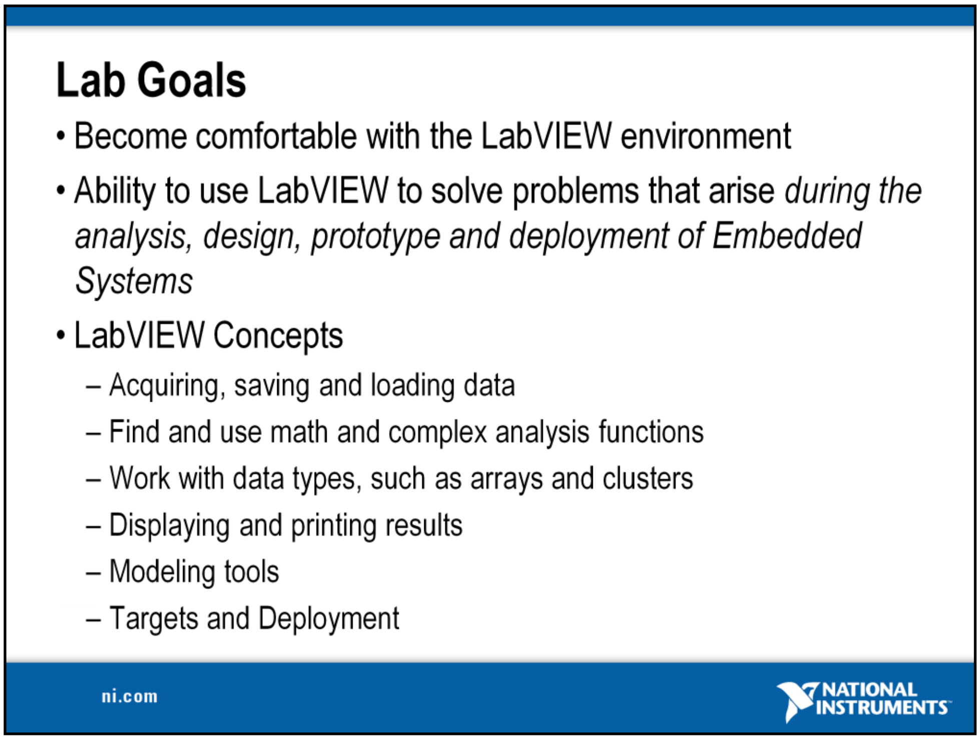
Introduction to LabVIEW
For Use in Embedded System Development

UC Berkeley EECS124 Lab 1

Hugo.Andrade@ni.com

ni.com



Lab Goals

- Become comfortable with the LabVIEW environment
- Ability to use LabVIEW to solve problems that arise *during the analysis, design, prototype and deployment of Embedded Systems*
- LabVIEW Concepts
  - Acquiring, saving and loading data
  - Find and use math and complex analysis functions
  - Work with data types, such as arrays and clusters
  - Displaying and printing results
  - Modeling tools
  - Targets and Deployment

ni.com

NATIONAL INSTRUMENTS

This is a list of the objectives of the course.

This course prepares you to do the following:

- Use LabVIEW to create applications.
- Understand front panels, block diagrams, and icons and connector panes.
- Use built-in LabVIEW functions.
- Create and save programs in LabVIEW so you can use them as subroutines.
- Create applications that use plug-in DAQ devices.

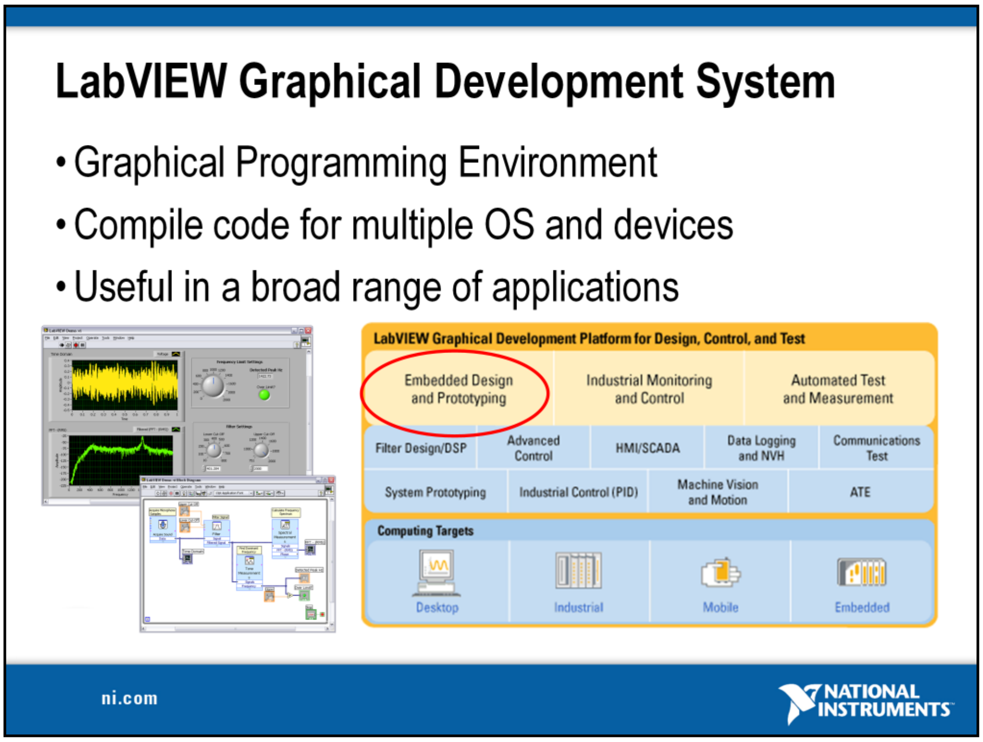
This course does *not* describe any of the following:

- Programming theory
- Every built-in LabVIEW function or object
- Analog-to-digital (A/D) theory

NI does provide free reference materials on the above topics on `ni.com`.

The *LabVIEW Help* is also very helpful:

LabVIEW»Help»Search the LabVIEW Help…



LabVIEW Graphical Development System

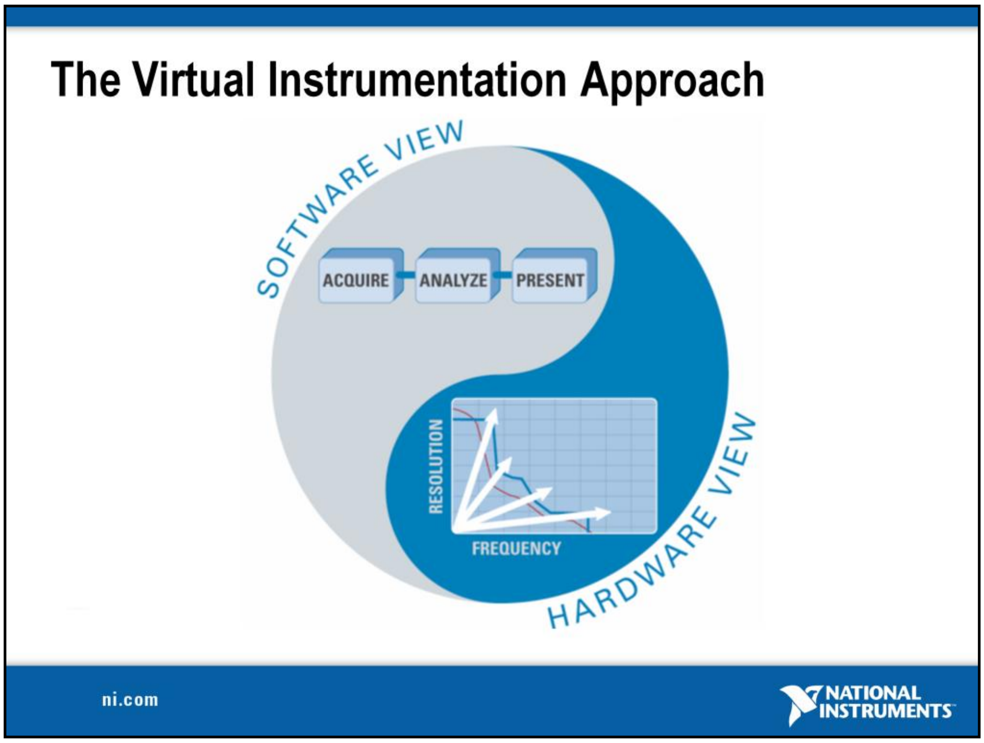
- Graphical Programming Environment
- Compile code for multiple OS and devices
- Useful in a broad range of applications

LabVIEW Graphical Development Platform for Design, Control, and Test
Embedded Design and Prototyping
Industrial Monitoring and Control
Automated Test and Measurement
Filter Design/DSP
Advanced Control
HMI/SCADA
Data Logging and NVH
Communications Test
System Prototyping
Industrial Control (PID)
Machine Vision and Motion
ATE
Computing Targets
Desktop
Industrial
Mobile
Embedded
ni.com
NATIONAL INSTRUMENTS

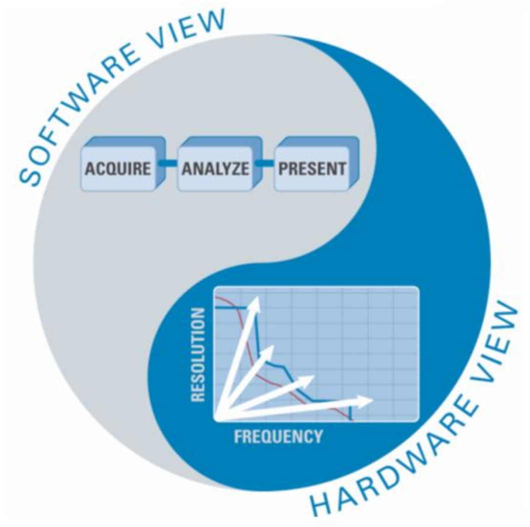
National Instruments LabVIEW is an industry-leading software tool for designing test, measurement, and control systems. Since its introduction in 1986, engineers and scientists worldwide who have relied on NI LabVIEW graphical development for projects throughout the product design cycle have gained improved quality, shorter time to market, and greater engineering and manufacturing efficiency. By using the integrated LabVIEW environment to interface with real-world signals, analyze data for meaningful information, and share results, you can boost productivity throughout your organization. Because LabVIEW has the flexibility of a programming language combined with built-in tools designed specifically for test, measurement, and control, you can create applications that range from simple temperature monitoring to sophisticated simulation and control systems. No matter what your project is, LabVIEW has the tools necessary to make you successful quickly.

Some of the applications have the following in common:

- Physical measurements
- Analysis and signal processing
- Decision making
- Data logging
- Report generation



The Virtual Instrumentation Approach

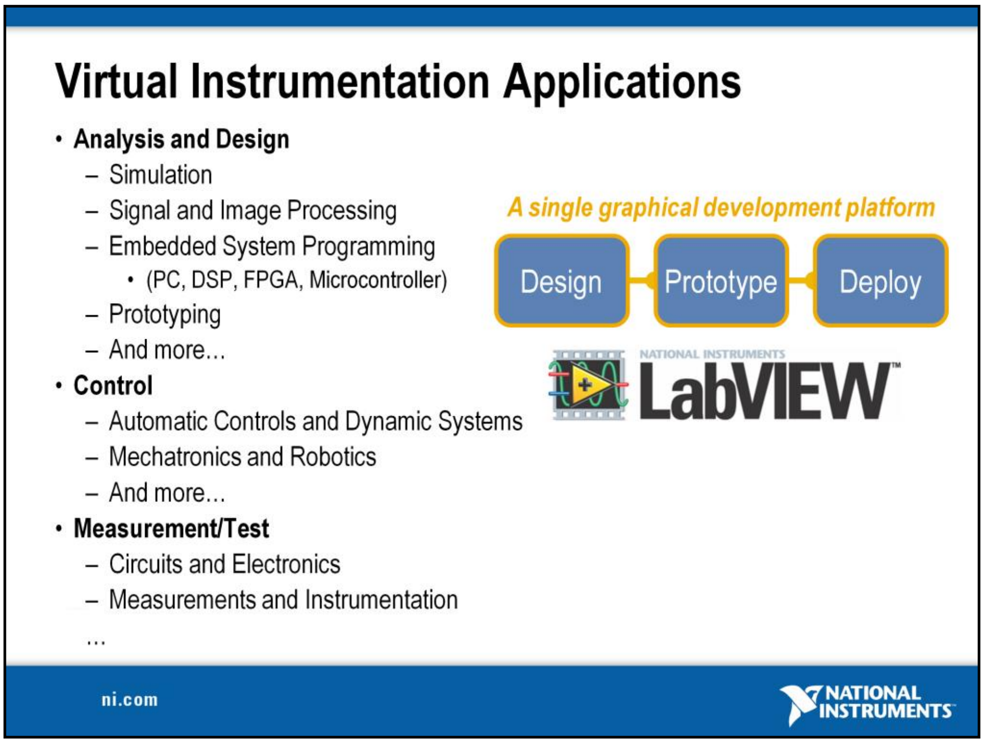


ni.com

Virtual Instrumentation

For more than 25 years, National Instruments has revolutionized the way engineers and scientists in industry, government, and academia approach measurement and automation. Leveraging PCs and commercial technologies, virtual instrumentation increases productivity and lowers costs for test, control, and design applications through easy-to-integrate software, such as NI LabVIEW, and modular measurement and control hardware for PXI, PCI, USB, and Ethernet.

With virtual instrumentation, engineers use graphical programming software to create user-defined solutions that meet their specific needs, which is a great alternative to proprietary, fixed functionality traditional instruments. Additionally, virtual instrumentation capitalizes on the ever-increasing performance of personal computers. For example, in test, measurement, and control, engineers have used virtual instrumentation to downsize automated test equipment (ATE) while experiencing up to a 10 times increase in productivity gains at a fraction of the cost of traditional instrument solutions. Last year 25,000 companies in 90 countries invested in more than 6 million virtual instrumentation channels from National Instruments.



Virtual Instrumentation Applications

- **Analysis and Design**
  - Simulation
  - Signal and Image Processing
  - Embedded System Programming
    - (PC, DSP, FPGA, Microcontroller)
  - Prototyping
  - And more…
- **Control**
  - Automatic Controls and Dynamic Systems
  - Mechatronics and Robotics
  - And more…
- **Measurement/Test**
  - Circuits and Electronics
  - Measurements and Instrumentation

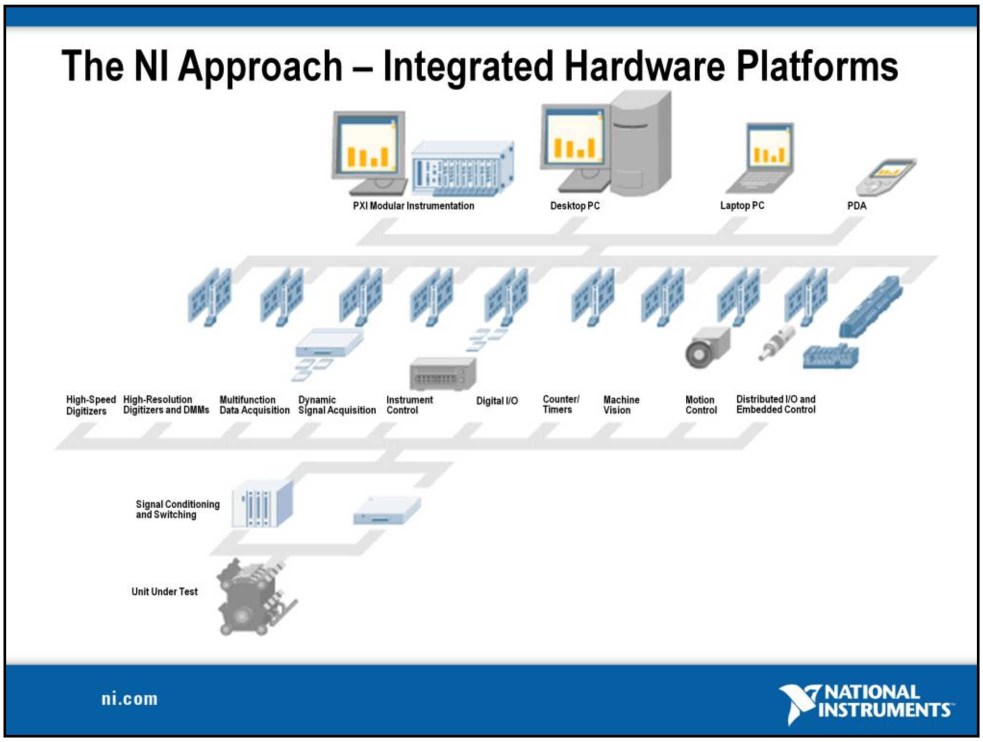
…

A single graphical development platform
Design
Prototype
Deploy
NATIONAL INSTRUMENTS LabVIEW
ni.com
NATIONAL INSTRUMENTS

Virtual Instrumentation Applications

Virtual instrumentation is applicable in many different types of applications, starting from design to prototyping and deployment. The LabVIEW platform provides specific tools and models to solve specific applications ranging from designing signal processing algorithms to making voltage measurements and can target any number of platforms from the desktop to embedded devices – with an intuitive, powerful graphical paradigm.

With version 8, LabVIEW scales from design and development on PCs to several embedded targets from ruggedized toaster size prototypes to embedded systems on chips. **LabVIEW streamlines system design with a single graphical development platform**. In doing so, LabVIEW encompasses better management of distributed, networked systems because as the targets for LabVIEW grow varied and embedded, you will need to be able to more easily distribute and communicate between various LabVIEW code pieces in your system.



The NI Approach – Integrated Hardware Platforms

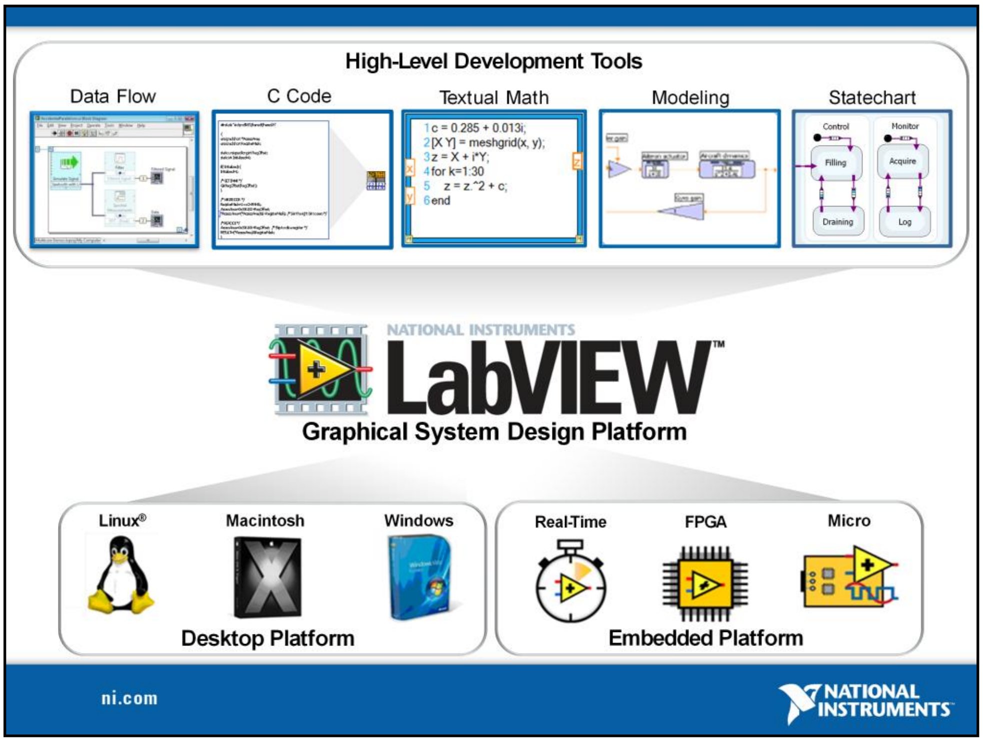
PXI Modular Instrumentation
Desktop PC
Laptop PC
PDA
High-Speed Digitizers
High-Resolution Digitizers and DMMs
Multifunction Data Acquisition
Dynamic Signal Acquisition
Instrument Control
Digital I/O
Counter/ Timers
Machine Vision
Motion Control
Distributed I/O and Embedded Control
Signal Conditioning and Switching
Unit Under Test
ni.com
NATIONAL INSTRUMENTS

Integrated Hardware Platforms

A virtual instrument consists of an industry-standard computer or workstation equipped with powerful application software, cost-effective hardware such as plug-in boards, and driver software, which together perform the functions of traditional instruments.

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 popular desktop computers and workstations.

Although the PC and integrated circuit technology have experienced significant advances in the last two decades, software truly offers the flexibility 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).

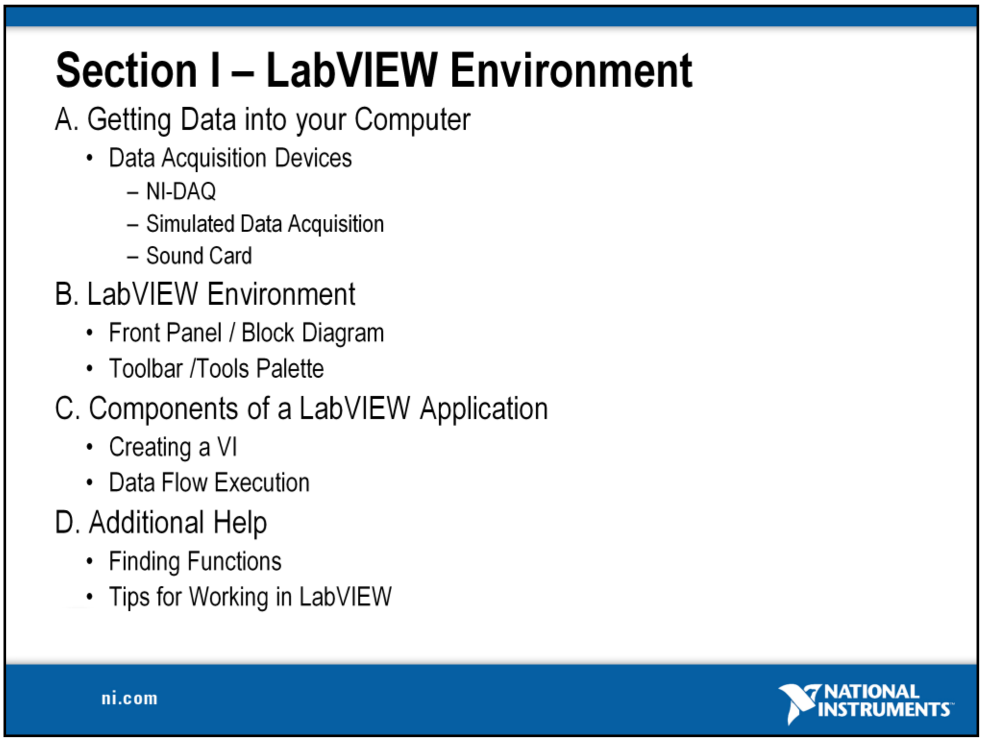


LabVIEW is a graphical system design platform that provides many techniques for solving problems in a single development environment.

Many of you may not be aware of this, but if you can program in LabVIEW, then you can program much more than desktop systems.

Quiz the audience as to who has used LV on Mac, Linux, RT, FPGA, Embedded, etc

The ability to create applications using a combination of development techniques enables scientist and engineers to choose best technique for each aspect of their application. The ability to deploy the applications to variety of custom and off-the-shelf IO hardware targets enables the domain expert to not only design their application, but implement it in the real-world.



