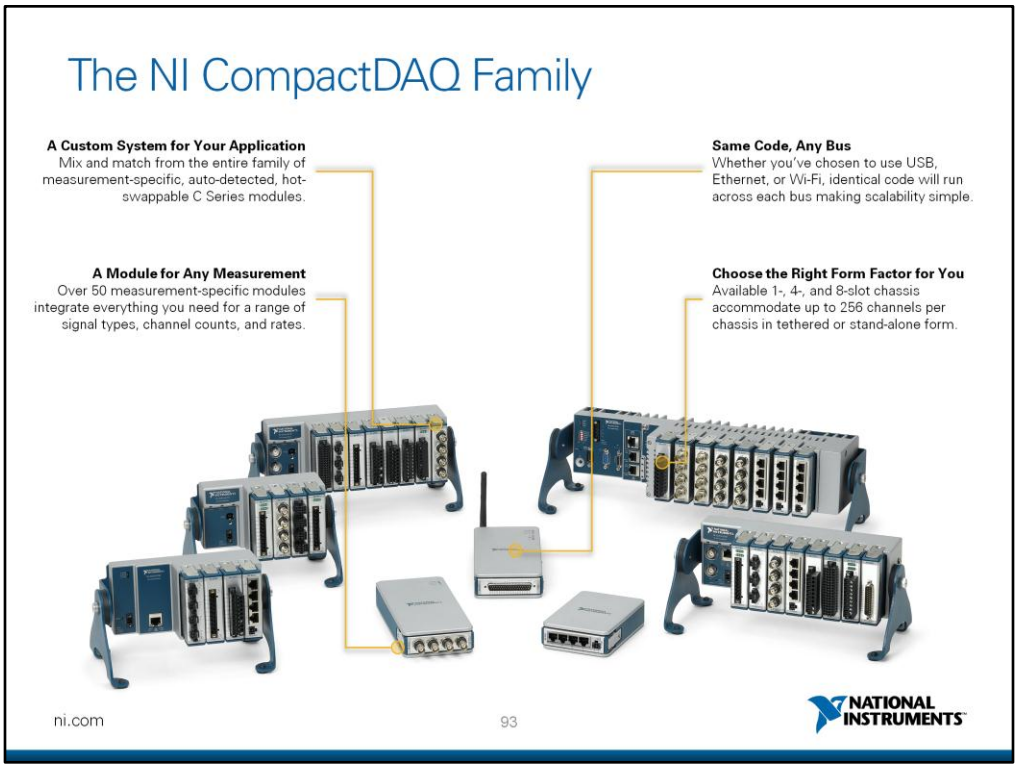


As we've learned so far in today's seminar, a traditional measurement system includes a lot of science, electronics, and hardware behind each type of sensor measurement in a mixed-measurement system. Here you can see four common measurements and some of the hardware involved to bring them into a modern test system. Notice the differences in connectivity and signal conditioning required for each sensor type; sensors such as thermocouples, strain gages, load cells, and accelerometers require different types of signal conditioning in order for them to operate properly. If you are working with high voltage signals (greater than +/– 10 V), you will also need signal conditioning to attenuate the signal so that it can be safely measured by your DAQ device.



Our hardware platform of choice for today's seminar is one of our scalable, modular systems. With C Series, we've dedicated each module option to a specific type of measurement. If you want to measure temperature, we have thermocouple modules. If you want to measure voltage, there are analog input modules. Each dedicated module is integrated, meaning that these data acquisition devices combine all of the necessary components (physical connectivity, signal conditioning) into single devices and all you need to do is supply the sensor and signal. This modularity – with each module completely taking care of all the needs for a specific measurement – enables better interoperability of multiple sensor types in a single system. Furthermore, most C Series modules can also be used on the RIO platform from National Instruments, in case you need the determinism of a real-time OS or the reconfigurability of an FPGA.

Insider Tip!
While C Series is an excellent and scalable platform, PXI provides yet another option for even larger systems and provides similar scalability benefits.



The NI CompactDAQ family provides a flexible, expandable platform to meet the needs of any PC-based electrical or sensor measurement system. The system is comprised of sensors, a chassis, one or more NI C Series modules, and a host PC.

The CompactDAQ Chassis controls the timing, synchronization, and data transfer between a host computer and up to eight C Series I/O modules. A single CompactDAQ chassis can manage multiple timing engines to run up to seven separate hardware-timed I/O tasks at different sample rates within the same system.

CompactDAQ chassis exist across a variety of buses including USB, Ethernet, and Wi-Fi. Because of its flexibility, CompactDAQ is used across dozens of industries and is applicable to nearly every automated measurement application.

ni.com/labview
ni.com/daq

Family Highlight: Stand-Alone NI CompactDAQ

Embedded Measurements and Logging

- >50 I/O modules
- Up to 24-bit, Up to 1 MS/s
- Dual-core processor
- 32 GB nonvolatile storage
- 0 to 55 °C Operating Temp
- 5g shock, 30g vibration
- Windows or Real-Time OS
- LabVIEW and NI-DAQmx

ni.com 94 NATIONAL INSTRUMENTS

For high-performance embedded measurement and logging applications, the NI CompactDAQ family also includes a stand-alone version of the hardware platform that unites modular, flexible C Series I/O with an integrated Intel Core i7 processor and onboard storage.

ni.com/labview
ni.com/daq

C Series I/O Modules

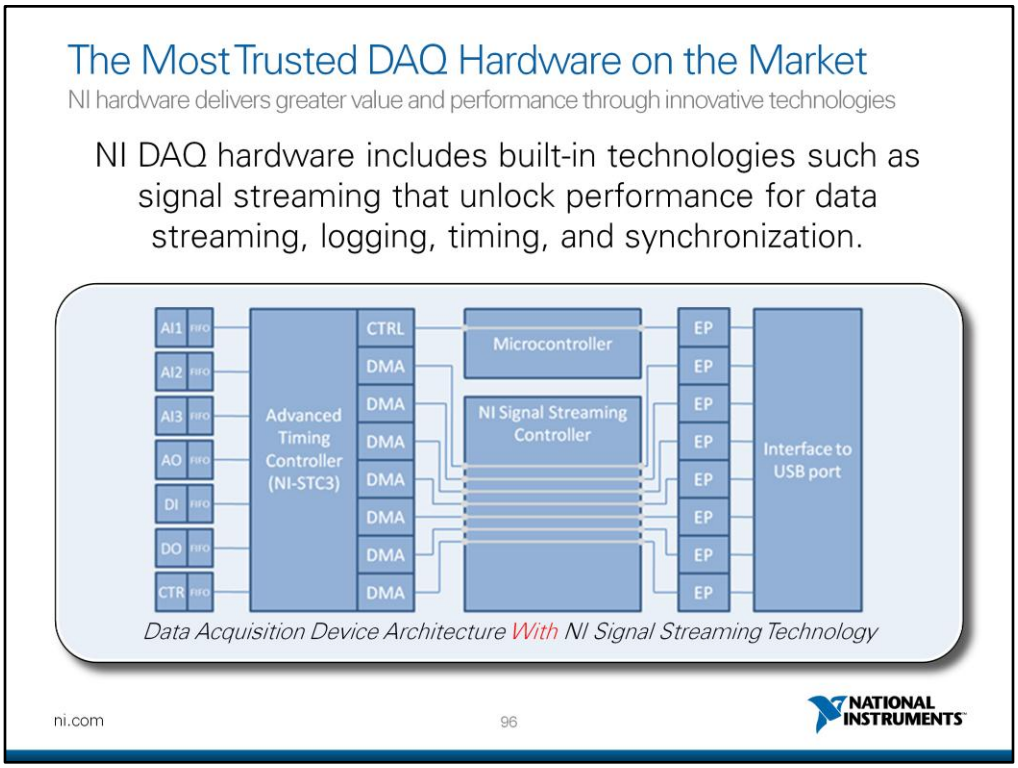
- Over 100 NI and Partner Modules
  - Analog Input
  - Analog Output
  - Digital I/O
  - Relay Output
  - Counter, Pulse Generation
  - Communication
    - CAN
    - LIN
    - PROFIBUS
  - Motion Control
  - Wireless
  - Engine Control
- Signal Conditioning
- Rugged Mechanicals
- Signal Conditioning/Filtering
- Isolation Barrier

ni.com 95 NATIONAL INSTRUMENTS

Each C Series module integrates all of the circuitry necessary for its application. It connects to the chassis backplane through a VGA style connector on the back of the module. In the case of an analog C Series module, the amplifier, filters, and the analog-to-digital converter are all contained on the module circuitry. The module then passes digital values over the backplane back to the controlling PC.

The modules listed on this slide are only a subset of the total number of modules. NI has developed over 50 measurement-specific modules, however, C Series is an open platform, so third party companies also create modules for certain applications.

C Series has a module for nearly every sensor or actuator application, and all of the complicated, application-specific signal conditioning that we already learned about is built right into the module. Whereas traditional DAQ hardware often requires separate signal conditioning and amplification (which means the user has to connect 2-3 times as many wires), we can directly connect the sensor into the CompactDAQ module and that is the only connection required.



Before the innovation of NI Signal Streaming – a patented technology only available to NI USB data acquisition devices – each item in the data acquisition front end (AI/AO/DIO, etc.) had to route data through the processor to the various endpoints (EP) in order to ultimately interface with the USB port. Multiple endpoints enabled multi-tasking, but because of the multiplexed timing controller and processor architecture, performance suffered when you tried to do more than one process at a time – a common task in a mixed-measurement system.

NI Signal Streaming technology introduced the ability to multitask streaming. With NI Signal Streaming, the processor microcontroller is now separate from the data flow; it receives the task, interprets, sets up the generation or acquisition then gets out of the way. The Signal Streaming technology also includes I/O DMAs, which can be thought of independent FIFOs for a specific task and not for the whole device.

The controller can receive the command, set up the task, and then move data directly from the DMA to the USB end-point where it then reaches the PC. This reduces overhead on the controller, allows you to configure one task while another one is running, and achieve drastically higher rates of data streaming for mixed-measurement applications performed using USB.

Increasing Proficiency in NI Software and Hardware

	Self-Paced	Instructor-Led		
Format Features	**Online** Prerecorded modules viewable at ni.com	**Online** 1- to 4-day classes held live remotely	**Regional** 1- to 3-day classes held at training facilities	**On-Site** 1- to 3-day classes held at your location
Learn from a certified instructor	–	✓	✓	✓
Access relevant hardware	–	✓	✓	✓
Eliminate distractions with a classroom setting	–	–	✓	✓
Interact with other students	–	–	✓	✓
Content modified to meet your group's needs	–	–	–	✓
Avoid travel expenses	✓	✓	–	✓
Printed manual that accompanies the course	✓	✓	✓	✓
Exercises to practice concepts you learn	✓	✓	✓	✓
Multimedia training	✓	✓	–	–
Concept review quizzes	✓	✓	✓	✓
Class duration	–	Half-Day	Full-Day	Full-Day
Price	$	$$	$$$	$$$

ni.com 97 NATIONAL INSTRUMENTS

NI training courses are the fastest, most certain route to productivity and successful application development with LabVIEW and NI data acquisition hardware. With NI training courses, you learn recommended techniques to reduce development time and improve application performance and scalability. Customers on average report 66% faster learning, 50% quicker development, and 43% less maintenance after taking NI training courses. NI courses are taught by experienced, certified instructors in live classroom and online settings and are available 24/7 online as recorded videos. NI training is a smart and safe investment to unlocking your application development potential.

For more information on NI learning paths and training options to increase your proficiency, visit **ni.com/training**.