Section I – LabVIEW Environment

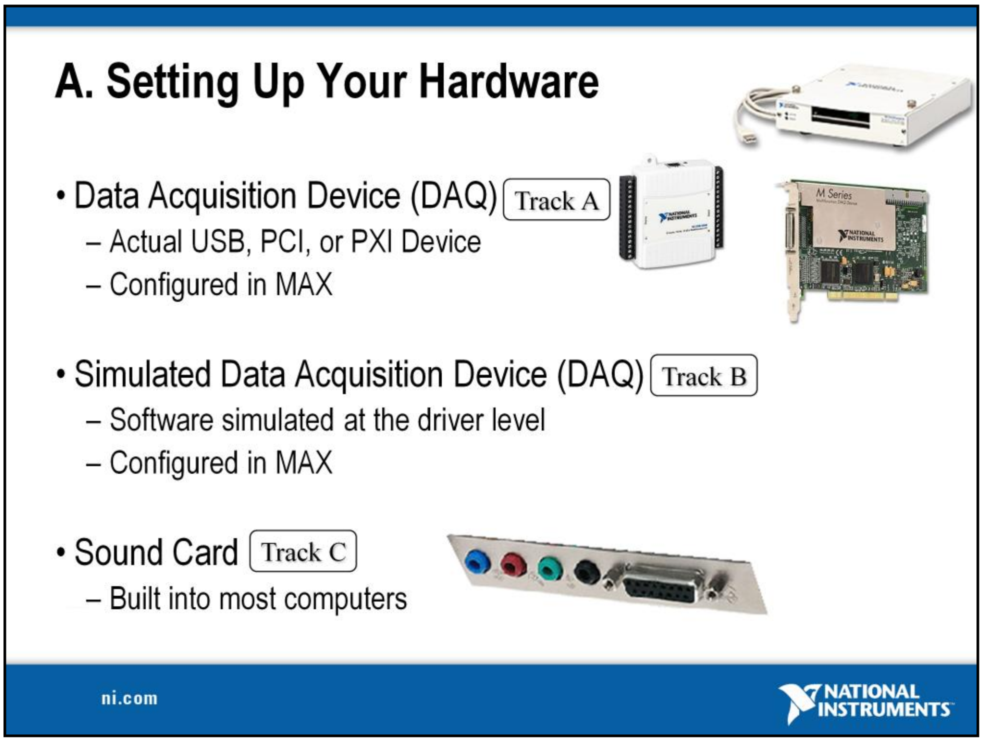
A. Getting Data into your Computer
- Data Acquisition Devices
  - NI-DAQ
  - Simulated Data Acquisition
  - Sound Card

B. LabVIEW Environment
- Front Panel / Block Diagram
- Toolbar /Tools Palette

C. Components of a LabVIEW Application
- Creating a VI
- Data Flow Execution

D. Additional Help
- Finding Functions
- Tips for Working in LabVIEW

ni.com



A. Setting Up Your Hardware

- Data Acquisition Device (DAQ) Track A
  - Actual USB, PCI, or PXI Device
  - Configured in MAX
- Simulated Data Acquisition Device (DAQ) Track B
  - Software simulated at the driver level
  - Configured in MAX
- Sound Card Track C
  - Built into most computers

ni.com

This LabVIEW course is designed for audiences with or without access to National Instruments hardware.

Each exercise is divided into three tracks, A, B, and C:

Track A is designed to be used with hardware supported by the National Instruments DAQmx driver. This includes most USB, PCI, and PXI data acquisition devices with analog input. Some signal conditioning and excitation is required to use a microphone with a DAQ device. (Some sensors, like microphones, require external power to work (excitation)).

Track B is designed to be used with no hardware. Hardware can be simulated with the NI-DAQmx Driver Version 7.5 and newer. A simulated NI-DAQmx device is a replica of a device created using the NI-DAQmx Simulated Device option in the Create New menu of MAX for the purpose of operating a function or program without hardware. An NI-DAQmx simulated device behaves similar to a real device. Its driver is loaded, and programs using it are fully verified.

Track C is designed to be used with a standard sound card and microphone. LabVIEW includes simple VIs for analog input and analog output using the soundcard built into many PCs. (This is very convenient for laptops because the soundcard and microphone are usually already built-in.)

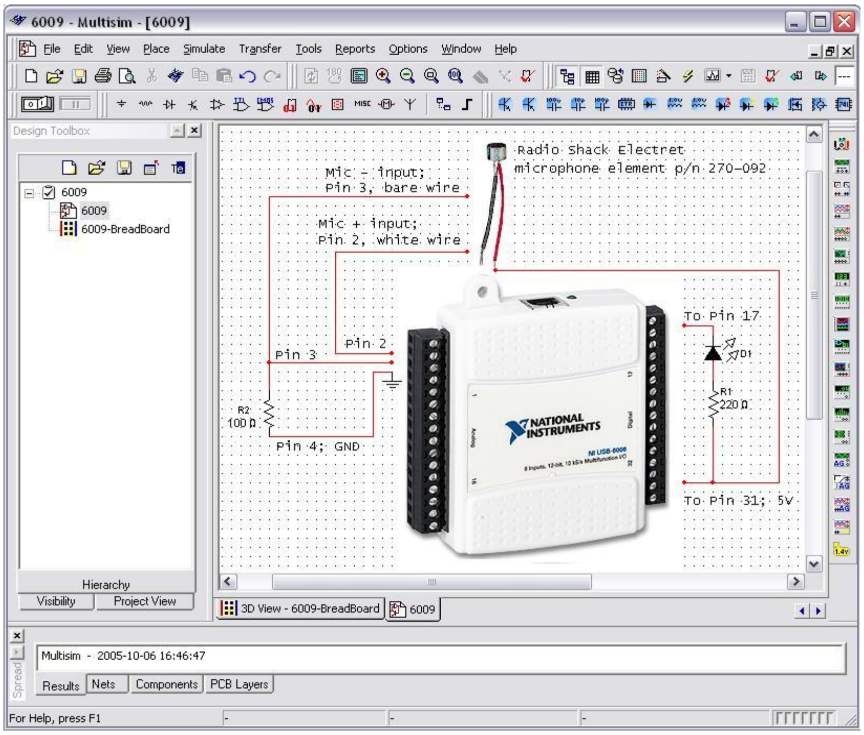
Setting-Up Your Hardware for Your Selected Track

Track A – NI Data Acqusion with Microphone: USB-6009 with Microphone & LED

Suggested Hardware:

Qty	Part Number	Description	Supplier
1	779321-22	Low-cost USB DAQ	National Instruments
1	270-092	Electret Microphone	RadioShack
1		100 Ohm Resistor	RadioShack
1		220 Ohm Resistor	RadioShack
1	276-307	Light Emitting Diode (LED)	RadioShack

The following schematic was drawn with **Multisim**, a widely used SPICE schematic capture and simulation tool. Visit http://www.electronicsworkbench.com for more info.



Multisim - 2005-10-06 16:46:47

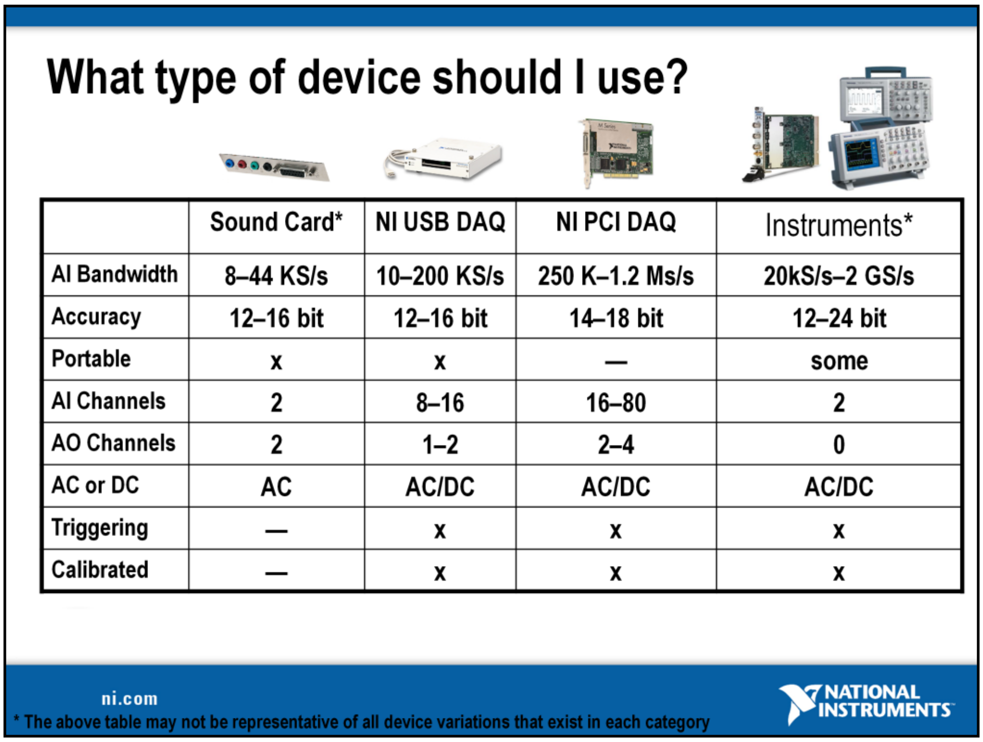
Track B – Simulated NI Data Acqusion: NI-DAQ Software version 8.0 or newer

Track C – Third-party Soundcard: Soundcard and Microphone

Suggested Hardware:

Qty	Part Number	Description	Supplier
1		Standard Plug-in PC Microphone*	RadioShack

* Laptops often have a built-in microphone (no plug-in microphone is required)



What type of device should I use?

	Sound Card*	NI USB DAQ	NI PCI DAQ	Instruments*
AI Bandwidth	8–44 KS/s	10–200 KS/s	250 K–1.2 Ms/s	20kS/s–2 GS/s
Accuracy	12–16 bit	12–16 bit	14–18 bit	12–24 bit
Portable	x	x	—	some
AI Channels	2	8–16	16–80	2
AO Channels	2	1–2	2–4	0
AC or DC	AC	AC/DC	AC/DC	AC/DC
Triggering	—	x	x	x
Calibrated	—	x	x	x

ni.com

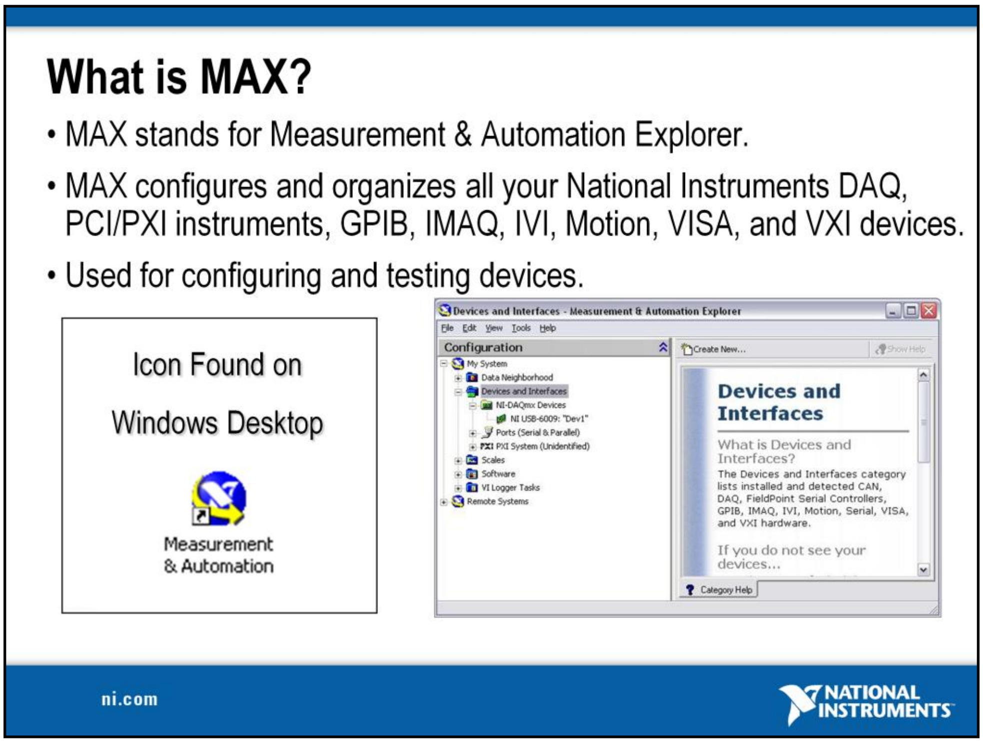
* The above table may not be representative of all device variations that exist in each category

What type of device should I use?
There are many types of data acquisition and control devices on the market. A few have been highlighted above. The trade-off usually falls between sampling rate (samples/second), resolution (bits), number of channels, and data transfer rate (usually limited by "bus" type: USB, PCI, PXI, etc.). Multifunction DAQ (data acqusion) devices are ideal because they can be used in a wide range of applications.

USB-6008 & USB-6009 Low-Cost USB DAQ
The National Instruments USB-6009 provides basic data acquisition functionality for applications such as simple data logging, portable measurements, and academic lab experiments. The NI USB-6008 and NI USB-6009 are ideal for students. Create your own measurement application by programming the NI USB-6009 using LabVIEW and NI-DAQmx driver software for Windows. For Mac OS X and Linux users, download and use the NI-DAQmx Base driver.

NI USB-6009 Specifications:
- Eight 14-bit analog inputs
- 12 digital I/O lines
- 2 analog outputs
- 1 counter

http://www.ni.com/daq/



What is MAX?

- MAX stands for Measurement & Automation Explorer.
- MAX configures and organizes all your National Instruments DAQ, PCI/PXI instruments, GPIB, IMAQ, IVI, Motion, VISA, and VXI devices.
- Used for configuring and testing devices.

Icon Found on Windows Desktop

Devices and Interfaces

What is Devices and Interfaces?

The Devices and Interfaces category lists installed and detected CAN, DAQ, FieldPoint Serial Controllers, GPIB, IMAQ, IVI, Motion, Serial, VISA, and VXI hardware.

If you do not see your devices...

ni.com

The next level of software we are concerned with is called Measurement & Automation Explorer (MAX). MAX is a software interface that gives you access to all of your National Instruments DAQ, GPIB, IMAQ, IVI, Motion, VISA, and VXI devices. The shortcut to MAX will be placed on your desktop after installation. A picture of the icon is shown above. MAX is mainly used to configure and test your National Instruments hardware, but it does offer other functionality such as checking to see if you have the latest version of NI-DAQ installed. When you run an application using NI-DAQmx, the software reads the MAX configuration to determine the devices you have configured. Therefore, you must configure DAQ devices first with MAX.

The functionality of MAX is broken into seven categories:

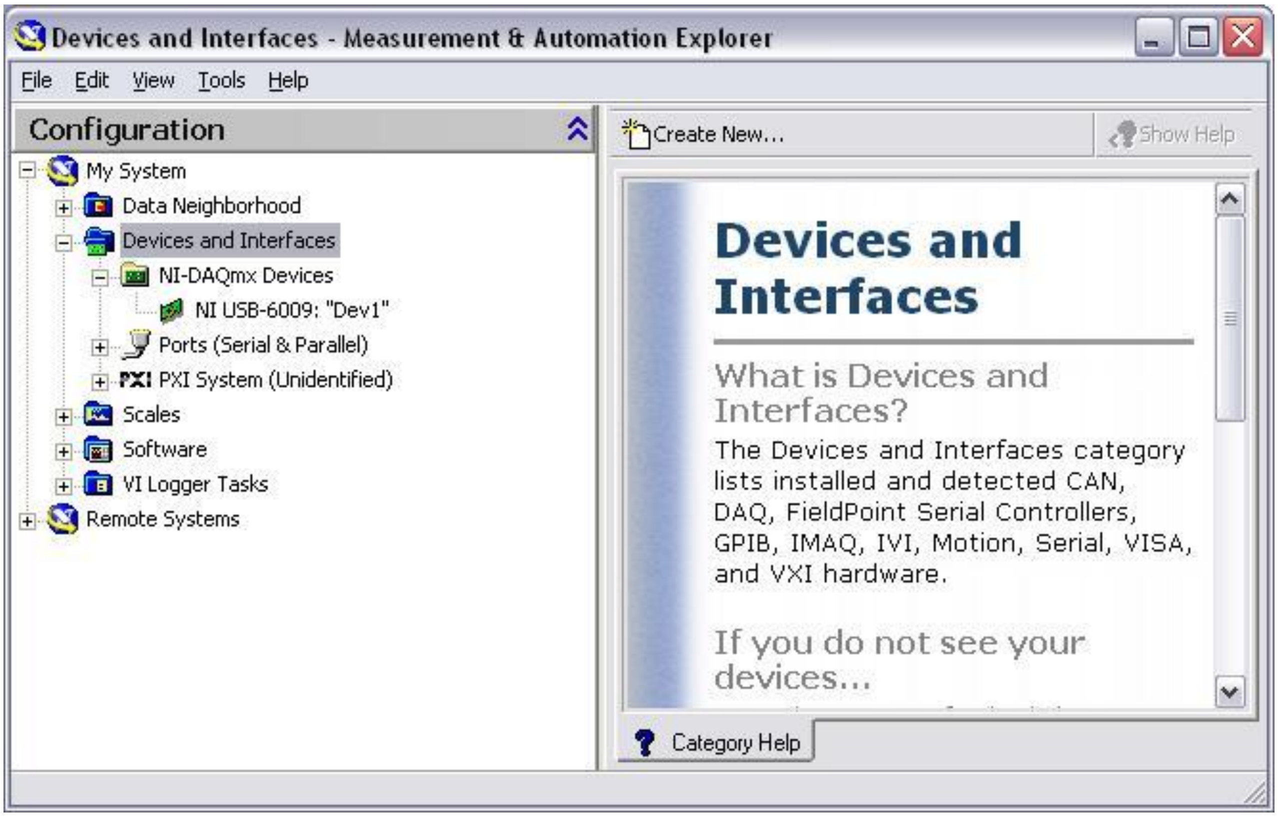
- Data Neighborhood
- Devices and Interfaces
- IVI Instruments
- Scales
- Historical Data
- Software
- VI Logger Tasks

For this course, we will focus on Data Neighborhood, Devices and Interfaces, Scales, and Software. We will now step through each one of these categories and learn about the functionality each one offers.

Exercise 1 – Testing Your Device (Track A)

In this exercise you will use Measurement and Automation Explorer (MAX) to test your NI USB-6009 DAQ device.

1. Launch MAX by double-clicking the icon on the desktop or by selecting **Start»Programs»National Instruments»Measurement & Automation**.
2. Expand the **Devices and Interfaces** section to view the installed National Instruments devices. MAX displays the National Instruments hardware and software in the computer.
3. Expand the **NI-DAQmx Devices** section to view the installed hardware that is compatible with NI-DAQmx. The device number appears in quotes following the device name. The data acquisition VIs use this device number to determine which device performs DAQ operations. You will see your hardware listed as NI USB-6009: “Dev1”.
4. Perform a self-test on the device by right-clicking it in the configuration tree and choosing **Self-Test** or clicking “Self-Test” along the top of the window. This tests the system resources assigned to the device. The device should pass the test because it is already configured.
5. Check the pinout for your device. Right-click the device in the configuration tree and select **Device Pinouts** or click “Device Pinouts” along the top of the center window.
6. Open the test panels. Right-click the device in the configuration tree and select **Test Panels…** or click “Test Panels…” along the top of the center window. The test panels allow you to test the available functionality of your device, analog input/output, digital input/output, and counter input/output without doing any programming.
7. On the **Analog Input** tab of the test panels, change **Mode** to “Continuous” and **Rate** to 10,000 Hz. Click “Start” and hum or whistle into your microphone to observe the signal that is plotted. Click “Finish” when you are done.
8. On the **Digital I/O** tab notice that initially the port is configured to be all input. Observe under **Select State** the LEDs that represent the state of the input lines. Click the “All Output” button under **Select Direction**. Notice you now have switches under **Select State** to specify the output state of the different lines. Toggle line 0 and watch the LED light up. Click “Close” to close the test panels.
9. Close MAX.



Devices and Interfaces

What is Devices and Interfaces?

The Devices and Interfaces category lists installed and detected CAN, DAQ, FieldPoint Serial Controllers, GPIB, IMAQ, IVI, Motion, Serial, VISA, and VXI hardware.

If you do not see your devices...

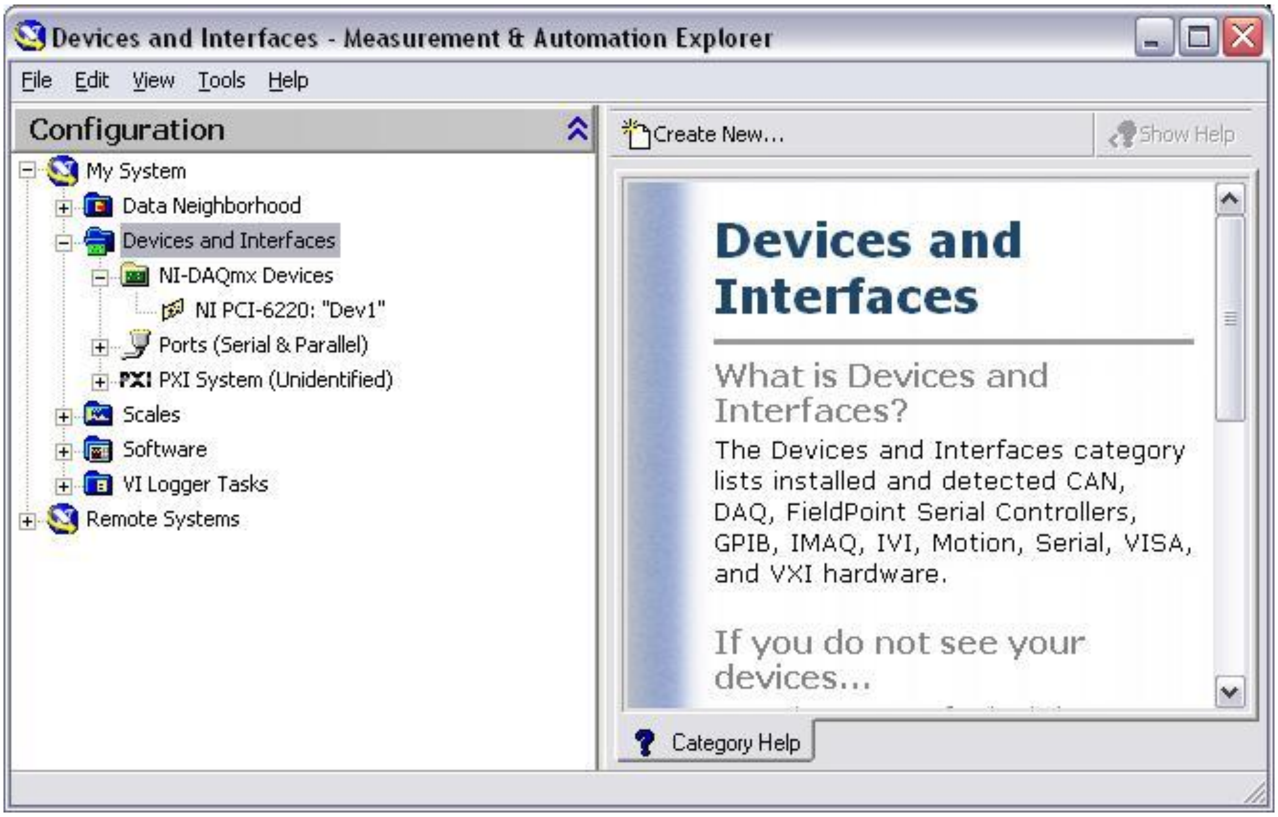
(End of Exercise)

Exercise 1 – Setting Up Your Device (Track B)

In this exercise you will use Measurement and Automation Explorer (MAX) to configure a simulated DAQ device.