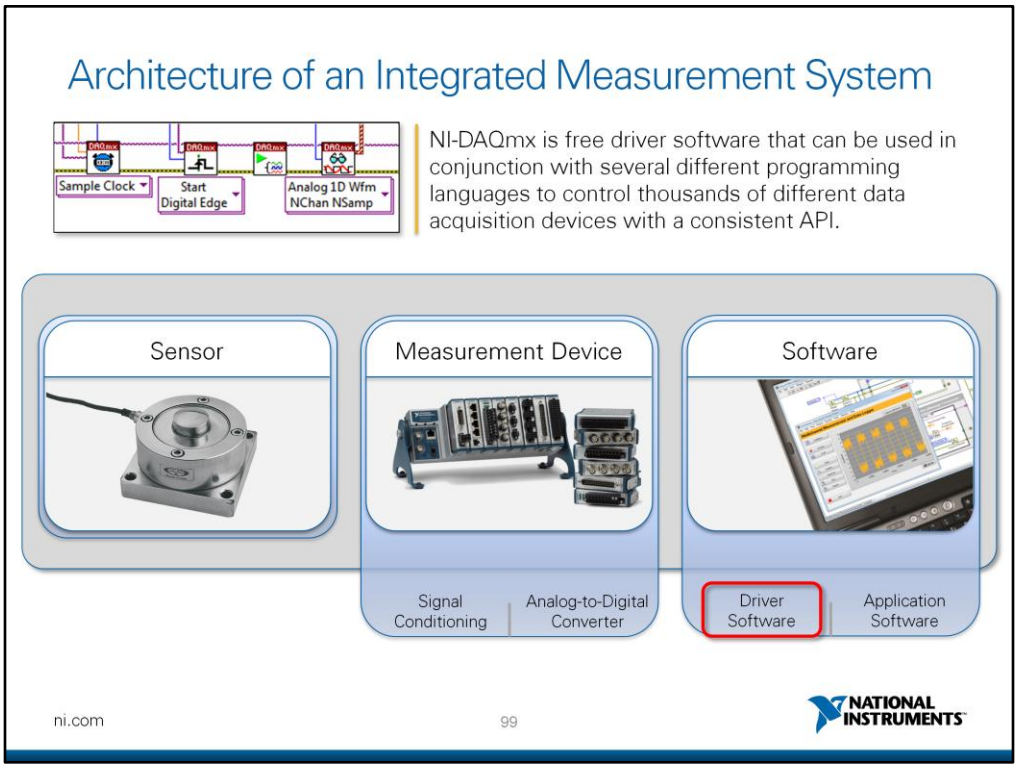
ni.com/labview
ni.com/daq

Combining Software and Hardware Into an Integrated System

Automated Measurement Solutions With LabVIEW and NI DAQ

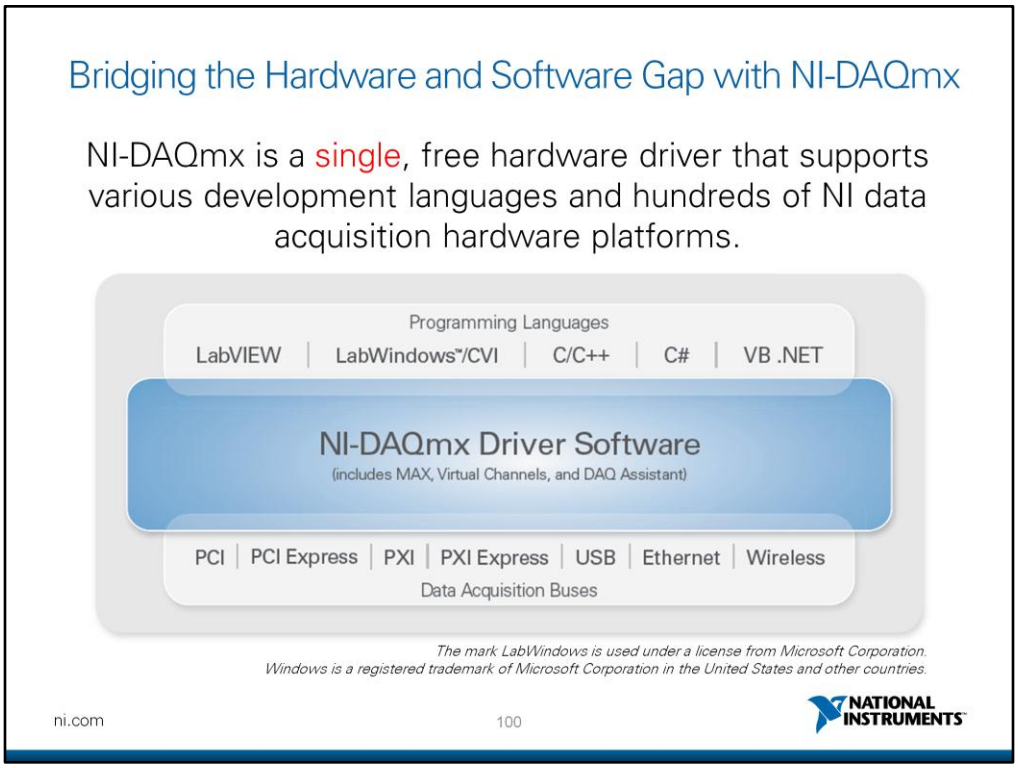
ni.com

NATIONAL INSTRUMENTS



One final piece remains before we fully understand the most imperative components of an integrated measurement system. The link between application software and measurement device is the driver software – NI-DAQmx.

ni.com/labview
ni.com/daq



NI-DAQmx driver software goes far beyond a basic DAQ driver to deliver increased productivity and performance and is one of the main reasons National Instruments continues to be the trusted leader in PC-based data acquisition.

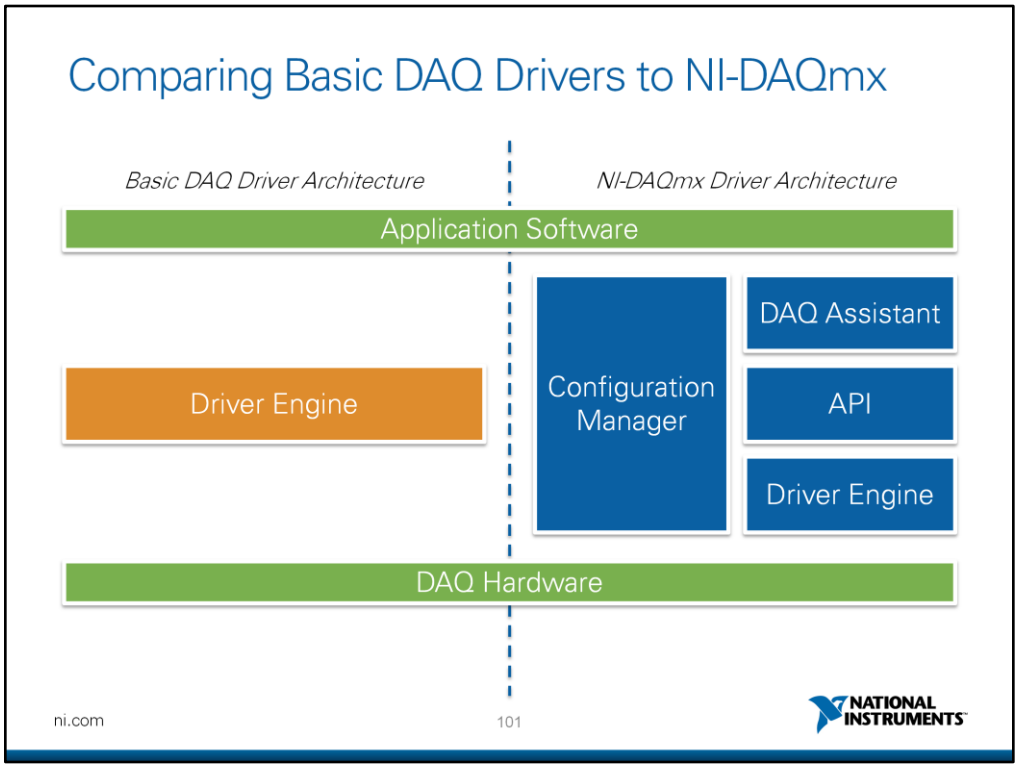
NI-DAQmx offers tools for both new and existing DAQ users including configuration-based and full programmatic interfaces, a sophisticated configuration management interface, robust documentation and hundreds of examples.