1. Launch MAX by double-clicking the icon on the desktop or by selecting **Start»Programs»National Instruments»Measurement & Automation**.
2. Expand the **Devices and Interfaces** section to view the installed National Instruments devices. MAX displays the National Instruments hardware and software in the computer. The device number appears in quotes following the device name. The data acquisition VIs use this device number to determine which device performs DAQ operations.
3. Create a simulated DAQ device for use later in this course. Simulated devices are a powerful tool for development without having hardware physically installed in your computer. Right-click **Devices and Interfaces** and select **Create New…»NI-DAQmx Simulated Device**. Click "Finish".
4. Expand the M Series DAQ section. Select **PCI-6220** or any other PCI device of your choice. Click "OK".
5. The NI-DAQmx Devices folder will expand and you will see a new entry for PCI-6220: "Dev1". You have now created a simulated device!
6. Perform a self-test on the device by right-clicking it in the configuration tree and choosing **Self-Test** or clicking "Self-Test" along the top of the window. This tests the system resources assigned to the device. The device should pass the test because it is already configured.
7. Check the pinout for your device. Right-click the device in the configuration tree and select **Device Pinouts** or click "Device Pinouts" along the top of the center window.
8. Open the test panels. Right-click the device in the configuration tree and select **Test Panels…** or click "Test Panels…" along the top of the center window. The test panels allow you to test the available functionality of your device, analog input/output, digital input/output, and counter input/output without doing any programming.
9. On the **Analog Input** tab of the test panels, change **Mode** to "Continuous". Click "Start" and observe the signal that is plotted. Click "Stop" when you are done.

10. On the **Digital I/O** tab notice that initially the port is configured to be all input. Observe under **Select State** the LEDs that represent the state of the input lines. Click the “All Output” button under **Select Direction**. Notice you now have switches under **Select State** to specify the output state of the different lines. Click “Close” to close the test panels.
11. Close MAX.



Devices and Interfaces

What is Devices and Interfaces?

The Devices and Interfaces category lists installed and detected CAN, DAQ, FieldPoint Serial Controllers, GPIB, IMAQ, IVI, Motion, Serial, VISA, and VXI hardware.

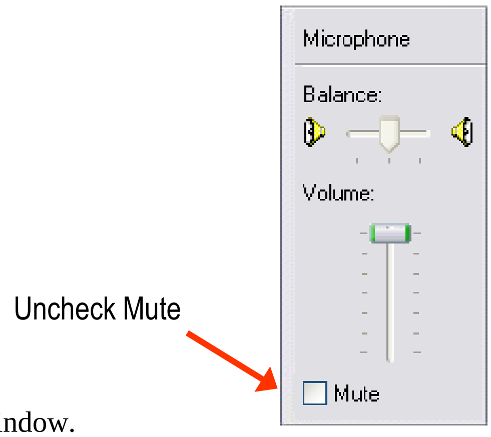
If you do not see your devices...

(End of Exercise)

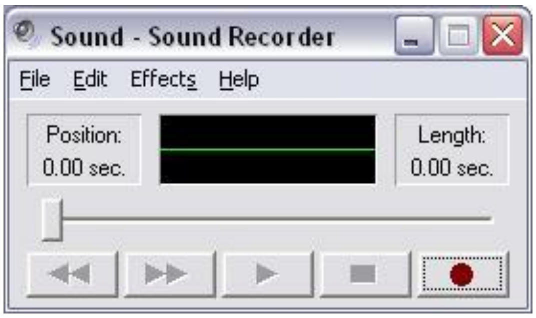
Exercise 1 – Setting Up Your Device (Track C)

In this exercise, you will use Windows utilities to verify your sound card and prepare it for use with a microphone.

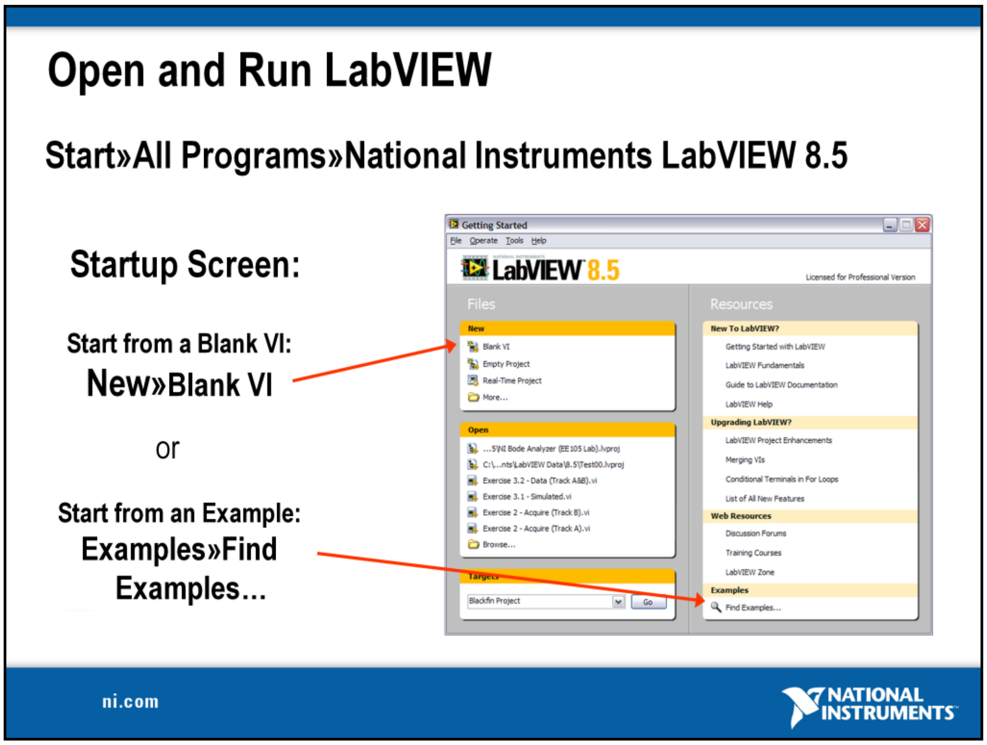
1. Prepare your microphone for use. Double-click the volume control icon on the task bar to open up the configuration window. The sound configuration window can also be found from the Windows Control Panel: **Start Menu»Control Panel»Sounds and Audio Devices»Advanced**.
2. If you do not see a microphone section, go to **Options»Properties»Recording** and place a checkmark in the box next to **Microphone**. This will display the Microphone volume control. Click "OK".
3. Uncheck the **Mute** box if it is not already unchecked. Make sure that the volume is turned up.



1. Close the volume control configuration window.
2. Open the Sound Recorder by selecting **Start»Programs»Accessories»Entertainment»Sound Recorder**.
3. Click the record button and speak into your microphone. Notice how the sound signal is displayed in the Sound Recorder.
4. Click stop and close the Sound Recorder without saving changes when you are finished.



(End of Exercise)



Open and Run LabVIEW

Start»All Programs»National Instruments LabVIEW 8.5

Startup Screen:

Start from a Blank VI:
New»Blank VI

or

Start from an Example:
Examples»Find Examples…

Getting Started
File Operate Tools Help
LabVIEW 8.5
Licensed for Professional Version
Files
New
Blank VI
Empty Project
Real-Time Project
More…
Open
…5\NI Bode Analyzer (EE105 Lab).lvproj
C:\…nts\LabVIEW Data\8.5\Test00.lvproj
Exercise 3.2 - Data (Track A&B).vi
Exercise 3.1 - Simulated.vi
Exercise 2 - Acquire (Track B).vi
Exercise 2 - Acquire (Track A).vi
Browse…
Targets
Blackfin Project
Go
Resources
New To LabVIEW?
Getting Started with LabVIEW
LabVIEW Fundamentals
Guide to LabVIEW Documentation
LabVIEW Help
Upgrading LabVIEW?
LabVIEW Project Enhancements
Merging VIs
Conditional Terminals in For Loops
List of All New Features
Web Resources
Discussion Forums
Training Courses
LabVIEW Zone
Examples
Find Examples…
ni.com
NATIONAL INSTRUMENTS

LabVIEW

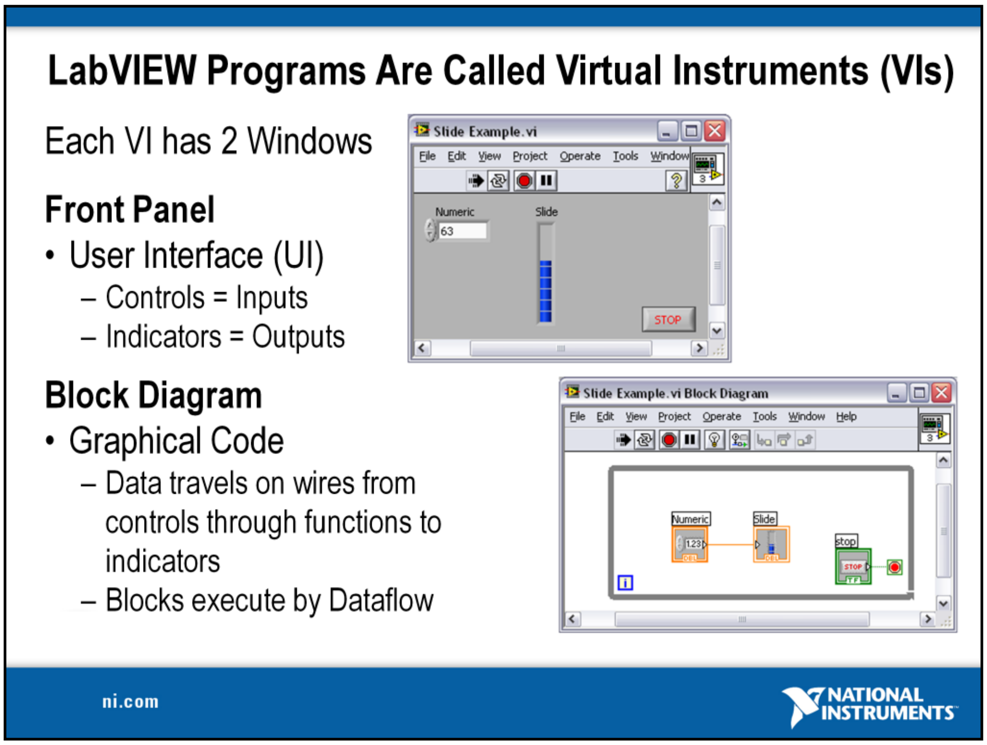
LabVIEW is a graphical programming language that uses icons instead of lines of text to create applications. In contrast to text-based programming languages, where instructions determine program execution, LabVIEW uses dataflow programming, where the flow of data determines execution order.

You can purchase several add-on software toolkits for developing specialized applications. All the toolkits integrate seamlessly in LabVIEW. Refer to the National Instruments Web site for more information about these toolkits.

LabVIEW also includes several wizards to help you quickly configure your DAQ devices and computer-based instruments and build applications.

LabVIEW Example Finder

LabVIEW includes hundreds of example VIs you can use and incorporate into VIs that you create. In addition to the example VIs that ship with LabVIEW, you also can access hundreds of example VIs on the NI Developer Zone (`zone.ni.com`). You can modify an example VI to fit an application, or you can copy and paste from one or more examples into a VI that you create.



LabVIEW Programs Are Called Virtual Instruments (VIs)

Each VI has 2 Windows

Front Panel

- User Interface (UI)
  - Controls = Inputs
  - Indicators = Outputs

Block Diagram

- Graphical Code
  - Data travels on wires from controls through functions to indicators
  - Blocks execute by Dataflow

Slide Example.vi
File Edit View Project Operate Tools Window
Numeric
63
Slide
STOP
Slide Example.vi Block Diagram
File Edit View Project Operate Tools Window Help
Numeric
Slide
stop
ni.com
NATIONAL INSTRUMENTS

LabVIEW programs are called virtual instruments (VIs).

Controls are inputs and indicators are outputs.

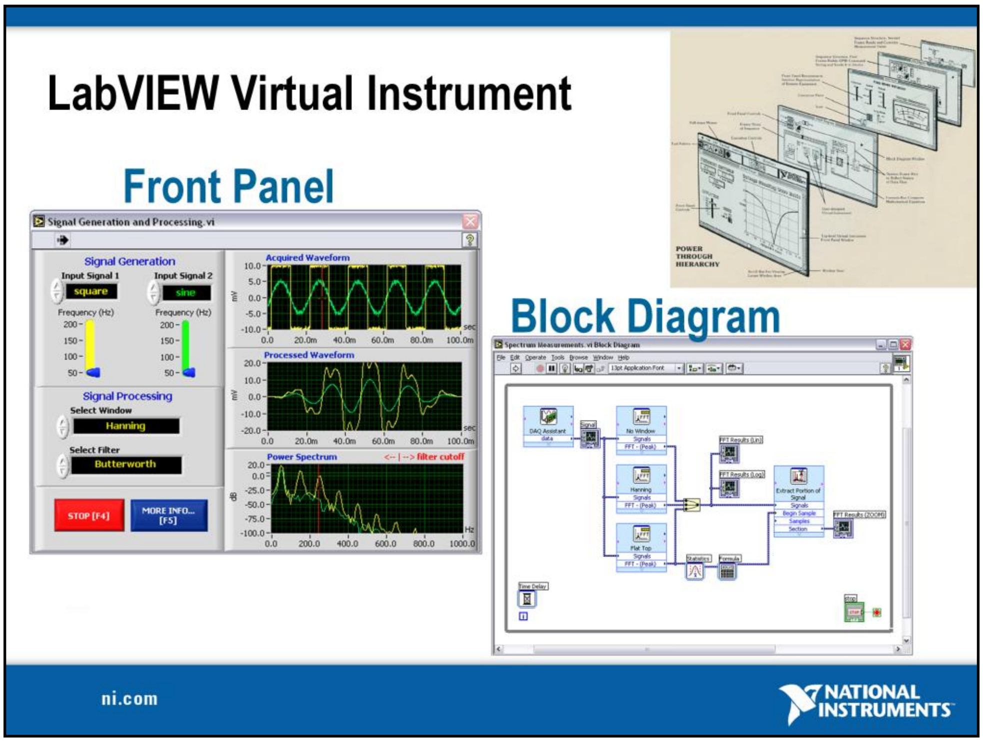
Each VI contains three main parts:

- Front Panel – How the user interacts with the VI.
- Block Diagram – The code that controls the program.
- Icon/Connector – Means of connecting a VI to other VIs.

In LabVIEW, you build a user interface by using a set of tools and objects. The user interface is known as the front panel. You then add code using graphical representations of functions to control the front panel objects. The block diagram contains this code. In some ways, the block diagram resembles a flowchart.

Users interact with the Front Panel when the program is running. Users can control the program, change inputs, and see data updated in real time. Controls are used for inputs such as, adjusting a slide control to set an alarm value, turning a switch on or off, or to stop a program. Indicators are used as outputs. Thermometers, lights, and other indicators display output values from the program. These may include data, program states, and other information.

Every front panel control or indicator has a corresponding terminal on the block diagram. When a VI is run, values from controls flow through the block diagram, where they are used in the functions on the diagram, and the results are passed into other functions or indicators through wires.



LabVIEW Virtual Instrument

Acquire-Analyze on Block Diagram

Present on Front Panel

LabVIEW programs are called virtual instruments (VIs).

Controls are inputs and indicators are outputs.

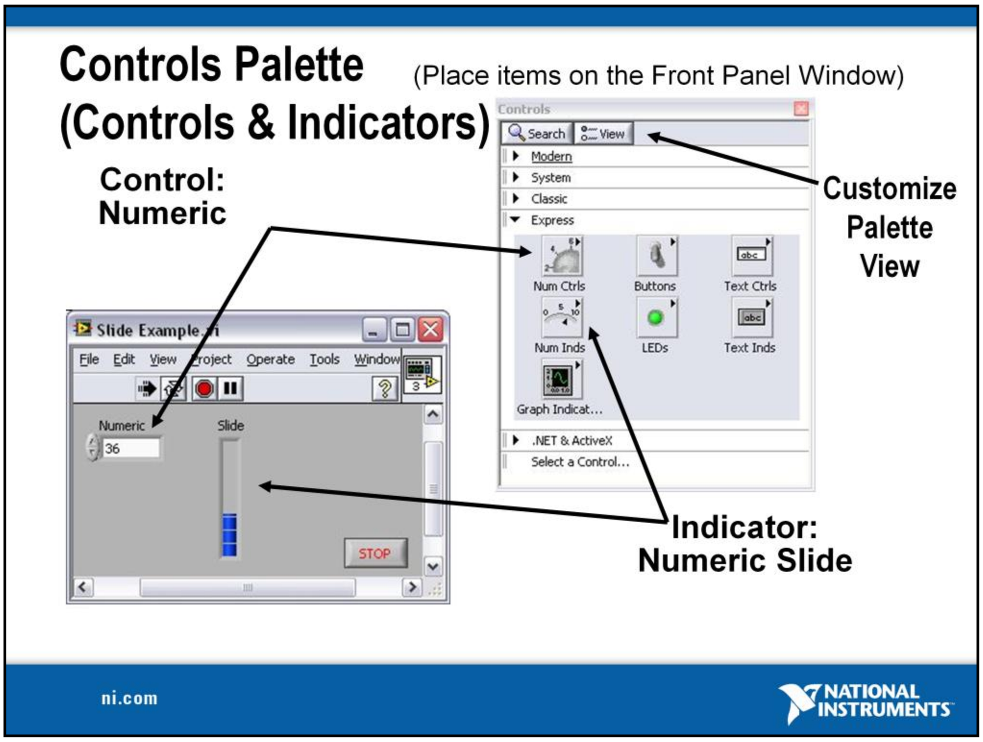
Each VI contains three main parts:

- •Front Panel – How the user interacts with the VI.
- •Block Diagram – The code that controls the program.
- •Icon/Connector – Means of connecting a VI to other VIs.

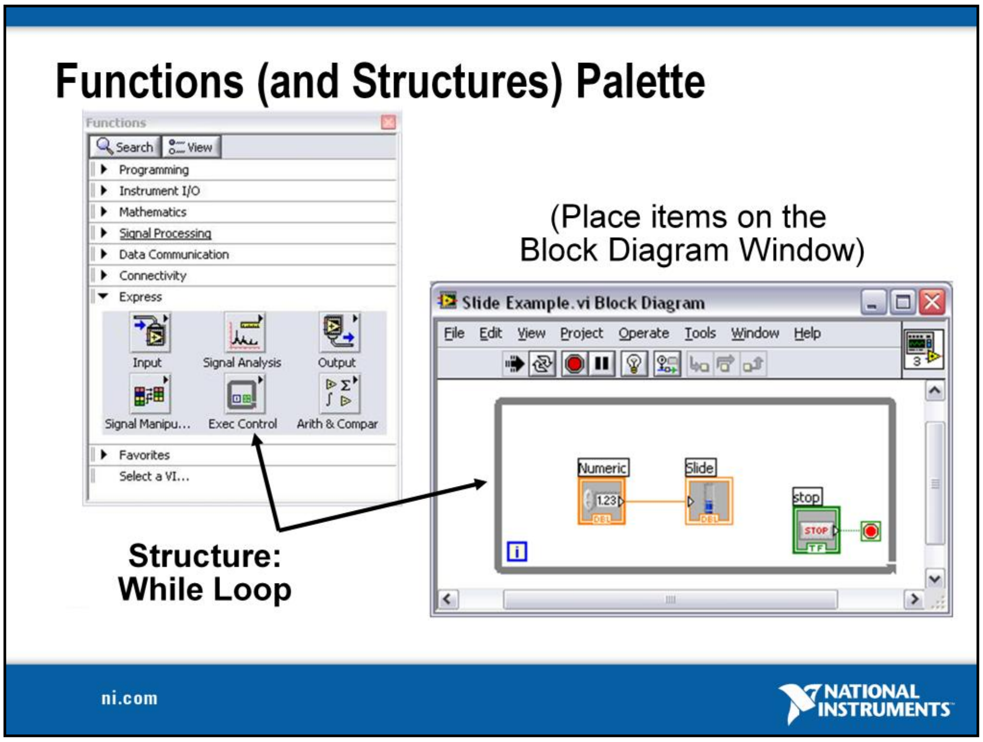
In LabVIEW, you build a user interface by using a set of tools and objects. The user interface is known as the front panel. You then add code using graphical representations of functions to control the front panel objects. The block diagram contains this code. In some ways, the block diagram resembles a flowchart.

Users interact with the Front Panel when the program is running. Users can control the program, change inputs, and see data updated in real time. Controls are used for inputs such as, adjusting a slide control to set an alarm value, turning a switch on or off, or to stop a program. Indicators are used as outputs. Thermometers, lights, and other indicators display output values from the program. These may include data, program states, and other information.

Every front panel control or indicator has a corresponding terminal on the block diagram. When a VI is run, values from controls flow through the block diagram, where they are used in the functions on the diagram, and the results are passed into other functions or indicators through wires.

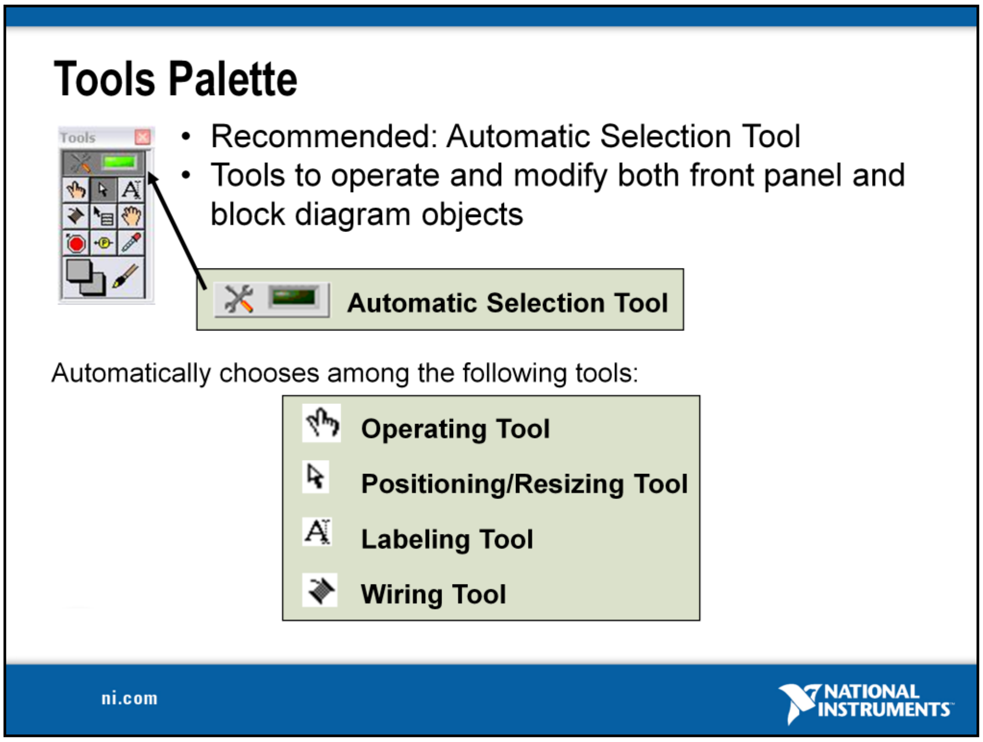


Use the **Controls** palette to place controls and indicators on the front panel. The **Controls** palette is available only on the front panel. To view the palette, select **Window»Show Controls Palette**. You also can display the **Controls** palette by right-clicking an open area on the front panel. Tack down the **Controls** palette by clicking the pushpin on the top left corner of the palette.



Functions (and Structures) Palette

Use the **Functions** palette to build the block diagram. The **Functions** palette is available only on the block diagram. To view the palette, select **Window»Show Functions Palette**. You also can display the **Functions** palette by right-clicking an open area on the block diagram. Tack down the **Functions** palette by clicking the pushpin on the top left corner of the palette.



Tools Palette

- Recommended: Automatic Selection Tool
- Tools to operate and modify both front panel and block diagram objects

Automatically chooses among the following tools:

- Operating Tool
- Positioning/Resizing Tool
- Labeling Tool
- Wiring Tool

If automatic tool selection is enabled and you move the cursor over objects on the front panel or block diagram, LabVIEW automatically selects the corresponding tool from the **Tools** palette. Toggle automatic tool selection by clicking the **Automatic Tool Selection** button in the **Tools** palette.

Use the **Operating tool** to change the values of a control or select the text within a control.

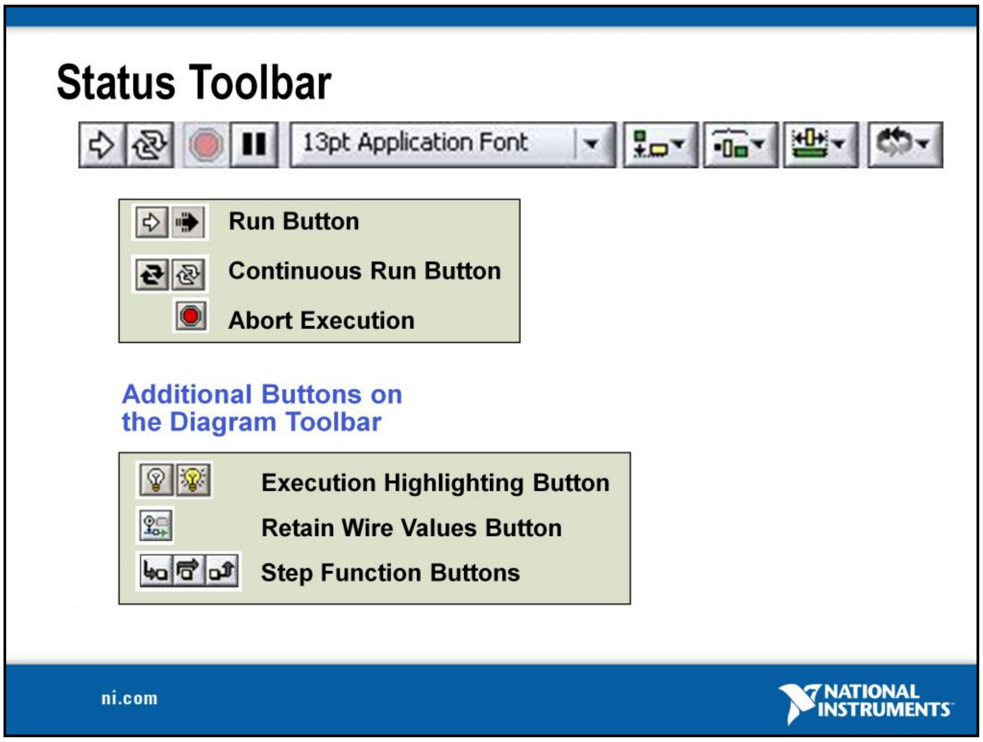
Use the **Positioning tool** to select, move, or resize objects. The Positioning tool changes shape when it moves over a corner of a resizable object.

Use the **Labeling tool** to edit text and create free labels. The Labeling tool changes to a cursor when you create free labels.

Use the **Wiring tool** to wire objects together on the block diagram.

Other important tools:

- Scrolling Tool
- Breakpoint Tool
- Probe Tool
- Color Copy Tool
- Coloring Tool
- Shortcut Menu Tool



Status Toolbar

- Run Button
- Continuous Run Button
- Abort Execution

Additional Buttons on the Diagram Toolbar

- Execution Highlighting Button
- Retain Wire Values Button
- Step Function Buttons

- Click the **Run** button to run the VI. While the VI runs, the **Run** button appears with a black arrow if the VI is a top-level VI, meaning it has no callers and therefore is not a subVI.
- Click the **Continuous Run** button to run the VI until you abort or pause it. You also can click the button again to disable continuous running.
- While the VI runs, the **Abort Execution** button appears. Click this button to stop the VI immediately.

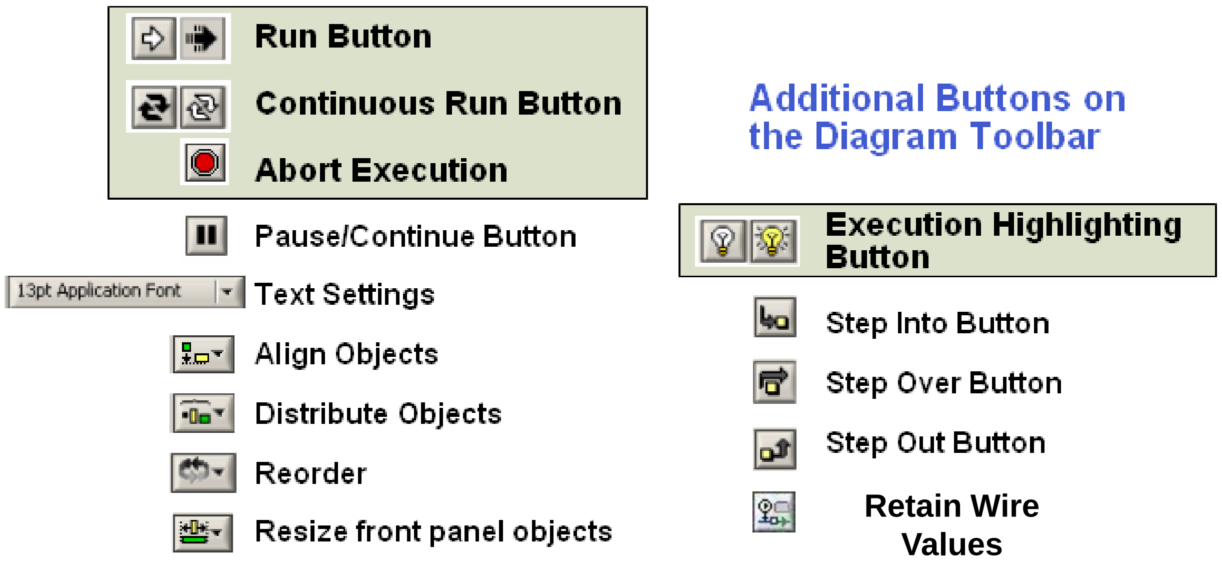
  Note: Avoid using the **Abort Execution** button to stop a VI. Either let the VI complete its data flow or design a method to stop the VI programmatically. By doing so, the VI is at a known state. For example, place a button on the front panel that stops the VI when you click it.
- Click the **Pause** button to pause a running VI. When you click the **Pause** button, LabVIEW highlights on the block diagram the location where you paused execution. Click the **Pause** button again to continue running the VI.
- Select the **Text Settings** pull-down menu to change the font settings for the VI, including size, style, and color.
- Select the **Align Objects** pull-down menu to align objects along axes, including vertical, top edge, left, and so on.
- Select the **Distribute Objects** pull-down menu to space objects evenly, including gaps, compression, and so on.
- Select the **Resize Objects** pull-down menu to change the width and height of front panel objects.

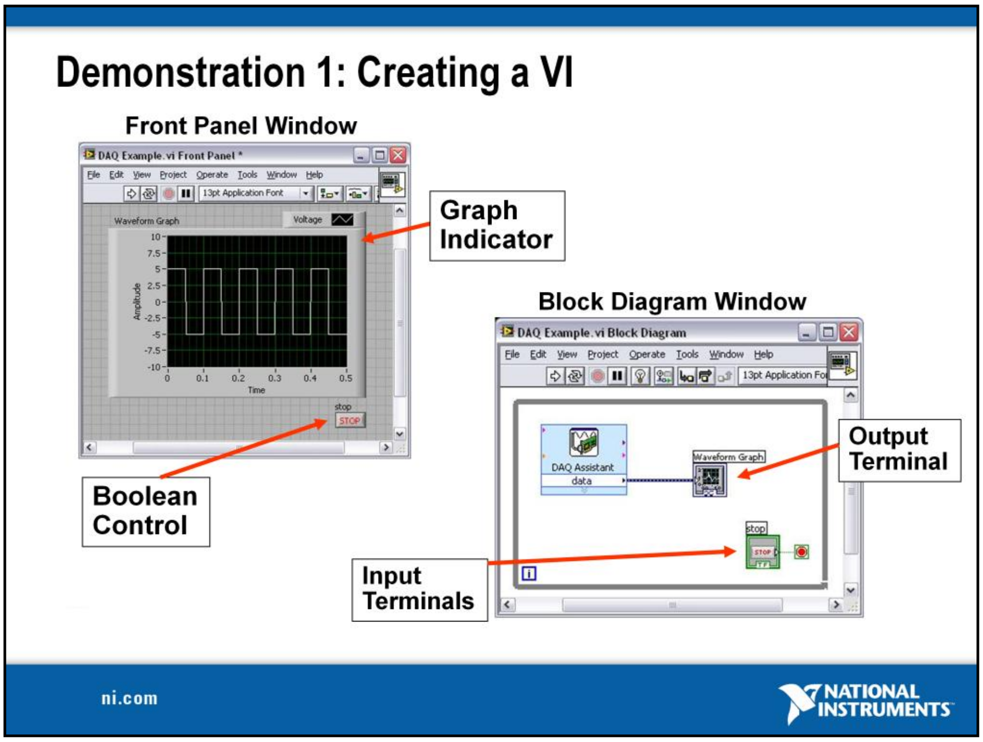
- Select the **Reorder** pull-down menu when you have objects that overlap each other and you want to define which one is in front or back of another. Select one of the objects with the Positioning tool and then select from **Move Forward**, **Move Backward**, **Move To Front**, and **Move To Back**.

Note: The following items only appear on the block diagram toolbar.

- Click the **Highlight Execution** button to see the flow of data through the block diagram. Click the button again to disable execution highlighting.
- Click **Retain Wire Values** button to save the wire values at each point in the flow of execution so that when you place a probe on a wire, you can immediately obtain the most recent value of the data that passed through the wire.
- Click the **Step Into** button to single-step into a loop, subVI, and so on. Single-stepping through a VI steps through the VI node to node. Each node blinks to denote when it is ready to execute. By stepping into the node, you are ready to single-step inside the node.
- Click the **Step Over** button to step over a loop, subVI, and so on. By stepping over the node, you execute the node without single-stepping through the node.
- Click the **Step Out** button to step out of a loop, subVI, and so on. By stepping out of a node, you complete single-stepping through the node and go to the next node.

Additional Tools:





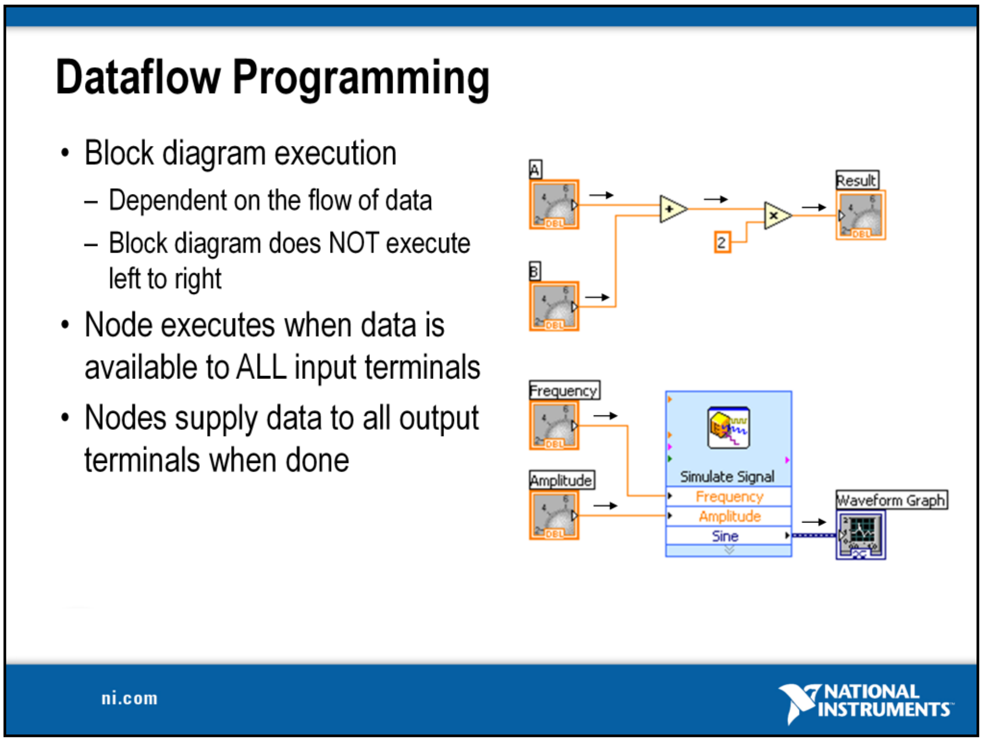
Demonstration 1: Creating a VI

When you create an object on the Front Panel, a terminal will be created on the Block Diagram. These terminals give you access to the Front Panel objects from the Block Diagram code.

Each terminal contains useful information about the Front Panel object it corresponds to. For example, the color and symbols provide information about the data type. For example: The dynamic data type is a polymorphic data type represented by dark blue terminals. Boolean terminals are green with TF lettering.

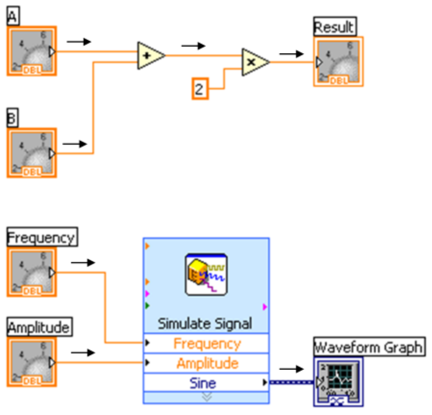
In general, blue terminals should wire to blue terminals, green to green, and so on. This is not a hard-and-fast rule; LabVIEW will allow a user to connect a blue terminal (dynamic data) to an orange terminal (fractional value), for example. But in most cases, look for a match in colors.

Controls have an arrow on the right side and have a thick border. Indicators have an arrow on the left and a thin border. Logic rules apply to wiring in LabVIEW: Each wire must have one (but only one) source (or control), and each wire may have multiple destinations (or indicators).



Dataflow Programming

- Block diagram execution
  - Dependent on the flow of data
  - Block diagram does NOT execute left to right
- Node executes when data is available to ALL input terminals
- Nodes supply data to all output terminals when done



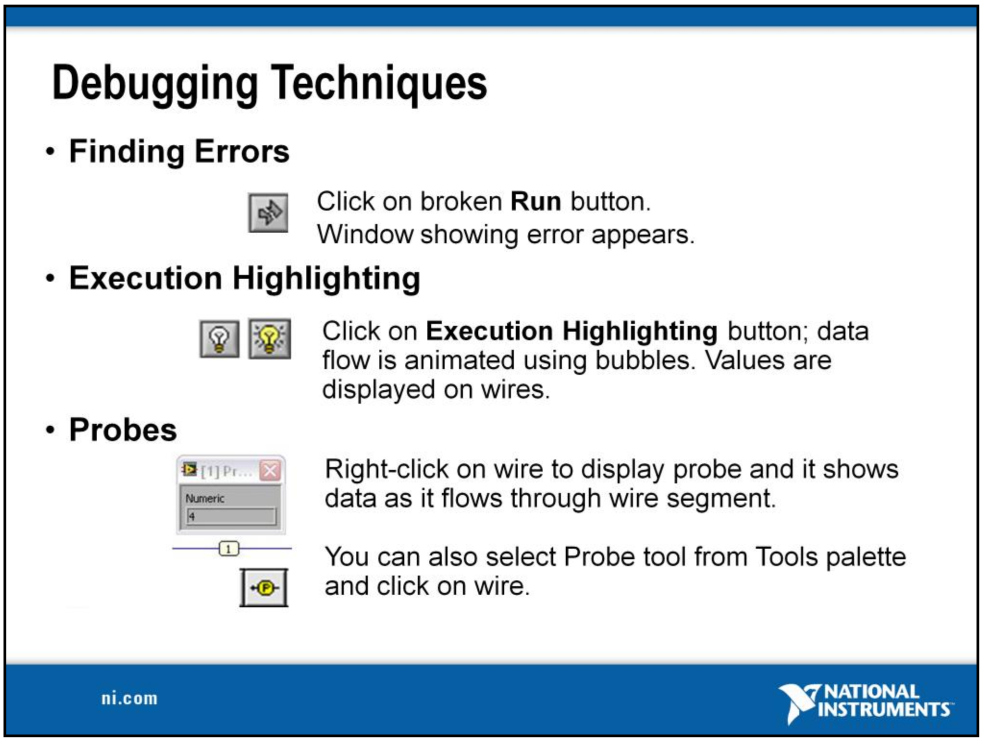
ni.com

NATIONAL INSTRUMENTS

LabVIEW follows a dataflow model for running VIs. A block diagram node executes when all its inputs are available. When a node completes execution, it supplies data to its output terminals and passes the output data to the next node in the dataflow path. Visual Basic, C++, JAVA, and most other text-based programming languages follow a control flow model of program execution. In control flow, the sequential order of program elements determines the execution order of a program.

Consider the block diagram above. It adds two numbers and then multiplies by 2 from the result of the addition. In this case, the block diagram executes from left to right, not because the objects are placed in that order, but because one of the inputs of the Multiply function is not valid until the Add function has finished executing and passed the data to the Multiply function. Remember that a node executes only when data are available at all of its input terminals, and it supplies data to its output terminals only when it finishes execution. In the second piece of code, the Simulate Signal Express VI receives input from the controls and passes its result to the Graph.

You may consider the add-multiply and the simulate signal code to co-exist on the same block diagram in parallel. This means that they will both begin executing at the same time and run independent of one another. If the computer running this code had multiple processors, these two pieces of code could run independent of one another (each on its own processor) without any additional coding.



Debugging Techniques

- **Finding Errors**

  Click on broken **Run** button.
  Window showing error appears.

- **Execution Highlighting**

  Click on **Execution Highlighting** button; data flow is animated using bubbles. Values are displayed on wires.

- **Probes**

  Right-click on wire to display probe and it shows data as it flows through wire segment.

  You can also select Probe tool from Tools palette and click on wire.

ni.com

When your VI is not executable, a broken arrow is displayed in the Run button in the palette.

- **Finding Errors**: To list errors, click on the broken arrow. To locate the bad object, click on the error message.
- **Execution Highlighting**: Animates the diagram and traces the flow of the data, allowing you to view intermediate values. Click on the **light bulb** on the toolbar.
- **Probe**: Used to view values in arrays and clusters. Click on wires with the **Probe** tool or right-click on the wire to set probes.
- **Retain Wire Values**: Used in conjunction with probes to view the values from the last iteration of the program.
- **Breakpoint**: Set pauses at different locations on the diagram. Click on wires or objects with the **Breakpoint** tool to set breakpoints.

Exercise 2 – Acquiring a Signal with DAQ (Track A)

Note: Before beginning this exercise, copy the Exercises and Solutions Folders to the desktop of your computer.

Complete the following steps to create a VI that acquires data continuously from your DAQ device.

1. Launch LabVIEW.
2. In the **Getting Started** window, click the **New** or **VI from Template** link to display the **New** dialog box.
3. Open a data acquisition template. From the Create New list, select **VI**»F**rom Templat**e»DAQ»D**ata Acquisition with NI-DAQmx.vi** and click "OK".
4. Display the block diagram by clicking it or by selecting **Window»Show Block Diagram**. Read the instructions written there about how to complete the program.
5. Double-click the DAQ Assistant to launch the configuration wizard.
6. Configure an analog input operation.
    a. Choose **Analog Input»Voltage**.
    b. Choose **Dev1 (USB-6009)»ai0** to acquire data on analog input channel 0 and click "Finish."
    c. In the next window you define parameters of your analog input operation. To choose an input range that works well with your microphone, on the settings tab enter **2 Volts** for the maximum and **–2 Volts** for the minimum. On the task timing tab, choose "**Continuous**" for the acquisition mode and enter **10000** for the rate. Leave all other choices set to their default values. Click "OK" to exit the wizard.
7. Place the Filter Express VI to the right of the DAQ Assistant on the block diagram. From the functions palette, select **Express»Signal Analysis**»**Filter** and place it on the block diagram inside the while loop. When you bring up the functions palette, press the small push pin in the upper left hand corner of the palette. This will tack down the palette so that it doesn't disappear. This step will be omitted in the following exercises, but should be repeated. In the configuration window under Filtering Type, choose "Highpass." Under Cutoff Frequency, use a value of 300 Hz. Click "OK."

8. Make the following connections on the block diagram by hovering your mouse over the terminal so that it becomes the wiring tool and clicking once on each of the terminals you wish to connect:

   a. Connect the “Data” output terminal of the DAQ Assistant VI to the “Signal” input of the Filter VI.

   b. Create a graph indicator for the filtered signal by right-clicking on the “Filtered Signal” output terminal and choose **Create»Graph Indicator.**

9. Return to the front panel by selecting **Wind**ow»Sh**ow Front Panel** or by pressing <Ctrl+E>.

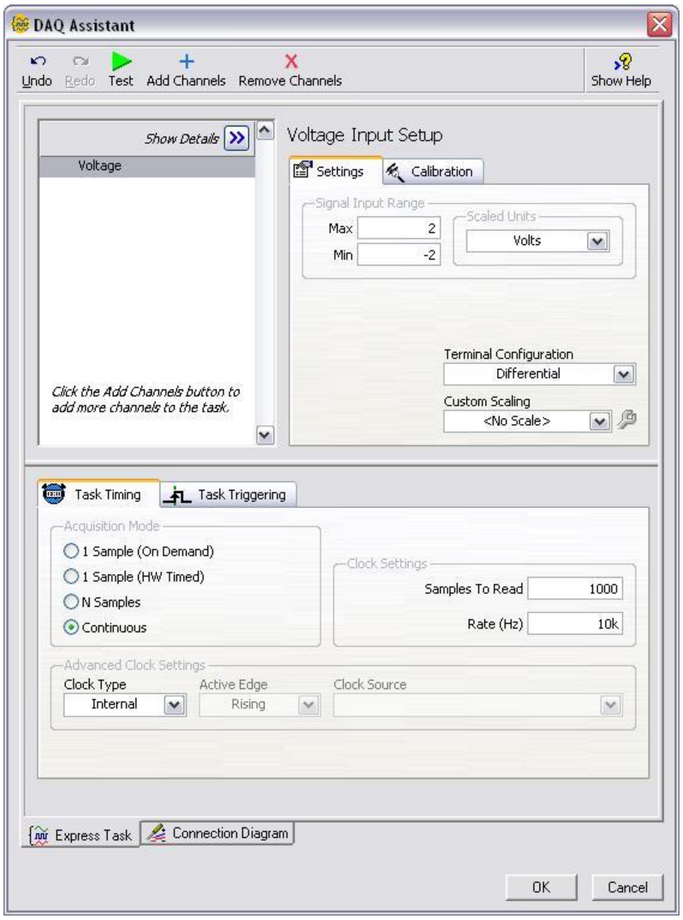
10. Run your program by clicking the run button. Hum or whistle into the microphone to observe the changing voltage on the graph.

11. Click stop once you are finished.

12. Save the VI as “Exercise 2 – Acquire.vi” in your Exercises folder and close it.

Note: The solution to this exercise is printed in the back of this manual.

Tip: You can place the DAQ Assistant on your block diagram from the Functions Palette. Right-click the block diagram to open the Functions Palette and go to **Express»Input** to find it.



Voltage

Click the Add Channels button to add more channels to the task.

Voltage Input Setup

Signal Input Range
Max 2
Min -2

Scaled Units: Volts

Terminal Configuration: Differential

Custom Scaling: <No Scale>

Acquisition Mode
1 Sample (On Demand)
1 Sample (HW Timed)
N Samples
[✓ Continuous]

Clock Settings
Samples To Read 1000
Rate (Hz) 10k

Advanced Clock Settings
Clock Type: Internal
Active Edge: Rising
Clock Source:

(End of Exercise)

Exercise 2 – Acquiring a Signal with DAQ (Track B)

Note: Before beginning this exercise, copy the Exercises and Solutions Folders to the desktop
of your computer.

Complete the following steps to create a VI that acquires data continuously from your simulated DAQ device.

1. Launch LabVIEW.
2. In the **Getting Started** window, click the **New** or **VI from Template** link to display the **New** dialog box.
3. Open a data acquisition template. From the Create New list, select **VI»From Template»DAQ»Data Acquisition with NI-DAQmx.vi** and click "OK".
4. Display the block diagram by clicking it or by selecting **Window»Show Block Diagram**. Read the instructions written there about how to complete the program.
5. Double-click the DAQ Assistant to launch the configuration wizard.
6. Configure an analog input operation.

   a. Choose **Analog Input»Voltage**.