The driver is supported across a variety data acquisition buses and programming languages, which means that an identical VI can be used to program PCI, PCI Express, PXI, PXI Express, USB, Ethernet, and Wireless NI DAQ devices. It also means that an identical programming paradigm can be used in LabVIEW, LabWindows™/CVI, C, C++, C#, and VB.NET. This makes altering or augmenting a system remarkably simple, because you only have to learn the driver paradigm once yet can use it across hardware devices and programming languages.

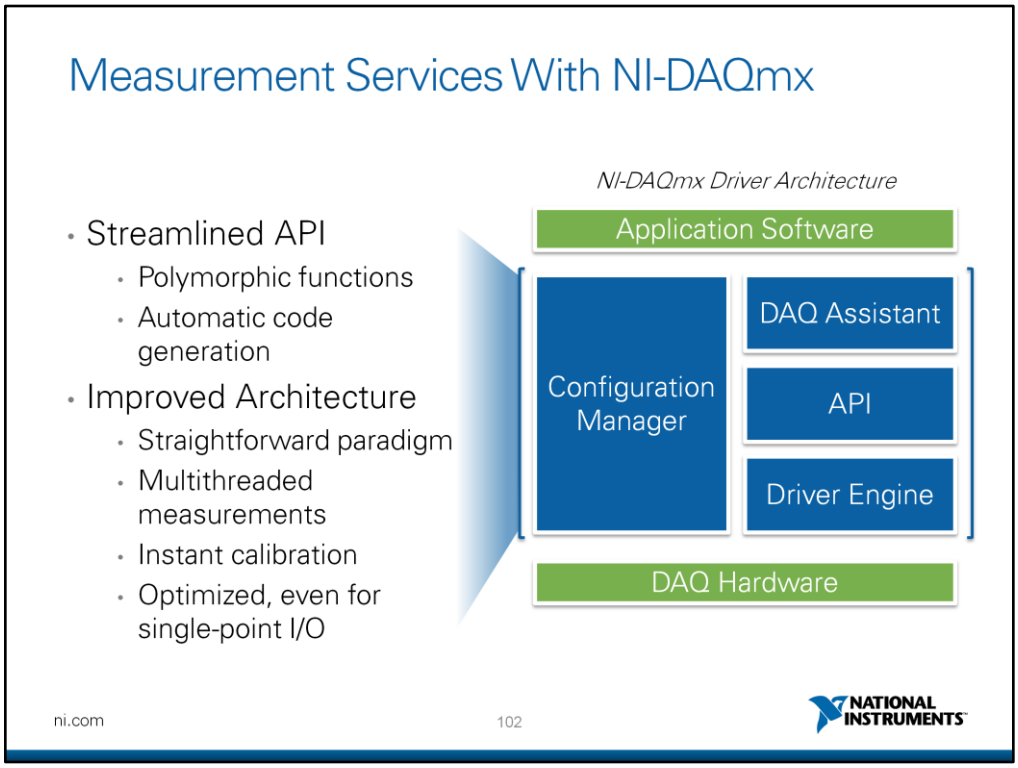
Example files are always free and always installed for any development environments that are present on the machine at time of install of NI-DAQmx.

Insider Tip!
The latest version of NI-DAQmx is hosted online at ni.com/drivers and the driver traditionally updates four times a year (though this is not guaranteed).

ni.com/labview
ni.com/daq



Basic data acquisition driver software serves the simple purpose of connecting your DAQ device with your software application. It provides for basic configuration, programming, and device control. This functional representation of a Basic DAQ driver describes what NI-DAQ was back in 1991. If you survey the marketplace, this is still a good representation of most DAQ drivers provided by many hardware vendors. So how is NI-DAQmx different from these basic drivers?

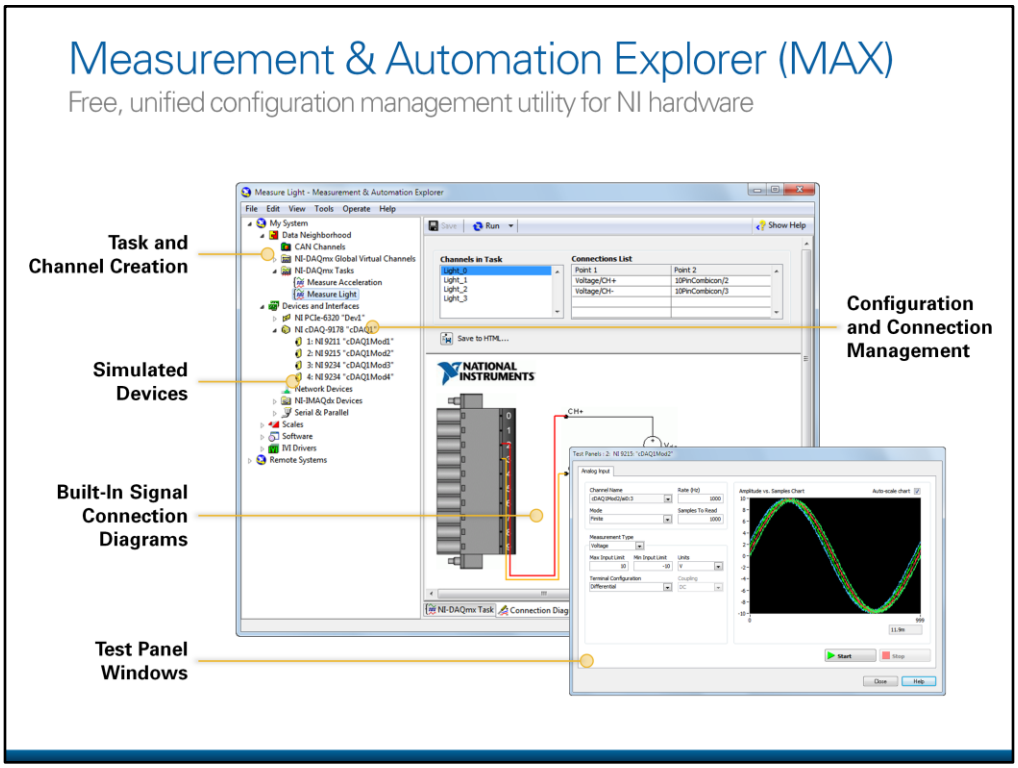


Fundamentally, NI-DAQmx still performs as a driver and connects your hardware with your application software, but there is so much more that contributes to making NI-DAQmx the single most powerful data acquisition driver available.

The NI-DAQmx API was built on 20 years of industry feedback in order provide the most simplified, flexible software interface to your DAQ hardware. The driver API supports advanced features such as polymorphism, which allows to make a single function call (for example, *DAQmx Read*) even though a number of instances of the function may exist to support the variety of operations the function is designed to handle (*Read* one sample of analog input on one channel, one sample on multiple channels, multiple samples on one channel, and so on).

Additionally, NI-DAQmx driver engine technologies enable the highest performance from your DAQ device. They eliminate the hidden costs of not achieving the desired DAQ specs by enabling advanced features absent from other DAQ drivers such as buffering, optimized single-point I/O, and multi-threaded measurements.

It is the goal of the entire measurement services software layer to make the hardware "disappear" with respect to your software application. Though we've devoted time to understanding hardware details in this seminar, the NI measurement platform allows you to spend little to no time learning about hardware or its intricacies; instead, you should focus only on your measurement.

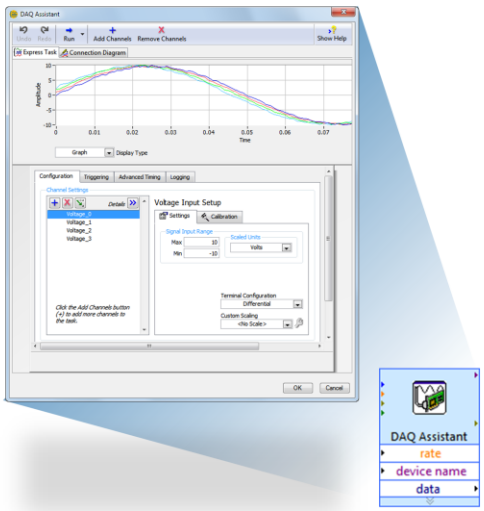


Measurement and Automation Explorer (MAX) is a configuration management interface built into NI-DAQmx measurement services software that you can use to view and configure all measurement software and hardware in your system. MAX allows you to set properties and parameters of hardware, verify hardware setup with device self-tests and test panel windows, and self-calibrate data acquisition hardware. In addition, it provides connection diagrams for many NI accessories so you can connect your sensors and signals correctly the first time.

Insider Tip!
Configure and use MAX simulated devices in order to trick your system into thinking your measurement hardware is present when it isn't; this way you can develop nearly your entire application before your hardware even arrives so that when it does, all you have to do is plug it in.

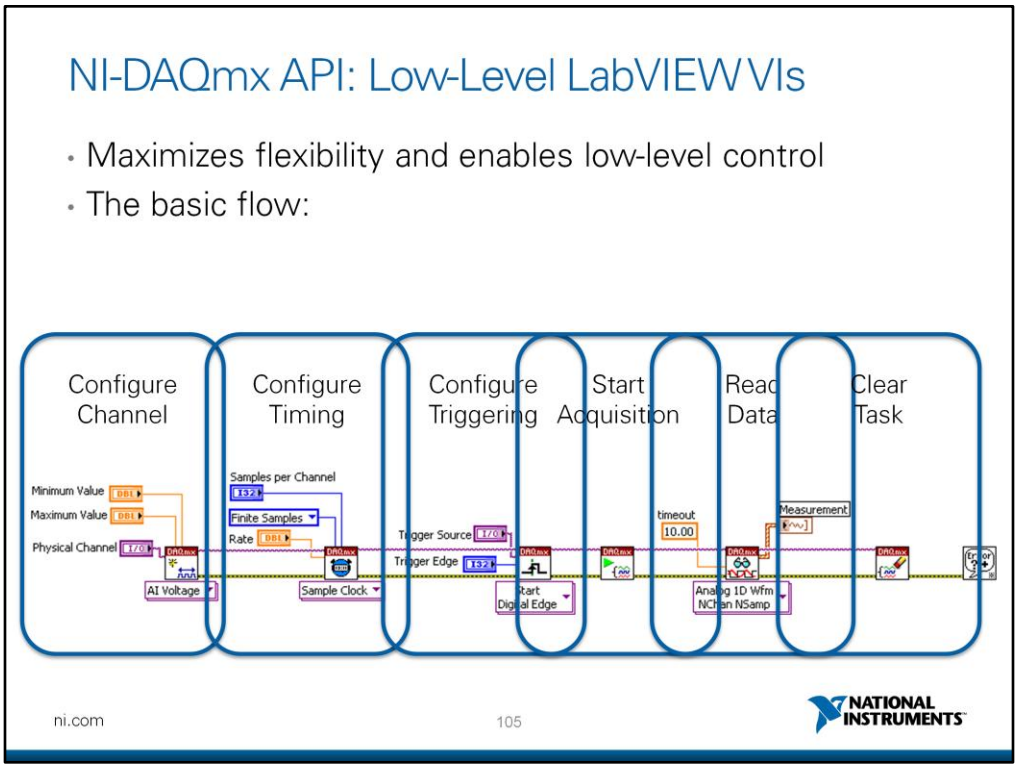
NI-DAQmx API: Configuration-Based DAQ Assistant

- Enables quick, configuration-based measurements
- Usable across multiple channels, multiple devices
- Maximum ease of use with some sacrificed flexibility
- Supported across multiple programming languages
- Automatically generates lower-level code