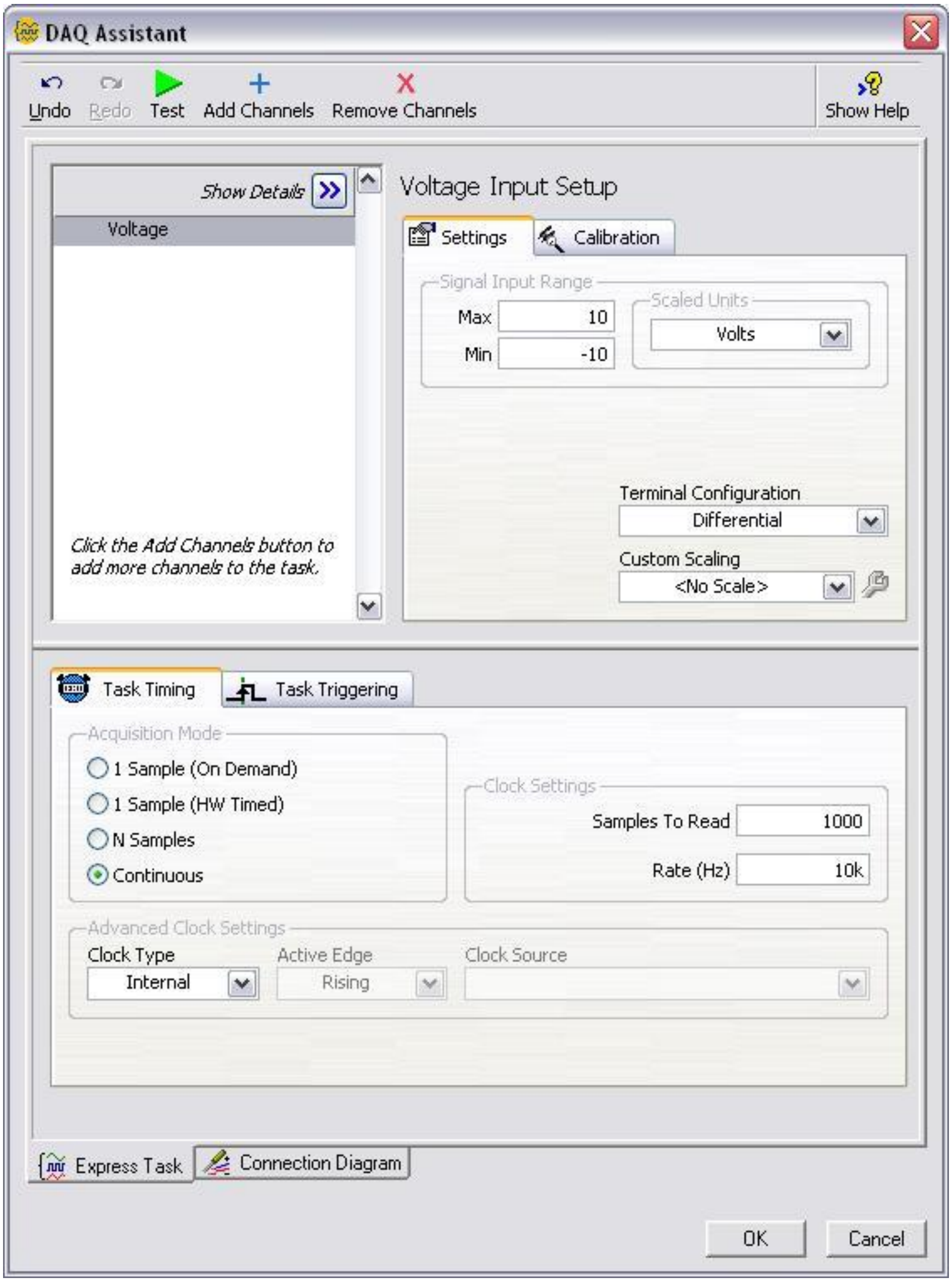
   b. Choose **Dev1 (PCI-6220)»ai0** to acquire data on analog input channel 0 and click "Finish."

   c. In the next window you define parameters of your analog input operation.

   On the task timing tab, choose "**Continuous**" for the acquisition mode, enter **1000** for samples to read, and **10000** for the rate. Leave all other choices set to their default values. Click "OK" to exit the wizard.
7. On the block diagram, right-click the black arrow to the right of where it says "data." Choose **Create**»**Graph Indicator** from the right-click menu.
8. Return to the front panel by selecting **Window»Show Front Panel** or by pressing <Ctrl+E>.
9. Run your program by clicking the run button. Observe the simulated sine wave on the graph.
10. Click stop once you are finished.
11. Save the VI as "Exercise 2 – Acquire.vi" in the Exercises folder. Close the VI.

Notes:

- The solution to this exercise is printed in the back of this manual.
- You can place the DAQ Assistant on your block diagram from the Functions Palette. Right-click the block diagram to open the Functions Palette and go to **Express»Input** to find it. When you bring up the functions palette, press the small push pin in the upper left hand corner of the palette. This will tack down the palette so that it doesn't disappear. This step will be omitted in the following exercises, but should be repeated.



Signal Input Range
Max: 10
Min: -10

Scaled Units: Volts

Terminal Configuration: Differential

Custom Scaling: <No Scale>

Acquisition Mode
○ 1 Sample (On Demand)
○ 1 Sample (HW Timed)
○ N Samples
[✓ Continuous]

Clock Settings
Samples To Read: 1000
Rate (Hz): 10k

Advanced Clock Settings
Clock Type: Internal
Active Edge: Rising
Clock Source:

(End of Exercise)

Exercise 2 – Acquiring a Signal with the Sound Card (Track C)

Note: Before beginning this exercise, copy the Exercises and Solutions Folders to the desktop of your computer.

Complete the following steps to create a VI that acquires data from your sound card.

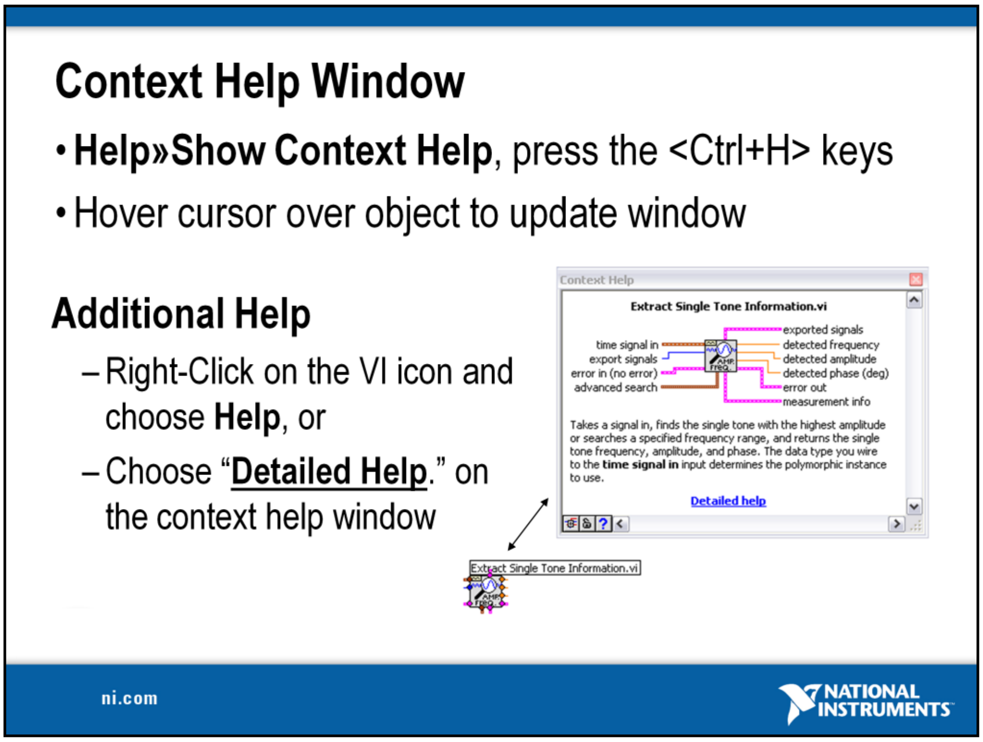
1. Launch LabVIEW.
2. In the **Getting Started** window, click the **Blank VI** link.
3. Display the block diagram by pressing <Ctrl+E> or selecting **Window»Show Block Diagram**.
4. Place the Acquire Sound Express VI on the block diagram. Right-click to open the functions palette and select **Express»Input»Acquire Sound.** Place the Express VI on the block diagram.
5. In the configuration window under **#Channels**, select **1** from the drop-down list and click "OK".
6. Place the Filter Express VI to the right of the Acquire Signal VI on the block diagram. From the functions palette, select **Express»Signal Analysis»Filter** and place it on the block diagram. In the configuration window under **Filtering Type**, choose "**Highpass**." Under Cutoff Frequency, use a value of 300 Hz. Click "OK."
7. Make the following connections on the block diagram by hovering your mouse over the terminal so that it becomes the wiring tool and clicking once on each of the terminals you wish to connect:

   a. Connect the "Data" output terminal of the Acquire Signal VI to the "Signal" input of the Filter VI.

   b. Create a graph indicator for the filtered signal by right-clicking on the "Filtered Signal" output terminal and choose **Create»Graph Indicator.**
8. Return to the front panel by pressing <Ctrl+E> or **Window»Show Front Panel**.
9. Run your program by clicking the run button. Hum or whistle into your microphone and observe the data you acquire from your sound card.
10. Save the VI as "Exercise 2 – Acquire.vi" in the Exercises folder.
11. Close the VI.

Note: The solution to this exercise is printed in the back of this manual.

(End of Exercise)



Context Help Window

- **Help»Show Context Help**, press the <Ctrl+H> keys
- Hover cursor over object to update window

Additional Help

– Right-Click on the VI icon and choose **Help**, or

– Choose "**Detailed Help**." on the context help window

Context Help

Extract Single Tone Information.vi

Takes a signal in, finds the single tone with the highest amplitude or searches a specified frequency range, and returns the single tone frequency, amplitude, and phase. The data type you wire to the **time signal in** input determines the polymorphic instance to use.

Detailed help

ni.com

The **Context Help window** displays basic information about LabVIEW objects when you move the cursor over each object. Objects with context help information include VIs, functions, constants, structures, palettes, properties, methods, events, and dialog box components.

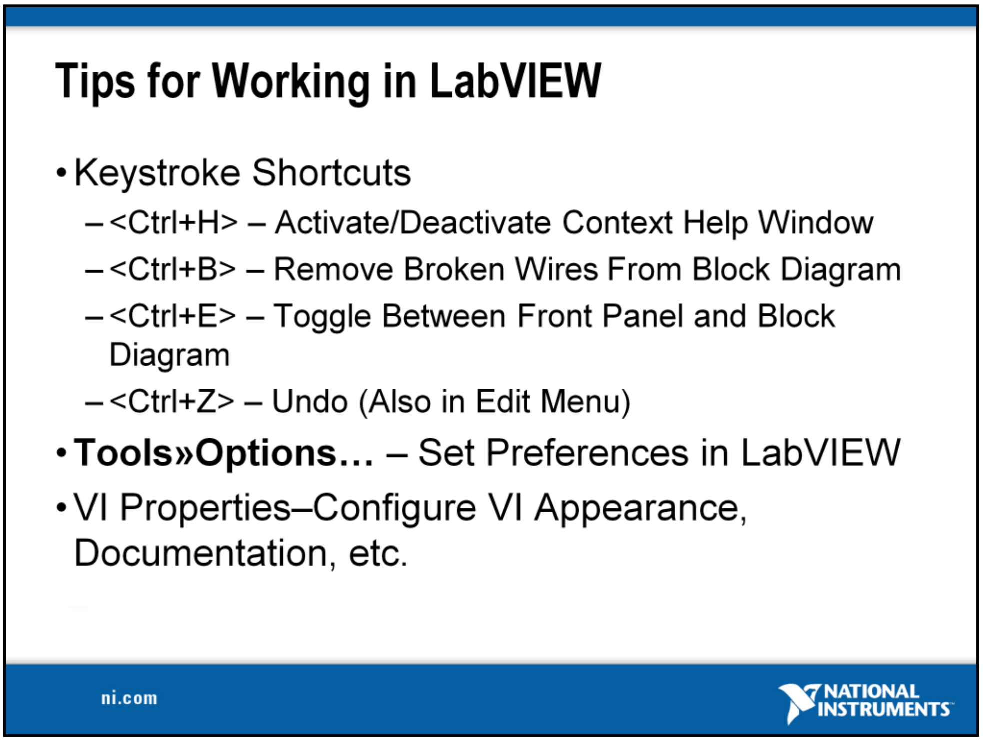
To display the Context Help window, select **Help»Show Context Help**, press the <Ctrl+H> keys, or press the **Show Context Help Window** button in the toolbar

Connections displayed in Context Help:

Required – bold
Recommended – normal
Optional – dimmed

Additional Help

- **VI, Function, & How-To Help is also available.**
  - **Help» VI, Function, & How-To Help**
  - Right-click the VI icon and choose **Help**, or
  - Choose "**Detailed Help**." on the context help window.
- **LabVIEW Help – reference style help**
  - **Help»Search the LabVIEW Help…**



Tips for Working in LabVIEW

- Keystroke Shortcuts
  - <Ctrl+H> – Activate/Deactivate Context Help Window
  - <Ctrl+B> – Remove Broken Wires From Block Diagram
  - <Ctrl+E> – Toggle Between Front Panel and Block Diagram
  - <Ctrl+Z> – Undo (Also in Edit Menu)
- **Tools»Options…** – Set Preferences in LabVIEW
- VI Properties–Configure VI Appearance, Documentation, etc.

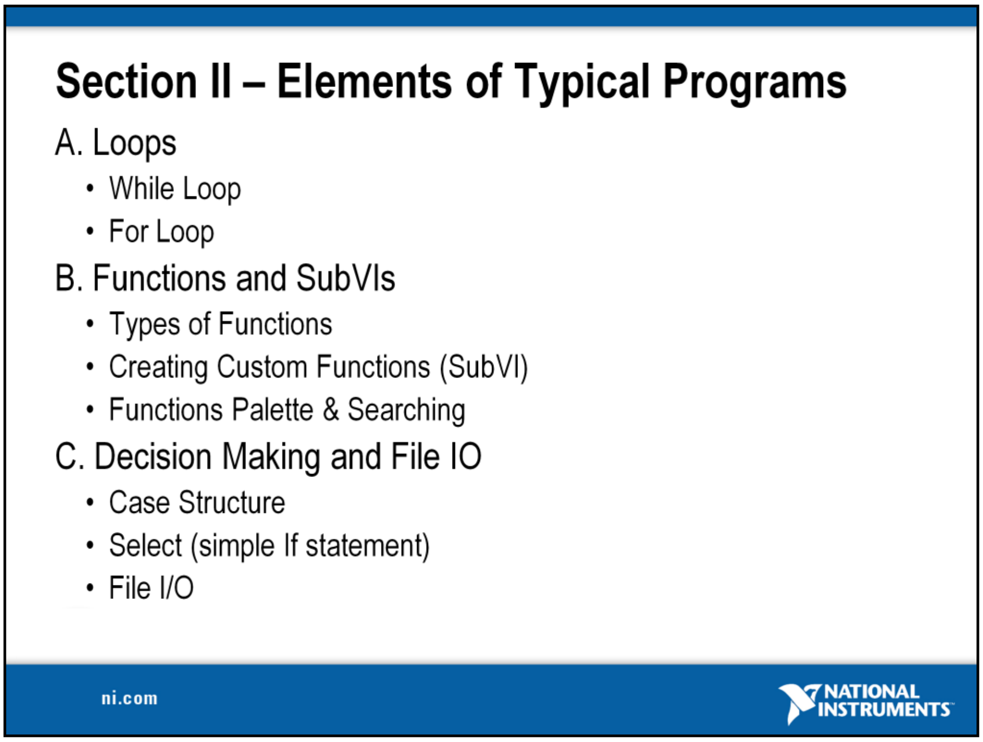
ni.com

LabVIEW has many keystroke shortcuts that make working easier. The most common shortcuts are listed above.

While the Automatic Selection Tool is great for choosing the tool you would like to use in LabVIEW, there are sometimes cases when you want manual control. Once the Automatic Selection Tool is turned off, use the Tab key to toggle between the four most common tools (Operate Value, Position/Size/Select, Edit Text, Set Color on Front Panel and Operate Value, Position/Size/Select, Edit Text, Connect Wire on Block Diagram). Once you are finished with the tool you choose, you can press <Shift+Tab> to turn the Automatic Selection Tool back on.

In the **Tools»Options…** dialog, there are many configurable options for customizing your Front Panel, Block Diagram, Colors, Printing, and much more.

Similar to the LabVIEW Options, you can configure VI specific properties by going to **File»VI Properties…** There you can document the VI, change the appearance of the window, and customize it in several other ways.



Section II – Elements of Typical Programs

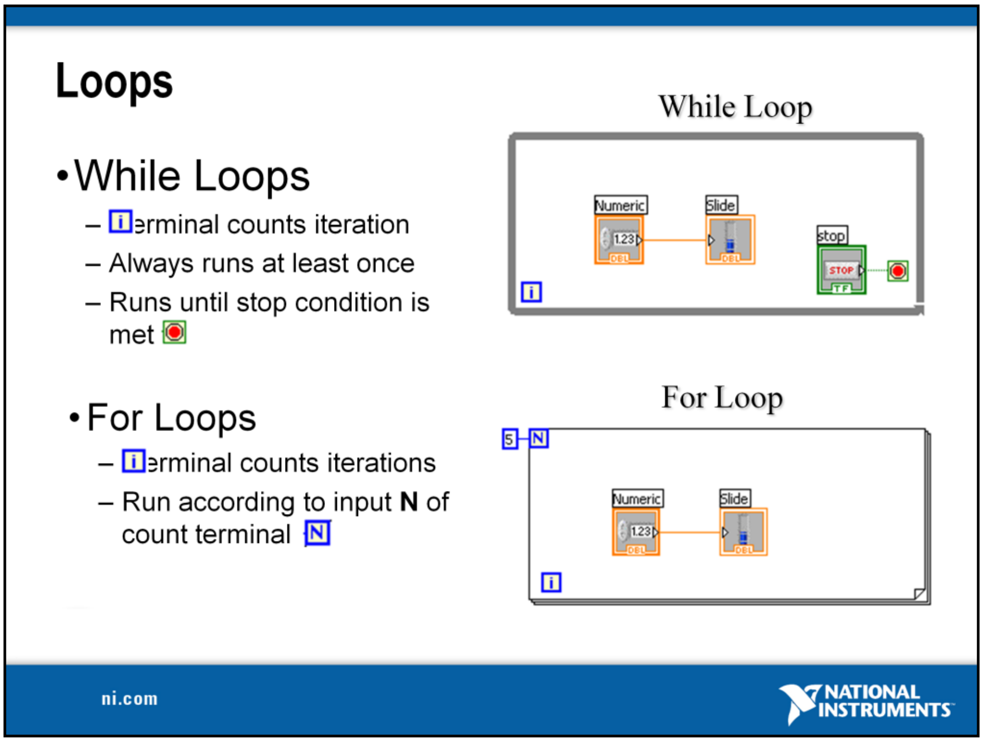
A. Loops
- While Loop
- For Loop

B. Functions and SubVIs
- Types of Functions
- Creating Custom Functions (SubVI)
- Functions Palette & Searching

C. Decision Making and File IO
- Case Structure
- Select (simple If statement)
- File I/O

ni.com

NATIONAL INSTRUMENTS



Loops

- While Loops
  - i terminal counts iteration
  - Always runs at least once
  - Runs until stop condition is met
- For Loops
  - i terminal counts iterations
  - Run according to input **N** of count terminal N

While Loop
Numeric
1.23
DBL
Slide
DBL
stop
STOP
TF
i
For Loop
5
N
Numeric
1.23
DBL
Slide
DBL
i
ni.com
NATIONAL INSTRUMENTS

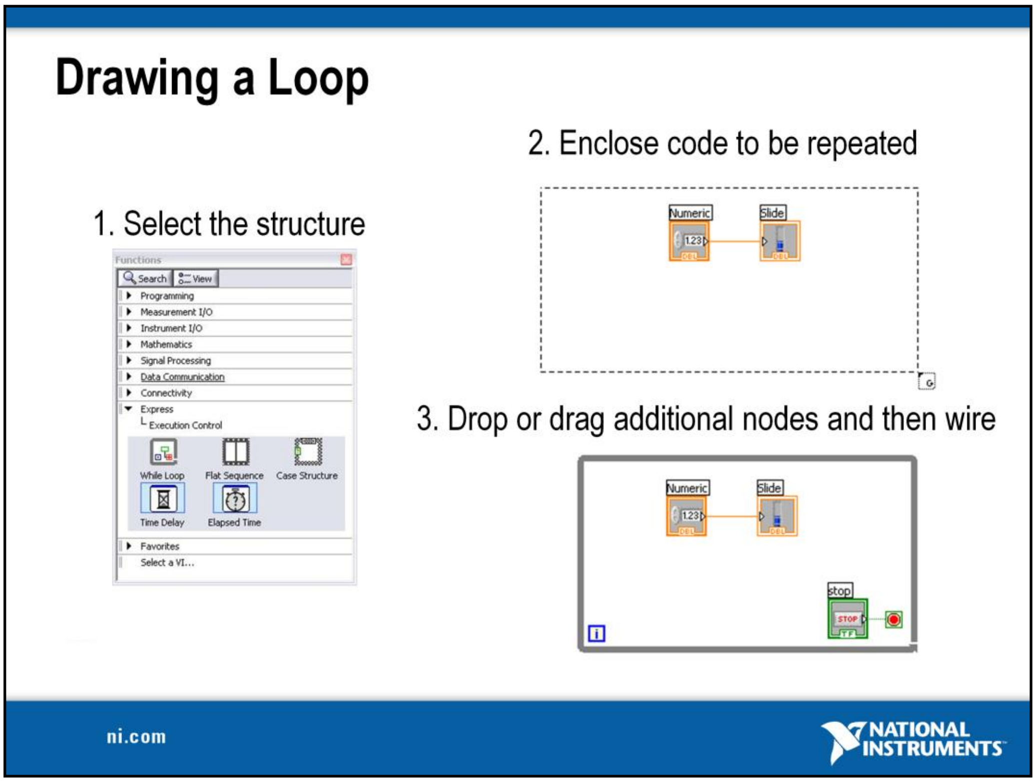
Both the While and For Loops are located on the **Functions»Structures** palette. The For Loop differs from the While Loop in that the For Loop executes a set number of times. A While Loop stops executing the subdiagram only if the value at the conditional terminal exists.

While Loops

Similar to a Do Loop or a Repeat-Until Loop in text-based programming languages, a While Loop, shown at the top right, executes a subdiagram until a condition is met. The While Loop executes the sub diagram until the conditional terminal, an input terminal, receives a specific Boolean value. The default behavior and appearance of the conditional terminal is **Stop If True**. When a conditional terminal is **Stop If True**, the While Loop executes its subdiagram until the conditional terminal receives a TRUE value. The iteration terminal (an output terminal), shown at left, contains the number of completed iterations. The iteration count always starts at zero. During the first iteration, the iteration terminal returns 0.

For Loops

A For Loop, shown above, executes a subdiagram a set number of times. The value in the count terminal (an input terminal) represented by the N, indicates how many times to repeat the subdiagram. The iteration terminal (an output terminal), shown at left, contains the number of completed iterations. The iteration count always starts at zero. During the first iteration, the iteration terminal returns 0.