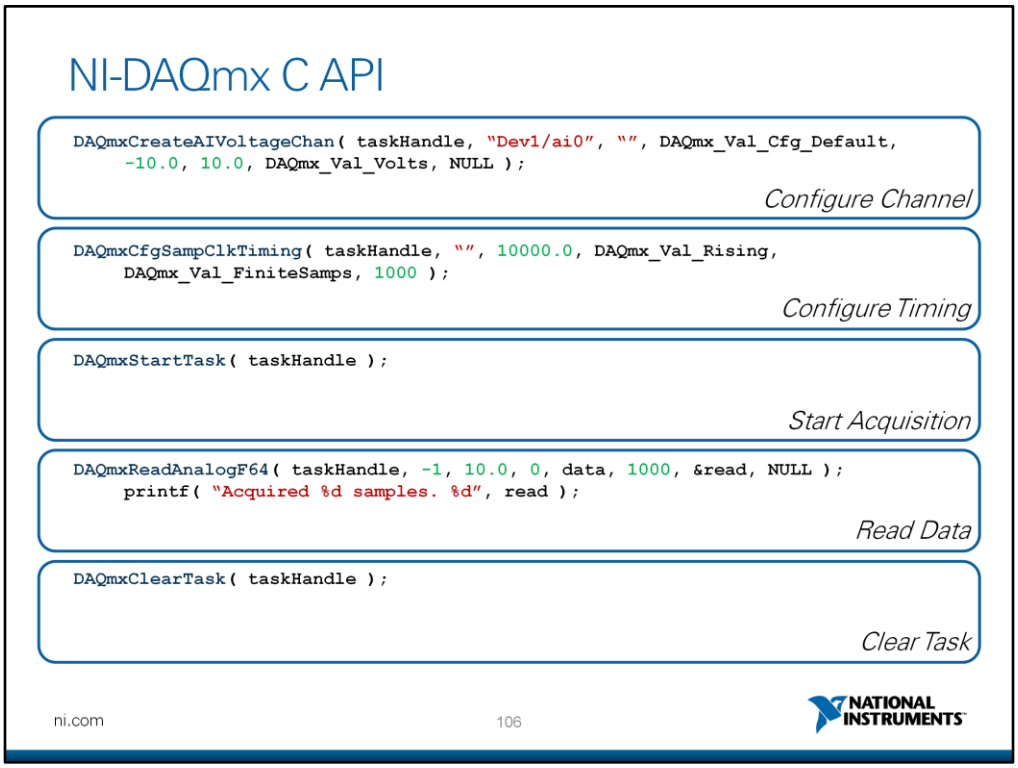


You can use NI-DAQmx to program measurement applications with several programming paradigms, the simplest of which is the DAQ Assistant. This is a configuration-based wizard that installs automatically with NI-DAQmx allows you to quickly and easily take simple measurements – even using multiple channels across multiple devices or modules. This approach that provides maximum ease-of-use, but does sacrifice some flexibility for those who want granular, lower-level control.

ni.com/labview
ni.com/daq



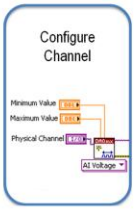
For those who do want lower-level, granular control of the details – particularly important when creating a synchronized or simultaneous mixed-measurement system – the NI-DAQmx API provides lower-level VIs that allow you to programmatically configure all details of the acquisition or generation.

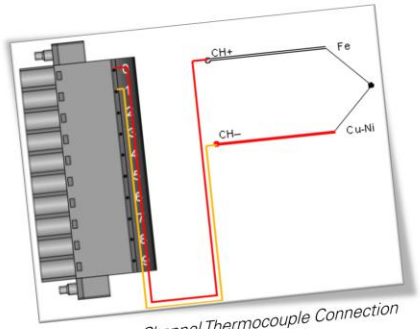


The NI-DAQmx API paradigm is standardized across programming languages, so the exact same functions and paradigm are applicable in C (and other languages) in the instance that you choose an alternate programming approach for your application. This means that you can learn the basic NI-DAQmx paradigm once and be comfortable applying the paradigm in different languages (not counting syntactical differences between languages, of course).

NI-DAQmx Channels

- NI-DAQmx channels encompass:
  - Measurement type, sensor/signal type
  - Terminal configuration
  - Physical connection settings
  - Name
  - Min/Max Value
    - Used to determine amplification level
  - Custom Scaling
    - Ex: thermocouple generates a mV signal; NI-DAQmx upscales to °C



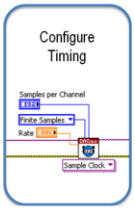


NI-DAQmx Channel Thermocouple Connection Diagram

NI-DAQmx channels allow you to configure all details about a physical channel in the system and encompass everything from terminal configuration (RSE, differential, etc.) to scaling.

Timing

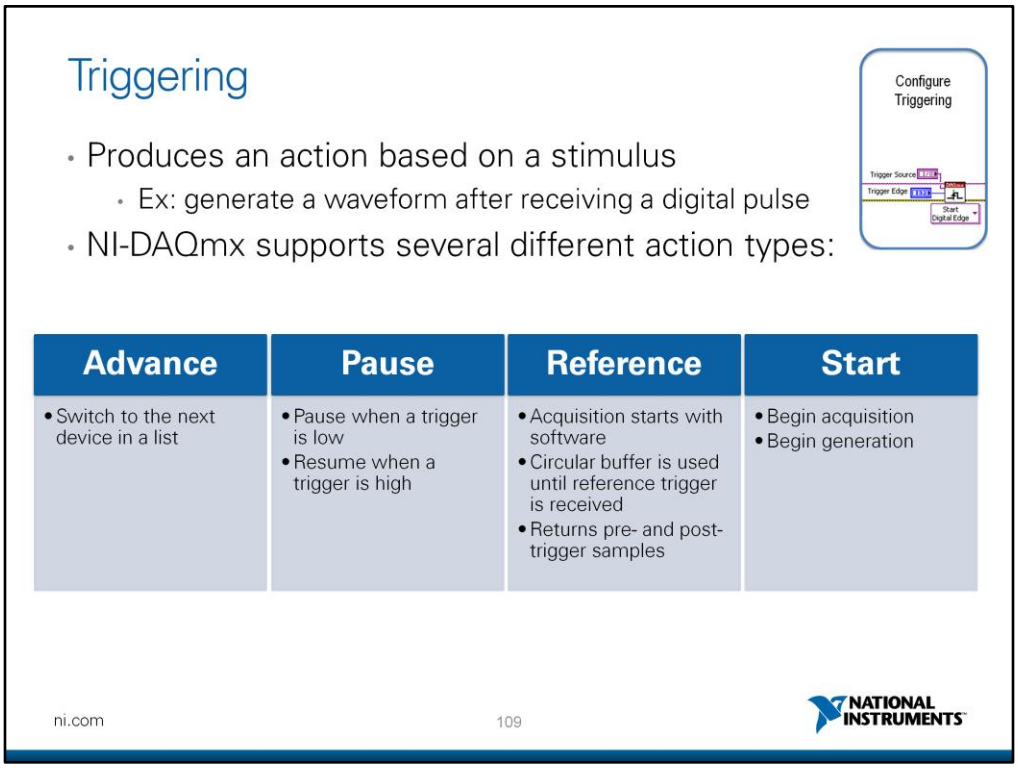
- Allows you to configure acquisition timing
- Set sample clock, rate of acquisition, and number of samples to acquire or generate



Timing Option	Description
Finite Samples	Acquire or generate a configurable number of samples at a configurable rate.
Continuous Samples	Acquire or generate samples continuously, until explicitly stopped by the API.
Hardware-Timed Single Point	Acquire or generate samples continuously on the edge of a hardware clock.

The *Task Timing VI* allows you to configure the timing for the acquisition or generation task. Every channel in the task will be affected by these timing settings – to use multiple timing configurations, you must use different tasks with independent timing VIs. For an analog input task, the timing settings allow you to acquire a single sample (On Demand), a single sample acquired on an edge of a hardware clock (HW Timed), N samples (finite acquisition), or continuous samples, as well as specifying the number of samples to read and sampling rate (when acquiring N samples or continuously). In addition, you can configure the clock settings you wish to use to control your task.

Each type of task (analog input, analog output, counter, digital) has different settings for the task timing VI.

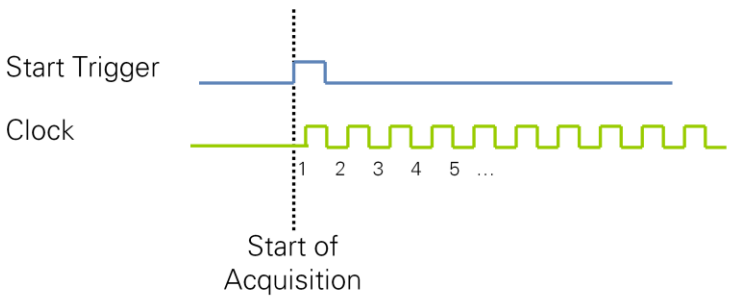


Whenever an NI-DAQmx device does something, we call that an action. Two example actions include generating waveform output and acquiring data. In order for each action to occur, there must be a cause, or stimulus. Causes for action are called *triggers*.

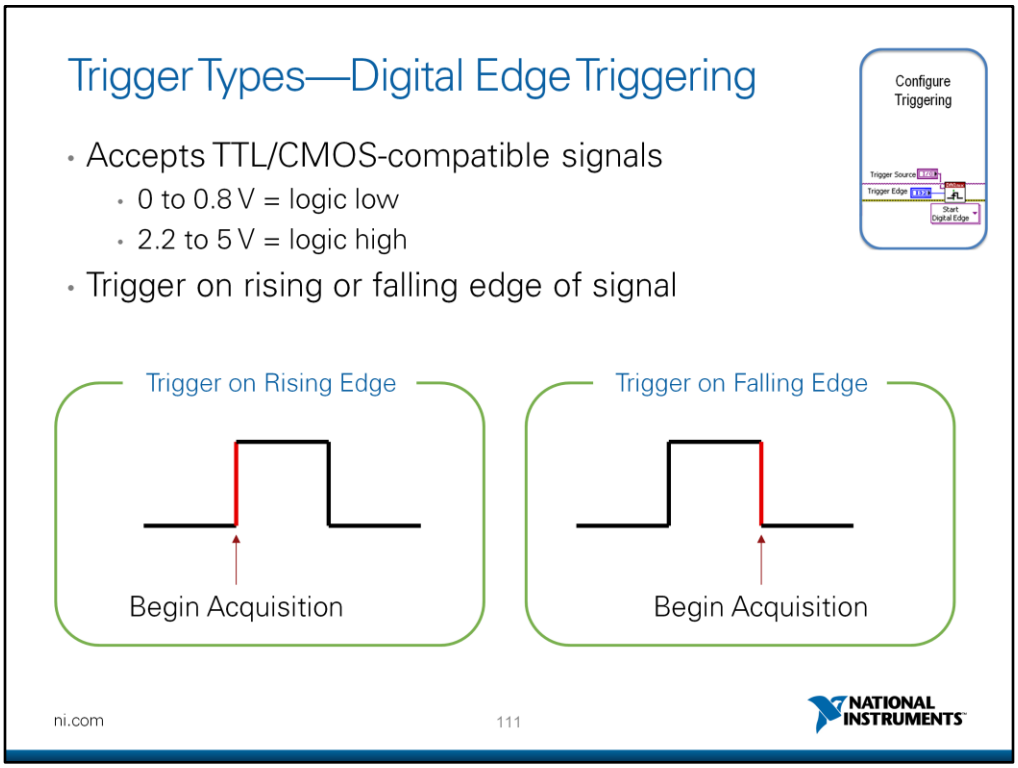
A trigger is named after the action it causes and the way that it was produced. Possible actions are advance, pause, reference, or start trigger.
The way that the trigger was produced also determines its name. This production method can be either analog or digital.

Triggering

- Event-driven acquisition or generation
- Valid for finite or continuous operations
- Example: acquire 5 samples on a start trigger:



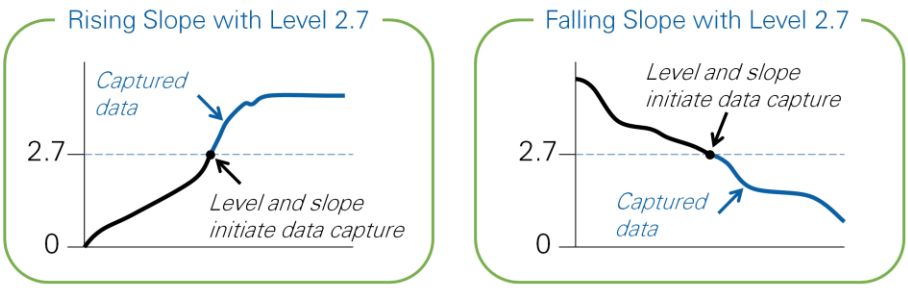
In this simple visualization of a start trigger, 5 samples are read in – one with every rising clock edge. The acquisition does not start until the first rising edge on the start trigger.



A digital trigger is a TTL (transistor-transistor logic) signal that is composed of two discrete levels: logic low and logic high. The logic low state can range between the values of 0 to 0.8 V. The logic high state can range between the values of 2.0 to 5.0 V. The edge that transitions between the logic low state to the logic high state is called the rising edge. Likewise, the edge that transitions between the logic high state to the logic low state is called the falling edge. You can set your acquisition to start as a result of either the rising or falling edge of your digital trigger signal.