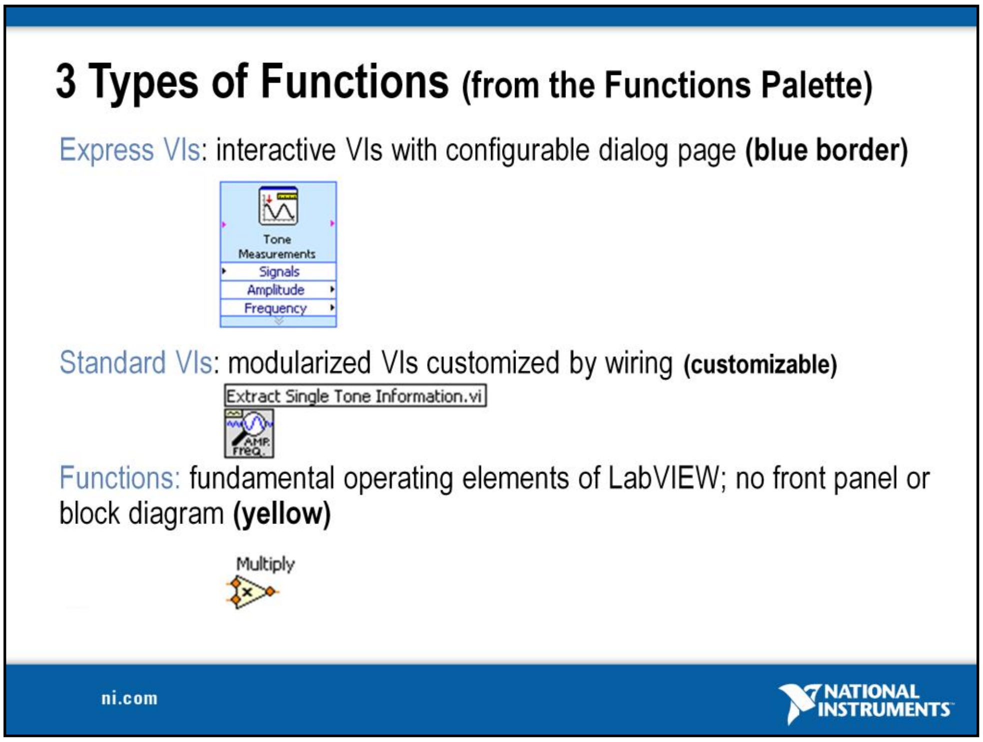


Drawing a Loop

1. Select the structure
2. Enclose code to be repeated
3. Drop or drag additional nodes and then wire

Place loops in your diagram by selecting them from the Structures palette of the Functions palette:

- When selected, the mouse cursor becomes a special pointer that you use to enclose the section of code you want to repeat.
- Click the mouse button to define the top-left corner, click the mouse button again at the bottom-right corner, and the While Loop boundary is created around the selected code.
- Drag or drop additional nodes in the While Loop if needed.



3 Types of Functions (from the Functions Palette)

Express VIs: interactive VIs with configurable dialog page **(blue border)**

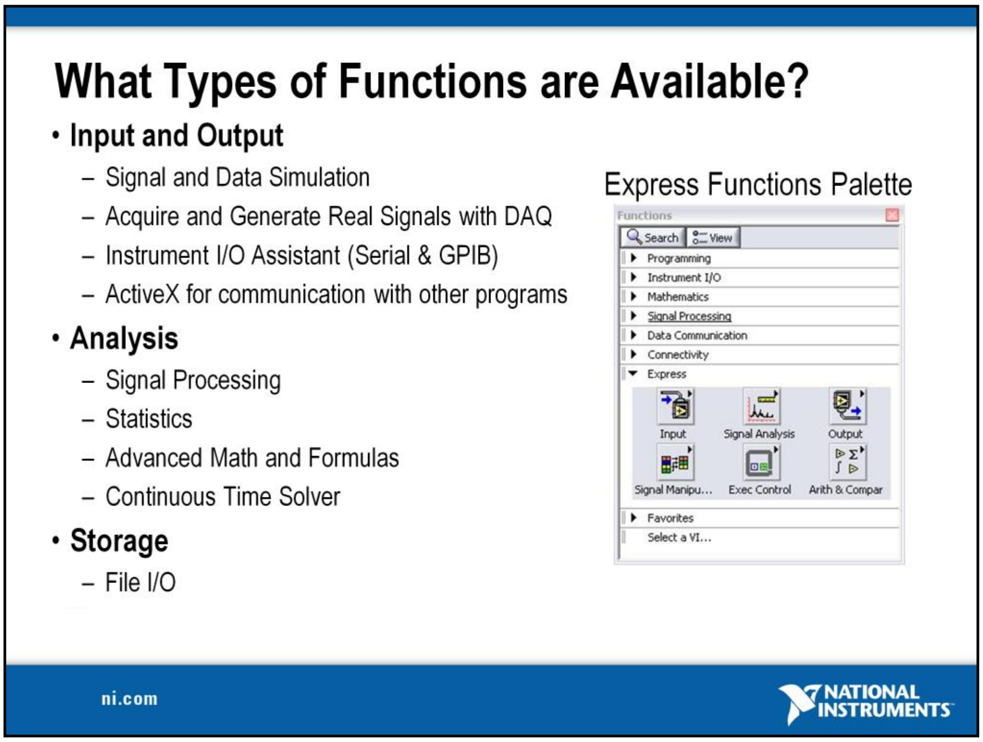
Standard VIs: modularized VIs customized by wiring **(customizable)**

Functions: fundamental operating elements of LabVIEW; no front panel or block diagram **(yellow)**

LabVIEW 7.0 introduced a new type of subVI called Express VIs. These are interactive VIs that have a configuration dialog box that allows the user to customize the functionality of the Express VI. LabVIEW then generates a subVI based on these settings.

SubVIs are VIs (consisting of a front panel and a block diagram) that are used within another VI.

Functions are the building blocks of all VIs. Functions do not have a front panel or a block diagram.



What Types of Functions are Available?

- Input and Output
  - Signal and Data Simulation
  - Acquire and Generate Real Signals with DAQ
  - Instrument I/O Assistant (Serial & GPIB)
  - ActiveX for communication with other programs
- Analysis
  - Signal Processing
  - Statistics
  - Advanced Math and Formulas
  - Continuous Time Solver
- Storage
  - File I/O

Express Functions Palette

Functions
Search
View
Programming
Instrument I/O
Mathematics
Signal Processing
Data Communication
Connectivity
Express
Input
Signal Analysis
Output
Signal Manipu...
Exec Control
Arith & Compar
Favorites
Select a VI...

ni.com

NATIONAL INSTRUMENTS

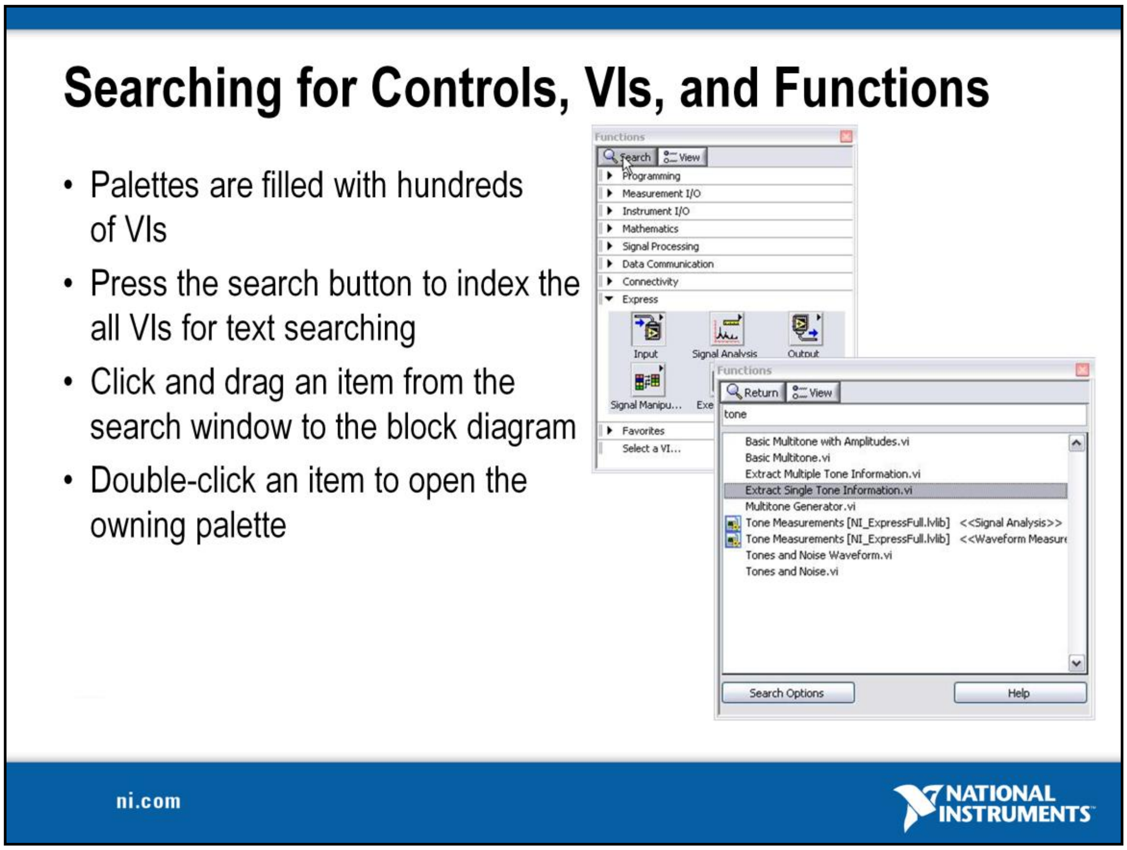
LabVIEW includes several hundreds of pre-built functions that help you to acquire, analyze, and present data. You would generally use these functions as outlined on the slide above.

LabVIEW Toolkits

Additional toolkits are available for adding domain specific functionality to LabVIEW. These toolkits include:

Application Deployment and Targeting Modules	**Signal Processing and Analysis**	**Control Design and Simulation**
* LabVIEW PDA Module * LabVIEW Real-Time Module * LabVIEW FPGA Module * LabVIEW Vision Development Module	* Sound and Vibration Toolkit * Advanced Signal Processing Toolkit * Modulation Toolkit * Spectral Measurements Toolkit * Order Analysis Toolkit * Digital Filter Design Toolkit	* Control Design and Simulation Bundle * LabVIEW Real-Time Module * System Identification Toolkit * Control Design Toolkit * LabVIEW Simulation Module * State Diagram Toolkit
Embedded System Deployment	**Software Engineering and Optimization Tools**	**Image Processing and Acquisition**
* DSP Test Integration Toolkit * Embedded Test Integration Toolkit * Digital Filter Design Toolkit * LabVIEW FPGA Module	* Execution Trace Toolkit for LabVIEW Real-Time * Express VI Development Toolkit * State Diagram Toolkit * VI Analyzer Toolkit	* LabVIEW Vision Development Module * NI Vision Builder for Automated Inspection * NI-IMAQ for IEEE 1394

http://www.ni.com/toolkits/



Searching for Controls, VIs, and Functions

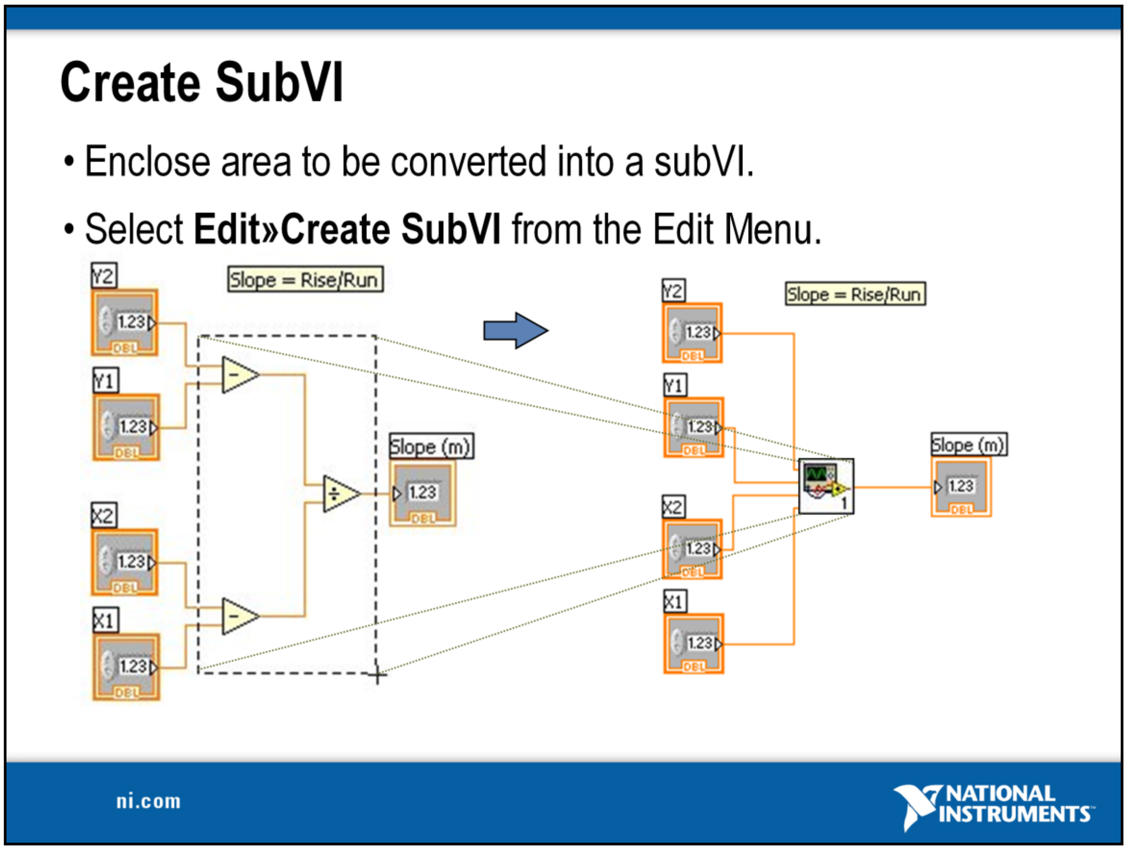
- Palettes are filled with hundreds of VIs
- Press the search button to index the all VIs for text searching
- Click and drag an item from the search window to the block diagram
- Double-click an item to open the owning palette

ni.com

NATIONAL INSTRUMENTS

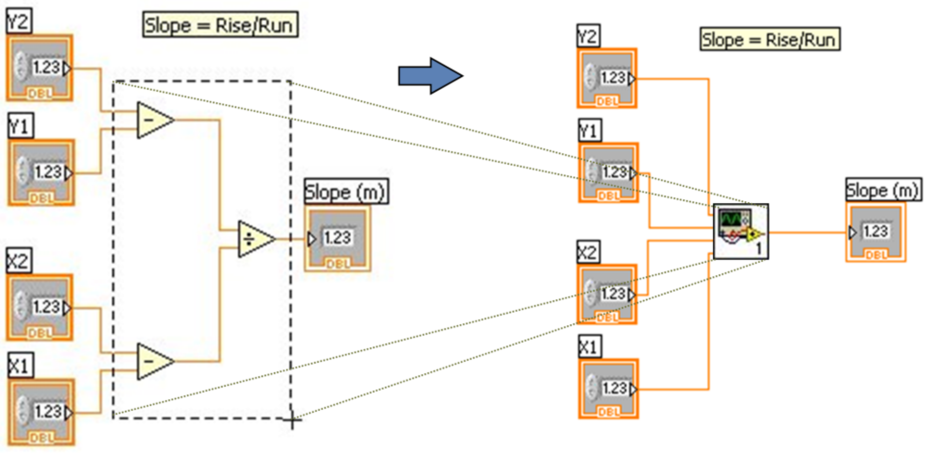
Use the buttons on top of the palette windows to navigate, search, and edit the palettes.

You can search for controls, VIs, and functions that either contain certain words or start with certain words. Double clicking a search result opens the palette that contains the search result. You also can click and drag the name of the control, VI, or function directly to the front panel or block diagram.



Create SubVI

- Enclose area to be converted into a subVI.
- Select **Edit»Create SubVI** from the Edit Menu.



ni.com

Creating SubVIs

After you build a VI, you can use it in another VI. A VI called from the block diagram of another VI is called a subVI. You can reuse a subVI in other VIs. To create a subVI, you need to build a connector pane and create an icon.

A subVI node corresponds to a subroutine call in text-based programming languages. A block diagram that contains several identical subVI nodes calls the same subVI several times.

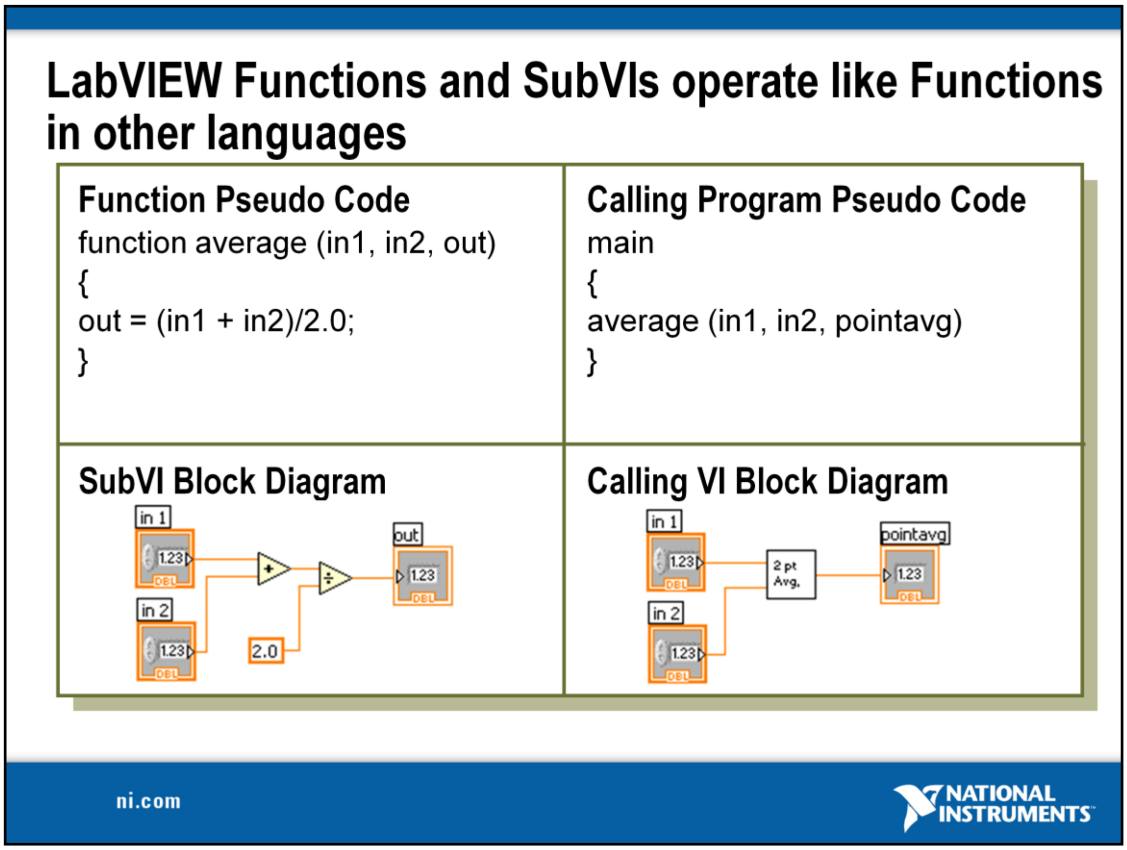
The subVI controls and indicators receive data from and return data to the block diagram of the calling VI. Click the **Select a VI** icon or text on the Functions palette, navigate to and double-click a VI, and place the VI on a block diagram to create a subVI call to that VI.

A subVI input and output terminals and the icon can be easily customized. Follow the instructions below to quickly create a subVI.

Creating SubVIs from Sections of a VI

Convert a section of a VI into a subVI by using the Positioning tool to select the section of the block diagram you want to reuse and selecting **Edit»Create SubVI**. An icon for the new subVI replaces the selected section of the block diagram. LabVIEW creates controls and indicators for the new subVI, automatically configures the connector pane based on the number of control and indicator terminals you selected, and wires the subVI to the existing wires.

See **Help»Search the LabVIEW Help…»SubVIs** for more information.



LabVIEW Functions and SubVIs operate like Functions in other languages

Function Pseudo Code

```
function average (in1, in2, out)
{
out = (in1 + in2)/2.0;
}
```

Calling Program Pseudo Code

```
main
{
average (in1, in2, pointavg)
}
```

SubVI Block Diagram

Calling VI Block Diagram

ni.com

NATIONAL INSTRUMENTS

A subVI node corresponds to a subroutine call in text-based programming languages. The node is not the subVI itself, just as a subroutine call statement in a program is not the subroutine itself. A block diagram that contains several identical subVI nodes calls the same subVI several times.

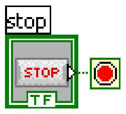
The modular approach makes applications easier to debug and maintain.

The functionality of the subVI does not matter for this example. The important point is the passing of two numeric inputs and one numeric output.

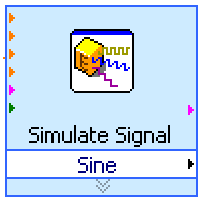
Exercise 3.1 – Analysis (Track A, B, & C)

Create a VI that produces a sine wave with a specified frequency and displays the data on a Waveform Chart until stopped by the user.

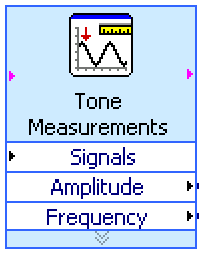
1. Open a blank VI from the Getting Started screen.
2. Place a chart on the front panel. Right-click to open the controls palette and select **Controls»Modern»Graph»Waveform Chart**.
3. Place a dial control on the front panel. From the controls palette, select **Controls»Modern »Numeric»Dial**. Notice that when you first place the control on the front panel, the label text is highlighted. While this text is highlighted, type "Frequency In" to give a name to this control.
4. Go to the block diagram (<Ctrl+E>) and place a while loop down. Right-click to open the functions palette and select **Express»Execution Control»While Loop**. Click and drag on the block diagram to make the while loop the correct size. Select the waveform chart and dial and drag them inside the while loop if they are not already. Notice that a stop button is already connected to the conditional terminal of the while loop.



5. Place the Simulate Signal Express VI on the block diagram. From the functions palette, select **Express»Signal Analysis»Simulate Signal** and place it on the block diagram inside the while loop. In the configuration window under Timing, choose "Simulate acquisition timing." Click "OK."



6. Place a Tone Measurements Express VI on the block diagram (**Express»Signal Analysis»Tone Measurements**). In the configuration window, choose Amplitude and Frequency measurements in the Single Tone Measurements section. Click "OK."



7. Make the following connections on the block diagram by hovering your mouse over the terminal so that it becomes the wiring tool and clicking once on each of the terminals you wish to connect:
   a. Connect the “Sine” output terminal of the Simulate Signal VI to the “Signals” input of the Tone Measurements VI.
   b. Connect the “Sine” output to the Waveform Chart.
   c. Create indicators for the amplitude and frequency measurements by right-clicking on each of the terminals of the Tone Measurements Express VI and selecting **Create»Numeric Indicator**.
   d. Connect the “Frequency In” control to the “Frequency” terminal of the Simulate Signal VI.
8. Return to the front panel and run the VI. Move the “Frequency In” dial and observe the frequency of the signal. Click the stop button once you are finished.
9. Save the VI as “Exercise 3.1 – Simulated.vi”.
10. Close the VI.

Notes

- When you bring up the functions palette, press the small push pin in the upper left hand corner of the palette. This will tack down the palette so that it doesn’t disappear. This step will be omitted in the following exercises, but should be repeated.
- The solution to this exercise is printed in the back of this manual.

(End of Exercise)

Exercise 3.2 – Analysis (Track A & B)

Create a VI that measures the frequency and amplitude of the signal from your (simulated) DAQ device and displays the acquired signal on a waveform chart. The instructions are the same as in Exercise 3.1, but a DAQ Assistant is used in place of the Simulate Signal VI. Try to do this without following the instructions!