Trigger Types—Analog Edge Triggering

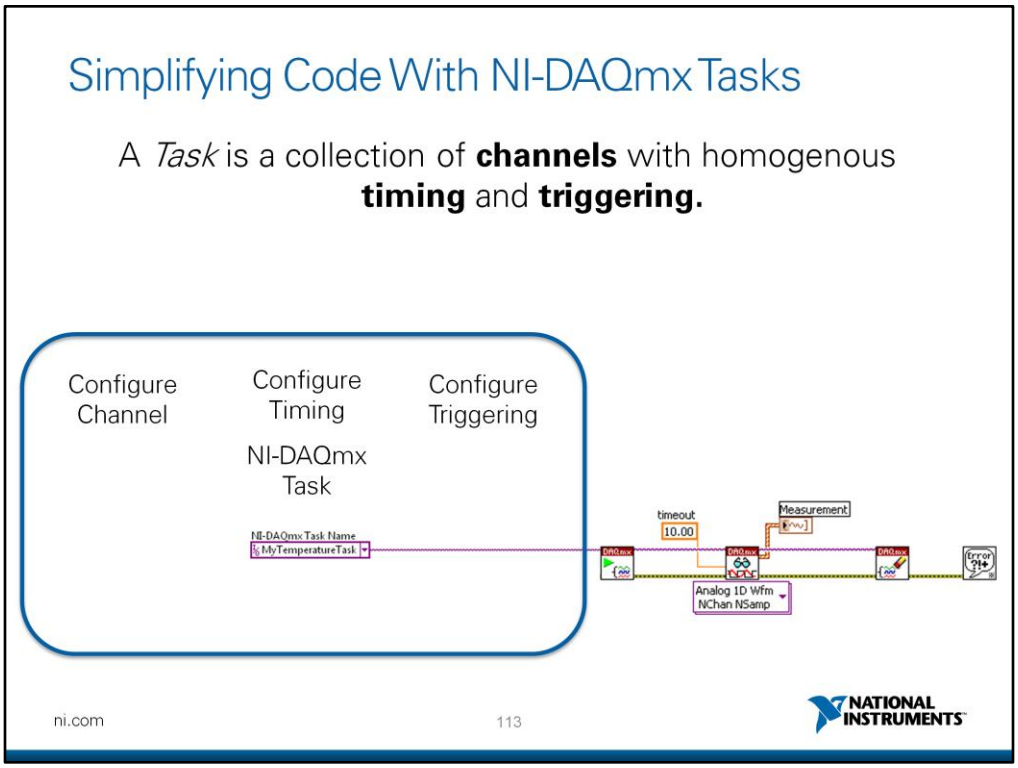
- Trigger off signal level and slope
- Slope can be rising or falling



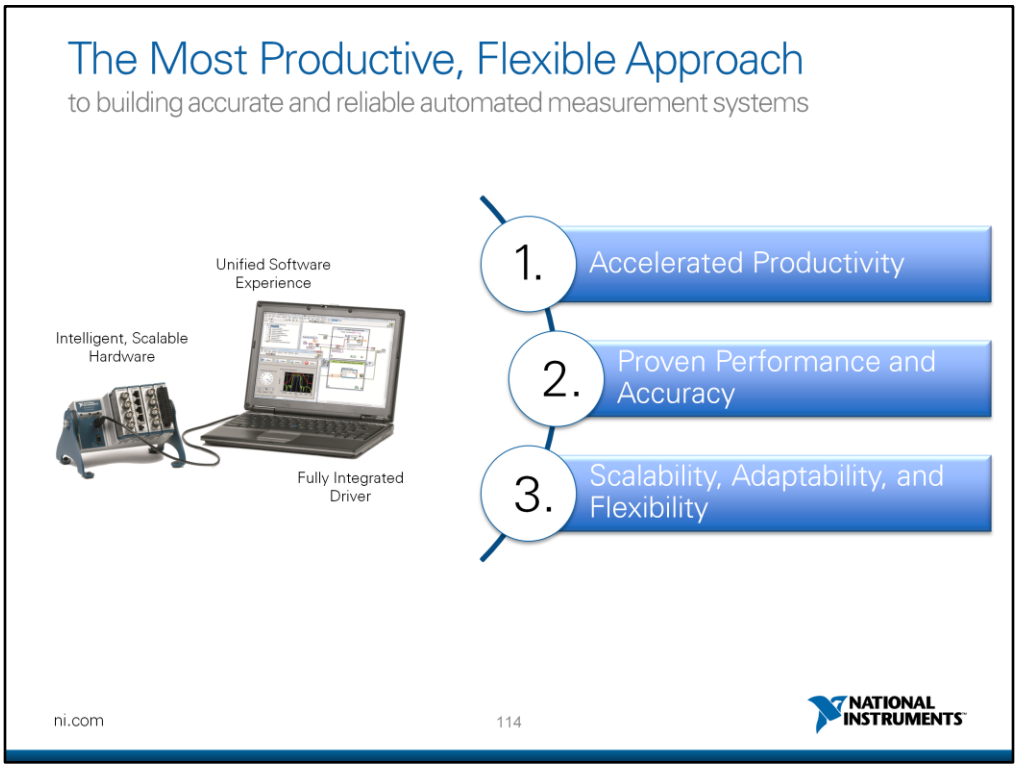
Analog triggering can be based off of an edge value or a window value. The edge value consists of a slope and level.

The slope is defined as either rising or falling and the level is the value (a voltage level) at which the acquisition should start or stop. A typical application using an analog hardware trigger could be a temperature monitoring system where the acquisition starts only when the temperature rises above a certain value. The trigger signal can be wired to either the PFI0 pin or to an analog input channel.

Your DAQ device monitors the analog trigger channel until trigger conditions are met. For example, if the trigger conditions are set to Slope = Rising and Level = 2.7, data capture will begin once the first data point rises equal to 2.7. Likewise for the settings of Slope = Falling and Level = 2.7, the capture will begin when the data falls to 2.7.



NI-DAQmx lets you programmatically configure each lower level channel, timing and triggering detail with the lower-level VIs, or you can use a configuration-based utility to predefine these settings in what is called an NI-DAQmx Task. As you can see, the NI-DAQmx API provides a wide range of programming and configuration options depending on the level of detail and flexibility you need, but all options work together seamlessly and you can mix-and-match them as your application requires.



In today's seminar, we've developed a solid foundation in the building blocks of the graphical system design approach which allows you to focus on building a working measurement system without having to understand multiple tool-chains or integrate disparate resources.

- NI LabVIEW is a single software environment that natively integrates NI DAQ and third party hardware and includes extensive libraries for signal processing and data visualization needed to build any measurement system.
- NI DAQ is the most trusted computer-based measurement hardware for engineers and scientists.

By combining LabVIEW software with NI DAQ hardware, you'll be able to create powerful, reusable measurement solutions that you can program and re-configure to meet your evolving needs.

We appreciate your time and participation in today's seminar. To engage with us at a later date, please don't hesitate to visit us on the web at **ni.com/labview** or **ni.com/daq**.