1. Open a blank VI.
2. Place a chart on the front panel. Right click to open the controls palette and select **Controls»Modern»Graph»Waveform Chart**.
3. Go to the block diagram and place a while loop down (**Express»Execution Control»While Loop**).
4. Place a DAQ Assistant on the block diagram (**Express»Input»DAQ Assistant**). Choose analog input on channel ai0 of your (simulated) device and click "Finish." On the task timing tab, choose "continuous" for the acquisition mode. If you are using the USB-6009, change the Input Range to -2 to 2 and the number of Samples to Read to 100.
5. Place the Filter Express VI to the right of the DAQ Assistant on the block diagram. From the functions palette, select **Express»Signal Analysis»Filter** and place it on the block diagram inside the while loop. In the configuration window under Filtering Type, choose "Highpass." Under Cutoff Frequency, use a value of 300 Hz. Click "OK."
6. Connect the "Data" output terminal of the DAQ Assistant VI to the "Signal" input of the Filter VI.
7. Connect the "Filtered Signal" terminal on the Filter VI to the Waveform Chart.
8. Place a Tone Measurements Express VI on the block diagram (**Express»Signal Analysis»Tone**). In the configuration window, choose Amplitude and Frequency measurements in the Single Tone Measurements section.
9. Create indicators for the amplitude and frequency measurements by right clicking on each of the terminals of the Tone Measurements Express VI and selecting **Create»Numeric Indicator**.
10. Connect the output of the Filter VI to the "Signals" input of the Tone Measurements Express VI.
11. Return to the front panel and run the VI. Observe your acquired signal and its frequency and amplitude. Hum or whistle into the microphone if you have a USB-6009 and observe the amplitude and frequency that you are producing.
12. Save the VI as "Exercise 3.2 - Data.vi".
13. Close the VI.

Note: The solution to this exercise is printed in the back of this manual.

(End of Exercise)

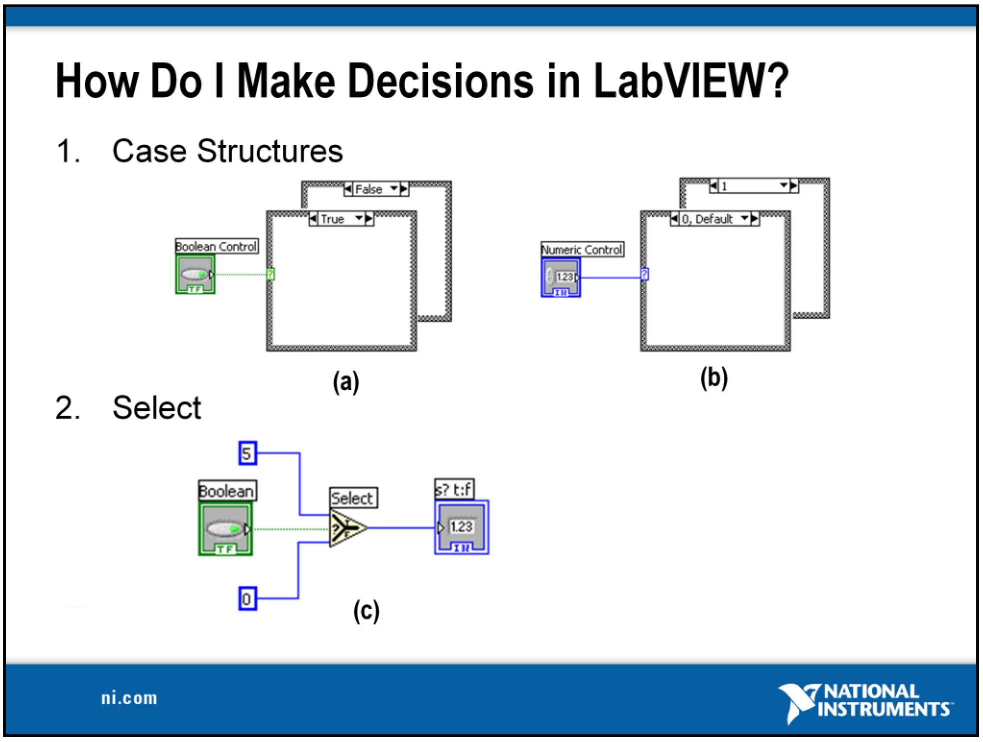
Exercise 3.2 – Analysis (Track C)

Create a VI that measures the frequency and amplitude of the signal from your sound card
and displays the acquired signal on a waveform chart. The instructions are the same as in
Exercise 3.1, but the Sound Signal VI is used in place of the Simulate Signal VI. Try to do
this without following the instructions!

1. Open a blank VI.
2. Go to the block diagram and place a While Loop down (**Express»Execution Control»While Loop**).
3. Place the Acquire Sound Express VI on the block diagram (**Express»Input» Acquire Sound**).
4. Place a Filter Express VI on the block diagram. In the configuration window choose a highpass filter and a cutoff frequency of 300 Hz.
5. Place a Tone Measurements Express VI on the block diagram (**Express»Signal Analysis**»**Tone**). In the configuration window, choose Amplitude and Frequency measurements in the Single Tone Measurements section.
6. Create indicators for the amplitude and frequency measurements by right-clicking on each of the terminals of the Tone Measurements Express VI and selecting **Create»Numeric Indicator**.
7. Connect the "Data" terminal of the Acquire Sound Express VI to the "Signal" input of the Filter VI.
8. Connect the "Filtered Signal" terminal of the Filter VI to the "Signals" input of the Tone Measurements VI.
9. Create a graph indicator for the Filtered Signal by right-clicking on the "Filtered Signal" terminal and selecting **Create**»**Graph Indicator.**
10. Return to the front panel and run the VI. Observe the signal from your sound card and its amplitude and frequency. Hum or whistle into the microphone and observe the amplitude and frequency you are producing.
11. Save the VI as "Exercise 3.2-Data.vi". Close the VI.

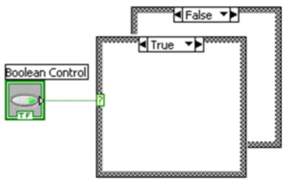
Note: The solution to this exercise is printed in the back of this manual.

(End of Exercise)

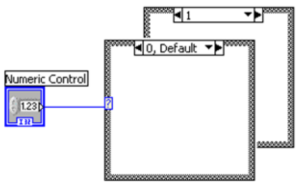


How Do I Make Decisions in LabVIEW?

1. Case Structures

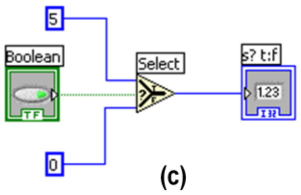


(a)



(b)

2. Select



(c)

ni.com

NATIONAL INSTRUMENTS

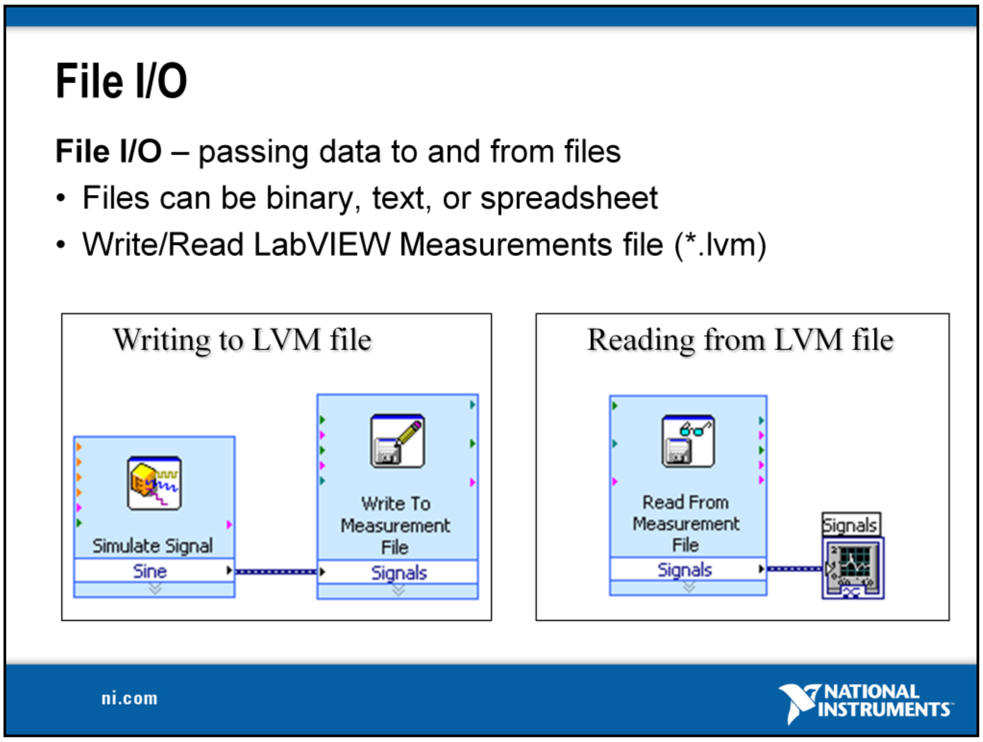
Case Structure

The Case Structure has one or more subdiagrams, or cases, exactly one of which executes when the structure executes. The value wired to the selector terminal determines which case to execute and can be boolean, string, integer, or enumerated type. Right-click the structure border to add or delete cases. Use the Labeling tool to enter value(s) in the case selector label and configure the value(s) handled by each case. It is found at **Functions»Programming»Structures»Case Structure**.

Select

Returns the value wired to the **t** input or **f** input, depending on the value of **s**. If **s** is TRUE, this function returns the value wired to **t**. If **s** is FALSE, this function returns the value wired to **f**. The connector pane displays the default data types for this polymorphic function. It is found at **Functions»Programming» Comparison»Select**.

- **Example a:** Boolean input: Simple if-then case. If the Boolean input is TRUE, the true case will execute; otherwise the FALSE case will execute.
- **Example b:** Numeric input. The input value determines which box to execute. If out of range of the cases, LabVIEW will choose the default case.
- **Example c:** When the Boolean passes a TRUE value to the Select VI, the value 5 is passed to the indicator. When the Boolean passes a FALSE value to the Select VI, 0 is passed to the indicator.



File I/O

File I/O – passing data to and from files

- Files can be binary, text, or spreadsheet
- Write/Read LabVIEW Measurements file (*.lvm)

Use LabVIEW measurement data files to save data that the Write Measurement File Express VI generates. The LabVIEW data file is a tab-delimited text file you can open with a spreadsheet application or a text-editing application. In addition to the data an Express VI generates, the .lvm file includes information about the data, such as the date and time the data was generated.

File I/O operations pass data from memory to and from files. In LabVIEW, you can use File I/O functions to:

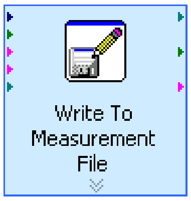
- Open and close data files
- Read data from and write data to files
- Read from and write to spreadsheet-formatted files
- Move and rename files and directories
- Change file characteristics
- Create, modify, and read a configuration file
- Write to or read from LabVIEW Measurements files.

In the next example we will examine how to write to or read from LabVIEW Measurements files (`*.lvm` files).

Exercise 3.3 – Decision Making and Saving Data (Track A, B, & C)

Create a VI that allows you to save your data to file if the frequency of your data goes below a user-controlled limit.

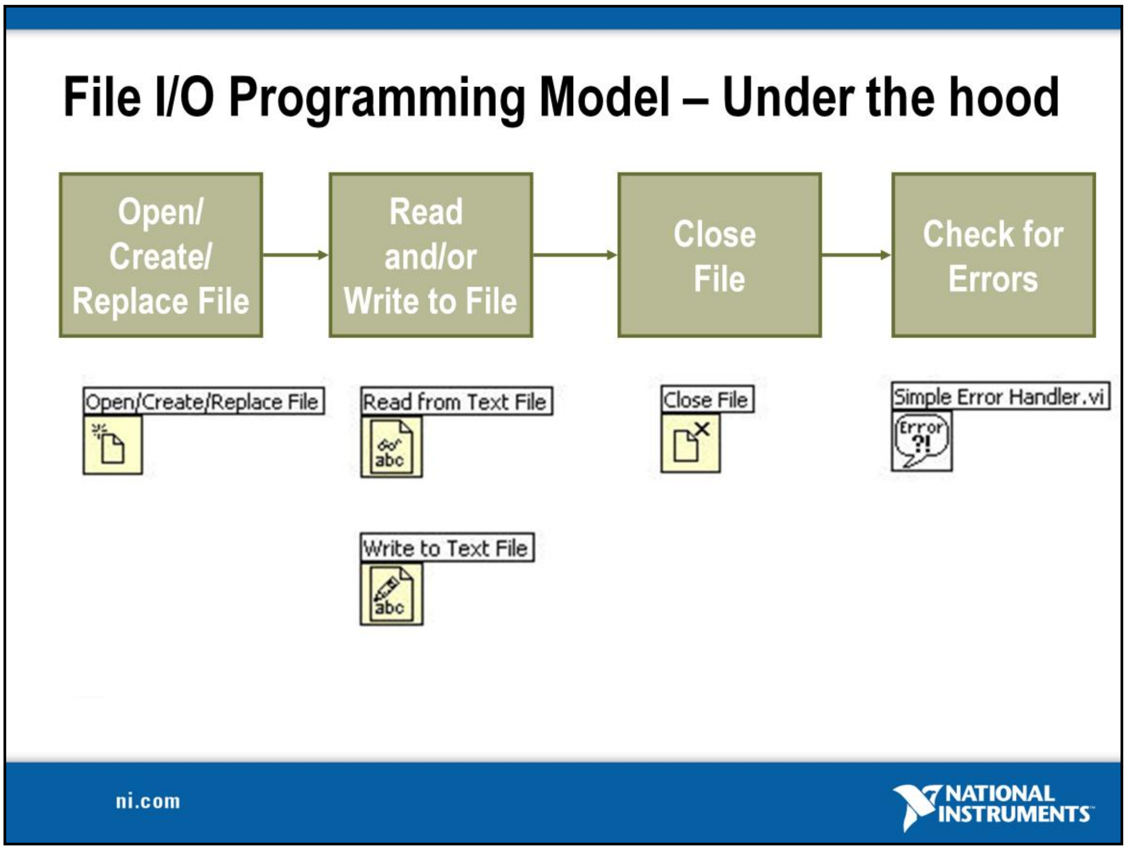
1. Open Exercise 3.2 – Data.vi.
2. Go to **File»Save As…** and save it as "Exercise 3.3 – Decision Making and Saving Data". In the "Save As" dialog box, make sure **substitute copy for original** is selected and click "Continue…".
3. Add a case structure to the block diagram inside the while loop (**Functions»Programming»Structures»Case Structure**).
4. Inside the "true" case of the case structure, add a Write to Measurement File Express VI (**Functions»Programming»File I/O»Write to Measurement File**).



   a. In the configuration window that opens, choose "Save to series of files (multiple files)." Note the default location your file will be saved to and change it if you wish.
   b. Click "Settings…" and choose "Use next available file name" under the **Existing Files** heading.
   c. Under **File Termination** choose to start a new file after 10 segments. Click "OK" twice.
5. Add code so that if the frequency computed from the Tone Measurements Express VI goes below a user-controlled limit, the data will be saved to file. **Hint:** Go to **Functions»Programming»Comparison»Less?**
6. Remember to connect your data from the DAQ Assistant or Acquire Sound Express VI to the "Signals" input of the Write to Measurement File VI. If you need help, refer to the solution to this exercise.
7. Go to the front panel and run your VI. Vary your frequency limit and then stop the VI.
8. Navigate to **My Documents»LabVIEW Data** and open one of the files that was saved there. Examine the file structure and check to verify that 10 segments are in the file.
9. Save your VI and close it.

Note: The solution to this exercise is printed in the back of this manual.

(End of Exercise)



File I/O Programming Model – Under the hood

Programming Model for the Intermediate File VIs

This same programming model applies to data acqusion, instrument control, file I/O, and most other communication schemes. In most instances you will open the file or communication channel, read and write multiple times, and then the communication will be closed or ended. It is also good programming practice to check for errors at the end. Remember this programming model when you move on to more advanced programming or look inside DAQ, communication, or file I/O Express VIs.

File I/O VIs and Functions

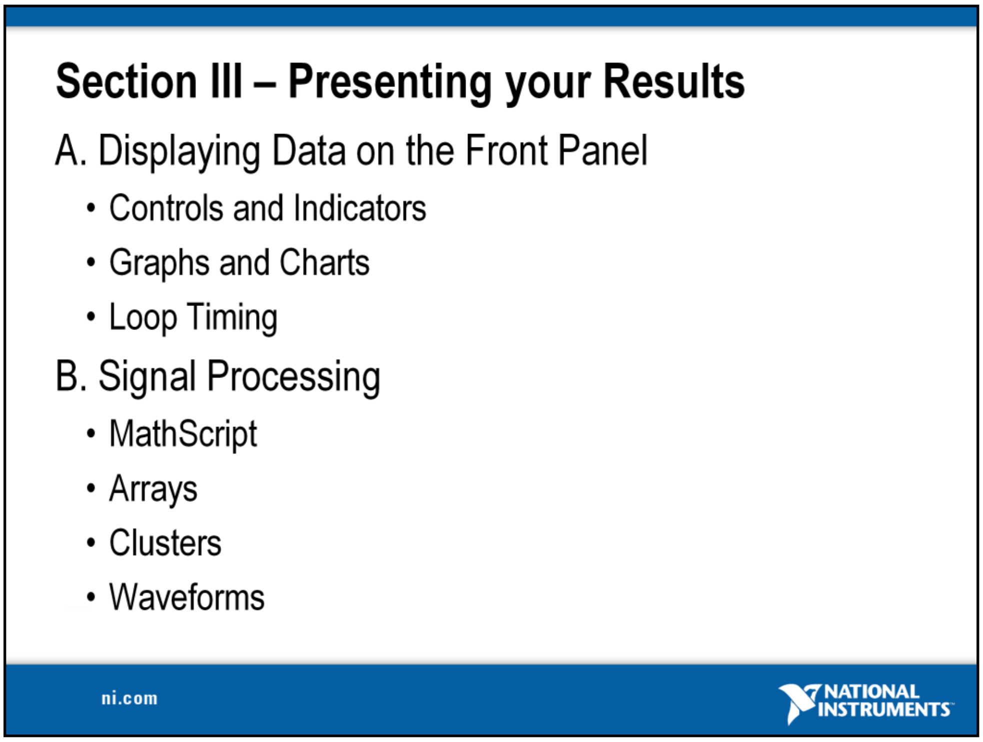
Use the File I/O VIs and functions to open and close files, read from and write to files, create directories and files you specify in the path control, retrieve directory information, and write strings, numbers, arrays, and clusters to files.

Use the high-level File I/O VIs located on the top row of the palette to perform common I/O operations, such as writing to or reading from various types of data. Acceptable types can include characters or lines in text files, 1D or 2D arrays of single-precision numeric values in spreadsheet text files, 1D or 2D arrays of single-precision numeric values in binary files, or 16-bit signed integers in binary files.

Use low-level File I/O VIs and functions located on the middle row of the palette, and the Advanced File Functions to control each file I/O operation individually.

Use the principal low-level functions to create or open, write data to, read data from, and close a file. You also can use low-level functions to create directories; move, copy, or delete files; list directory contents; change file characteristics; or manipulate paths.

Refer to the NI Developer Zone for more information about choosing a file format.



Section III – Presenting your Results

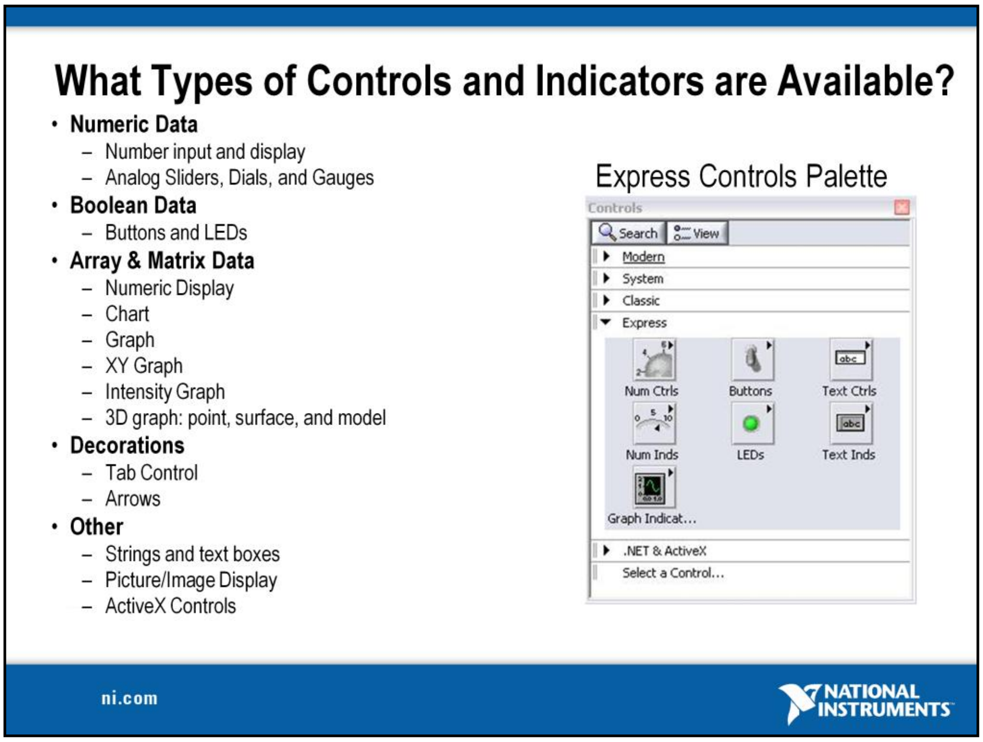
A. Displaying Data on the Front Panel

- Controls and Indicators
- Graphs and Charts
- Loop Timing

B. Signal Processing

- MathScript
- Arrays
- Clusters
- Waveforms

ni.com



What Types of Controls and Indicators are Available?

- **Numeric Data**
  - Number input and display
  - Analog Sliders, Dials, and Gauges
- **Boolean Data**
  - Buttons and LEDs
- **Array & Matrix Data**
  - Numeric Display
  - Chart
  - Graph
  - XY Graph
  - Intensity Graph
  - 3D graph: point, surface, and model
- **Decorations**
  - Tab Control
  - Arrows
- **Other**
  - Strings and text boxes
  - Picture/Image Display
  - ActiveX Controls

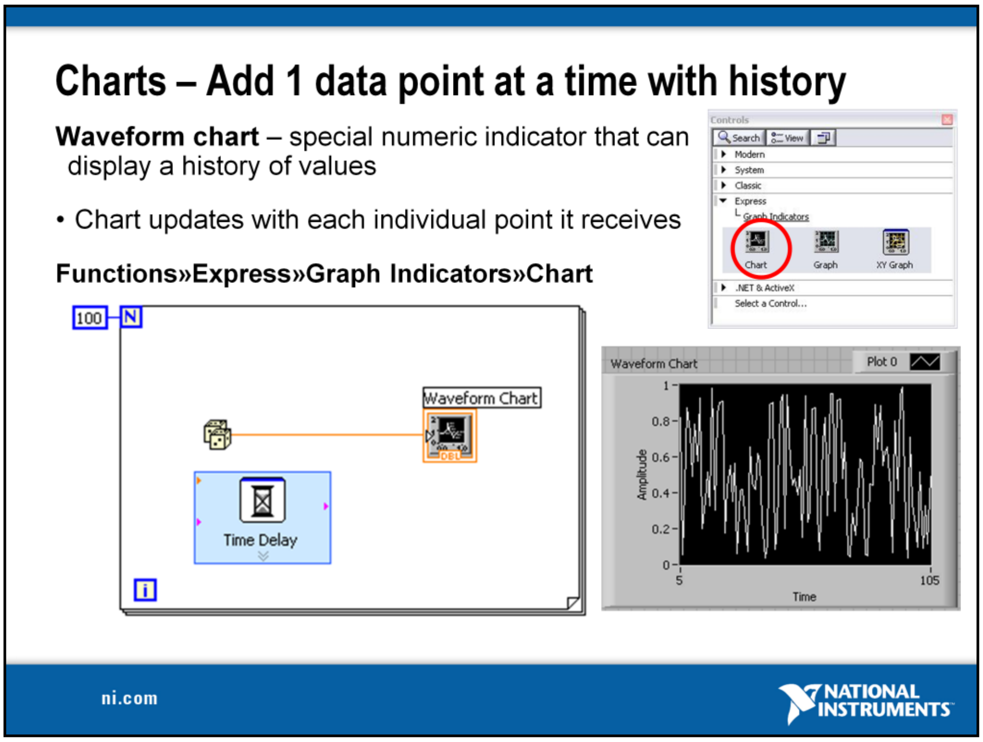
Express Controls Palette

ni.com

Controls and Indicators are Front Panel items that allow the user to interact with your program to both provide input and display results. You can access Controls and Indicators by right-clicking the front panel.

In addition, you will get additional controls and indicators when you install toolkits and modules.

For example, when you install the Control Design tools, you will get specialized plots such as Bode and Nyquist plots that are not available by default.



Charts – Add 1 data point at a time with history

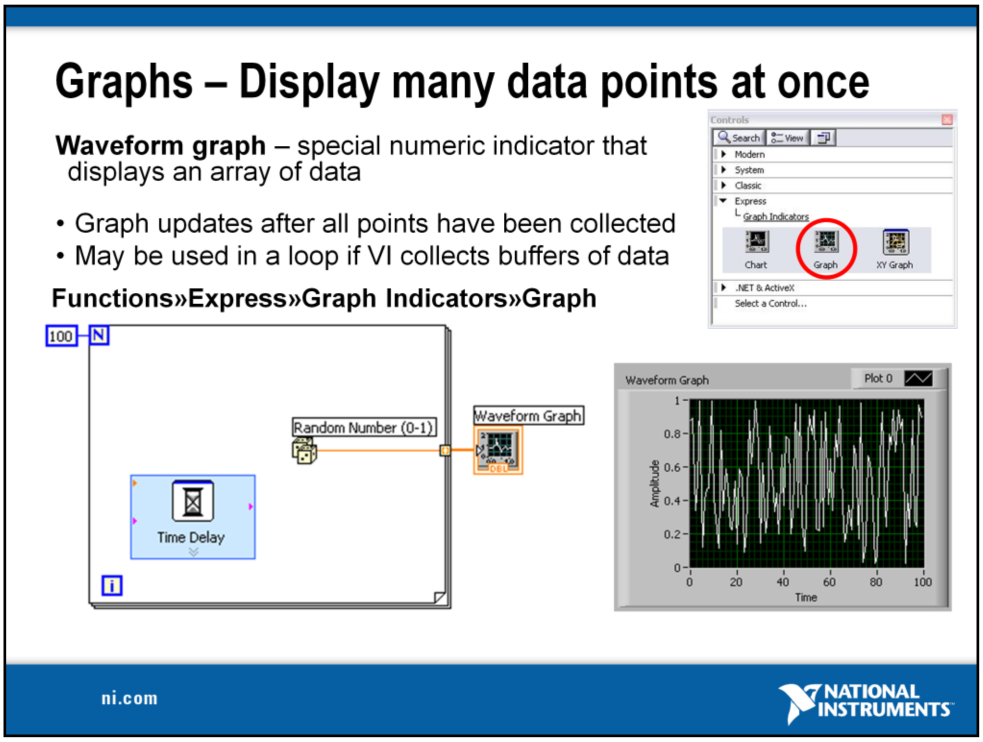
Waveform chart – special numeric indicator that can display a history of values

- Chart updates with each individual point it receives

Functions»Express»Graph Indicators»Chart

The waveform chart is a special numeric indicator that displays one or more plots. The waveform chart is located on the **Controls»Modern»Graph** palette. Waveform charts can display single or multiple plots. The following front panel shows an example of a multi-plot waveform chart.

You can change the min and max values of either the x or y axis by double clicking on the value with the labeling tool and typing the new value. Similarly, you can change the label of the axis. You can also right click the plot legend and change the style, shape, and color of the trace that is displayed on the chart.



Graphs – Display many data points at once

Waveform graph – special numeric indicator that displays an array of data

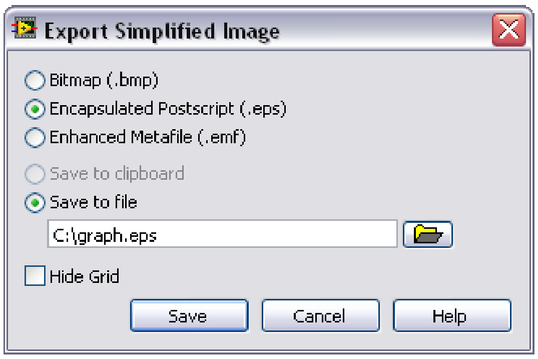
- Graph updates after all points have been collected
- May be used in a loop if VI collects buffers of data

Functions»Express»Graph Indicators»Graph

Graphs are very powerful indicators in LabVIEW. The can are highly customizable, and can be used to concisely display a great deal of information.

The properties page of the graph allows you to display settings for plot types, scale and cursor options, and many other features of the graph. To open the properties page, right-click the graph on the front panel and choose **Properties.**

Graphs also allow you to create technical paper quality graphics with the “export simplified image” function. Right-click the graph, select **Data Operations»Export Simplified Image…**

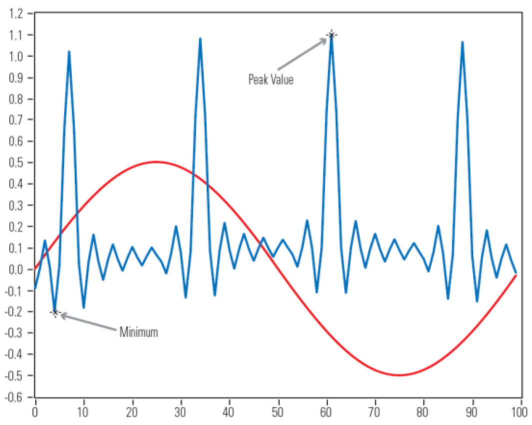


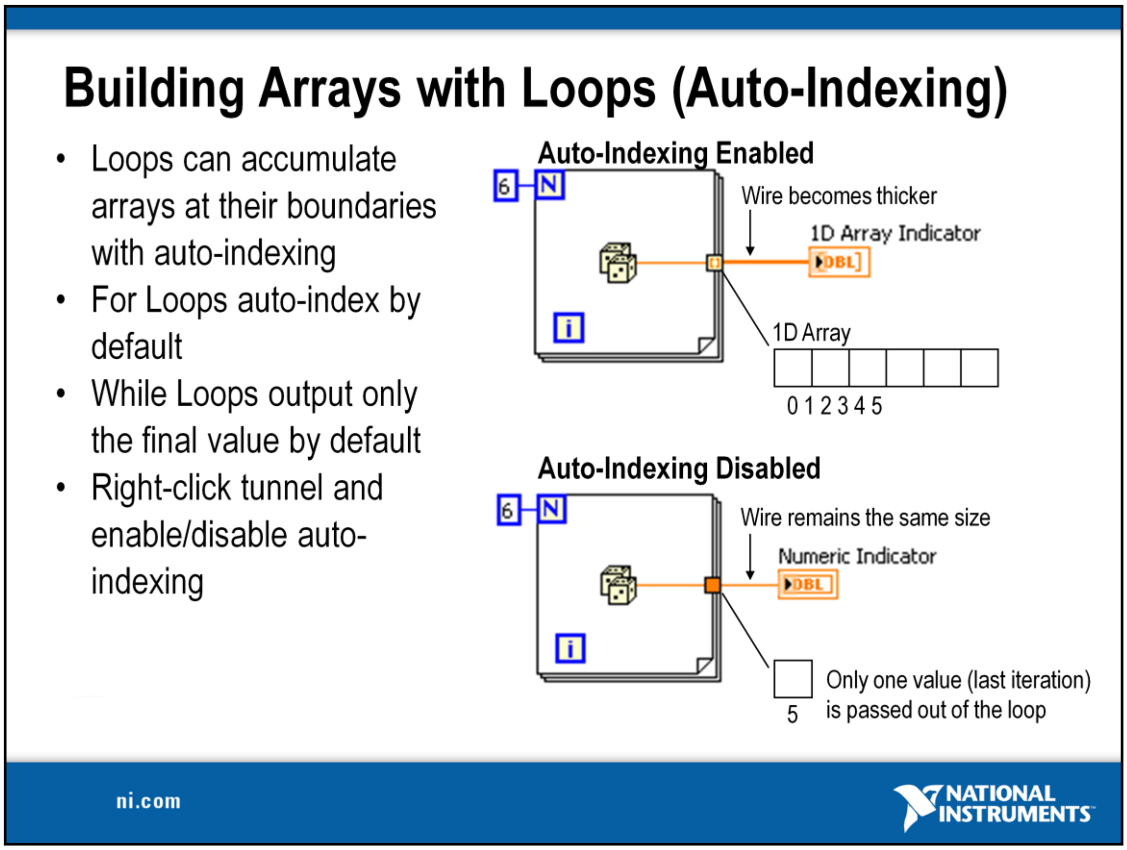
Export Simplified Image

- ○ Bitmap (.bmp)
- ● Encapsulated Postscript (.eps)
- ○ Enhanced Metafile (.emf)
- ○ Save to clipboard
- ● Save to file

C:\graph.eps

☐ Hide Grid





Building Arrays with Loops (Auto-Indexing)

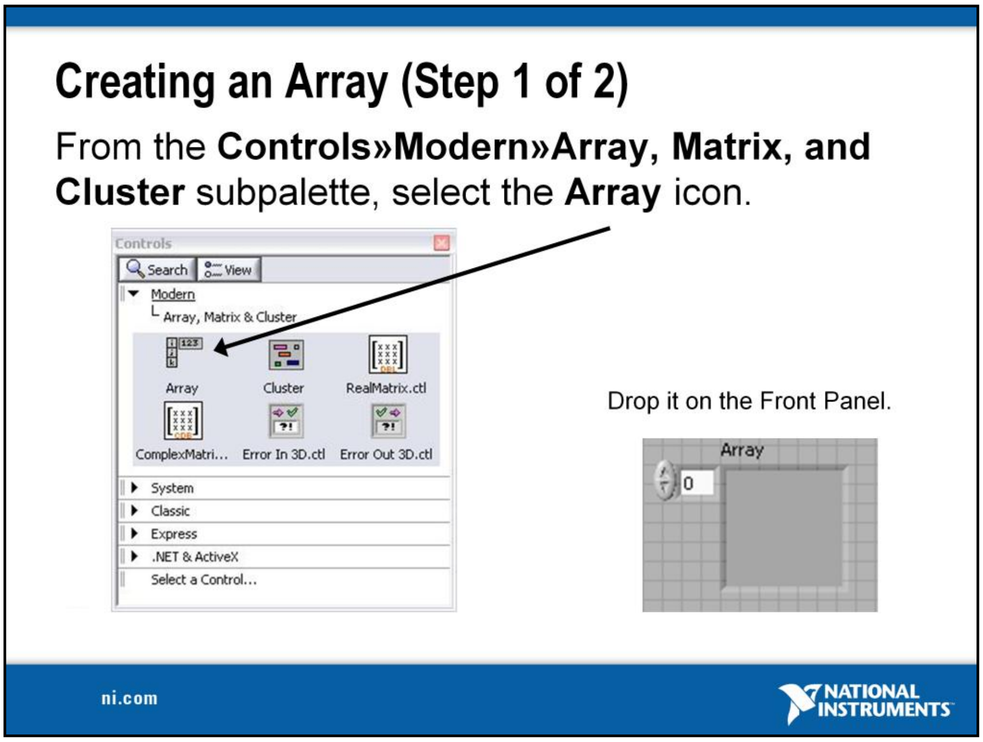
- Loops can accumulate arrays at their boundaries with auto-indexing
- For Loops auto-index by default
- While Loops output only the final value by default
- Right-click tunnel and enable/disable auto-indexing

For Loops and While Loops can index and accumulate arrays at their boundaries. This is known as auto-indexing.

- The indexing point on the boundary is called a tunnel.
- The For Loop default is auto-indexing enabled.
- The While Loop default is auto-indexing disabled.

Examples:

- Enable auto-indexing to collect values within the loop and build the array. All values are placed in array upon exiting loop.
- Disable auto-indexing if you are interested only in the final value.



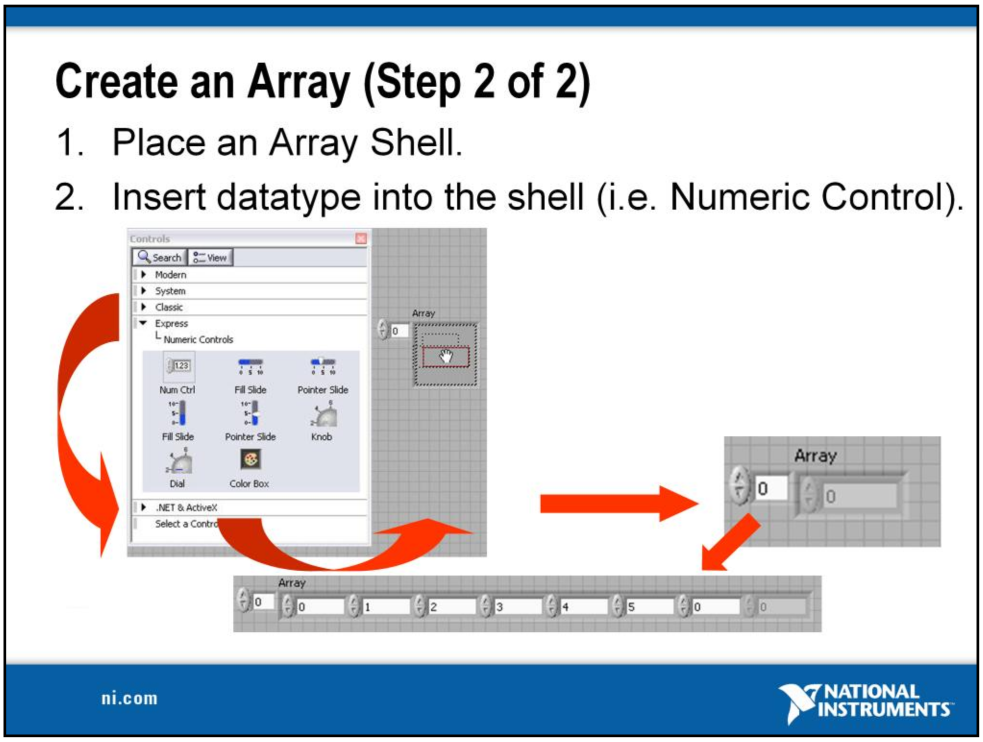
Creating an Array (Step 1 of 2)

From the **Controls»Modern»Array, Matrix, and Cluster** subpalette, select the **Array** icon.

Drop it on the Front Panel.

To create an array control or indicator as shown, select an array on the **Controls»Modern»Array, Matrix, and Cluster** palette, place it on the front panel, and drag a control or indicator into the array shell. If you attempt to drag an invalid control or indicator such as an XY graph into the array shell, you are unable to drop the control or indicator in the array shell.

You must insert an object in the array shell before you use the array on the block diagram. Otherwise, the array terminal appears black with an empty bracket.

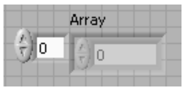


Create an Array (Step 2 of 2)

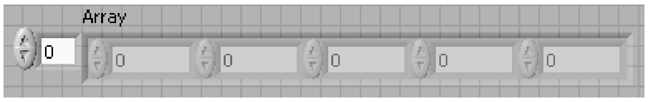
1. Place an Array Shell.
2. Insert datatype into the shell (i.e. Numeric Control).

To add dimensions to an array one at a time, right-click the index display and select **Add Dimension** from the shortcut menu. You also can use the Positioning tool to resize the index display until you have as many dimensions as you want.

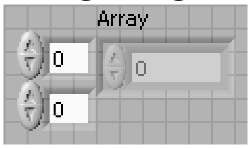
1D Array Viewing a Single Element:



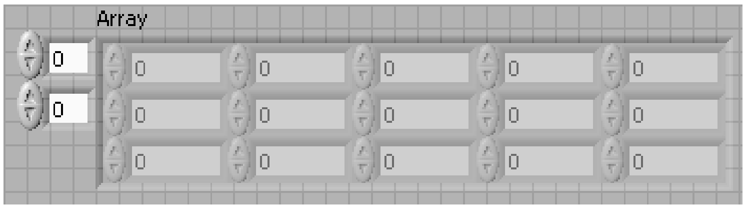
1D Array Viewing Multiple Elements:

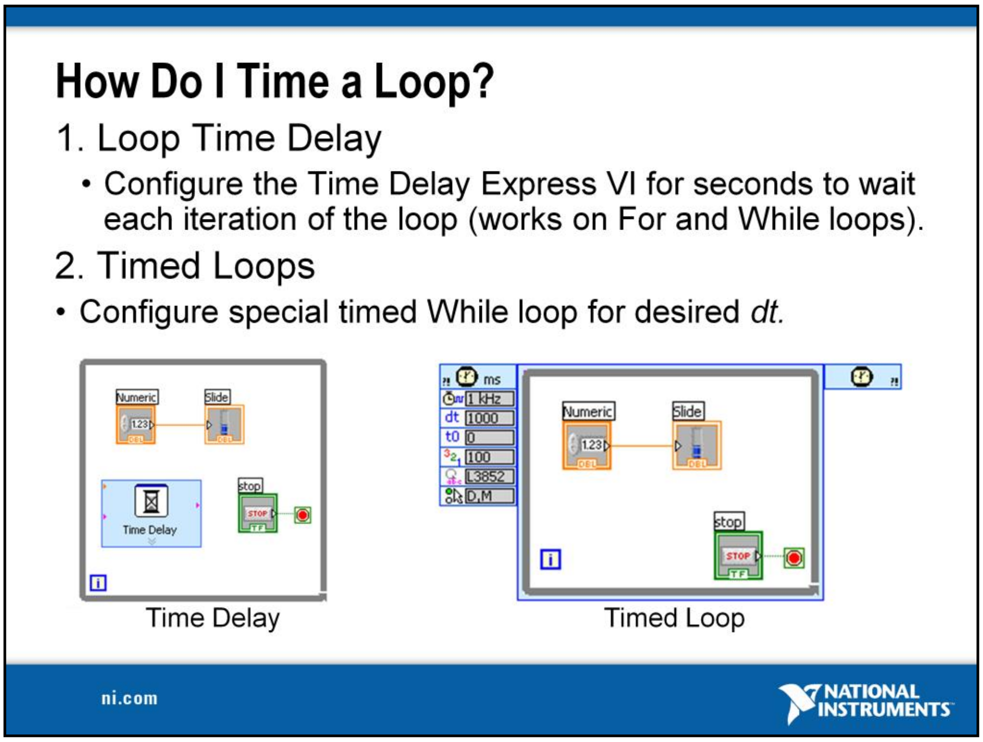


2D Array Viewing a Single Element:



2D Array Viewing Multiple Elements:





How Do I Time a Loop?

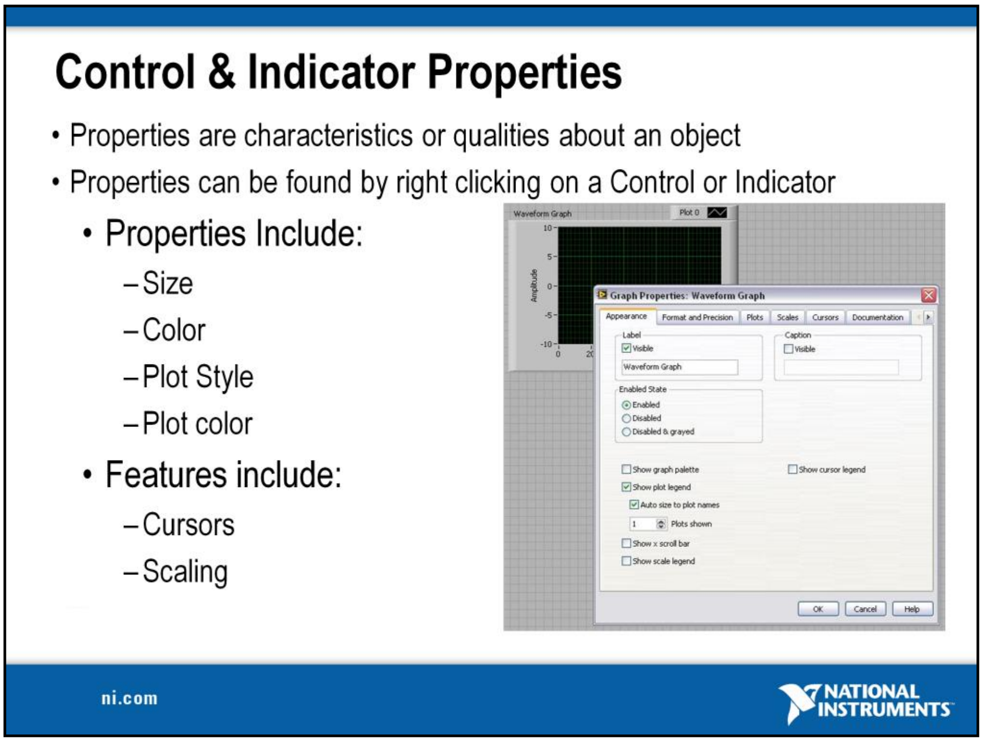
1. Loop Time Delay
   - Configure the Time Delay Express VI for seconds to wait each iteration of the loop (works on For and While loops).
2. Timed Loops
   - Configure special timed While loop for desired *dt*.

Time Delay Timed Loop

ni.com NATIONAL INSTRUMENTS

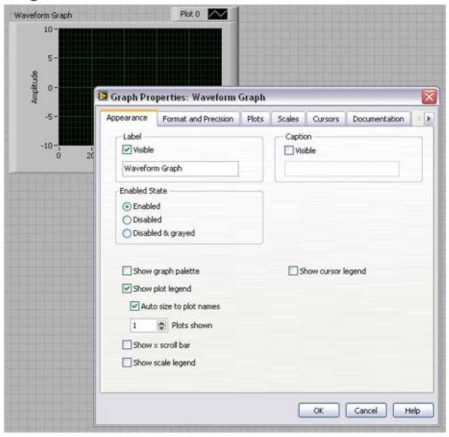
Time Delay
The Time Delay Express VI delays execution by a specified number of seconds. Following the rules of Data Flow Programming, the while loop will not iterate until all tasks inside of it are complete, thus delaying each iteration of the loop.
Timed Loops
Executes each iteration of the loop at the period you specify. Use the Timed Loop when you want to develop VIs with multi-rate timing capabilities, precise timing, feedback on loop execution, timing characteristics that change dynamically, or several levels of execution priority.
Double-click the Input Node or right-click the Input Node and select **Configure Timed Loop** from the shortcut menu to display the Loop Configuration dialog box, where you can configure the Timed Loop. The values you enter in the **Loop Configuration** dialog box appear as options in the Input Node.

Wait Until Next ms Multiple
Waits until the value of the millisecond timer becomes a multiple of the specified **millisecond multiple**. Use this function to synchronize activities. You can call this function in a loop to control the loop execution rate. However, it is possible that the first loop period might be short. This function makes asynchronous system calls, but the nodes themselves function synchronously. Therefore, it does not complete execution until the specified time has elapsed.
Functions»Programming»Timing»Wait Until Next ms Multiple



Control & Indicator Properties

- Properties are characteristics or qualities about an object
- Properties can be found by right clicking on a Control or Indicator
  - Properties Include:
    - Size
    - Color
    - Plot Style
    - Plot color
  - Features include:
    - Cursors
    - Scaling



Label
[✓] Visible
Waveform Graph

Caption
[ ] Visible

Enabled State
[✓ Enabled] Disabled Disabled & grayed

[ ] Show graph palette
[ ] Show cursor legend
[✓] Show plot legend
[✓] Auto size to plot names
1 Plots shown
[ ] Show x scroll bar
[ ] Show scale legend

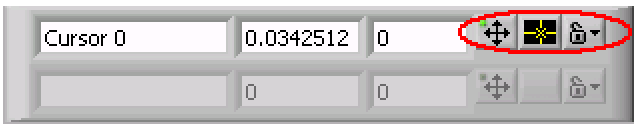
ni.com

Properties are all the qualities of a front panel object. With properties, you can set or read such characteristics as foreground and background color, data formatting and precision, visibility, descriptive text, size and location on the front panel, and so on.

Exercise 4.1 – Manual Analysis (Track A, B, & C)

Create a VI that displays simulated data on a waveform graph and measures the frequency and amplitude of that data. Use cursors on the graph to verify the frequency and amplitude measurements.

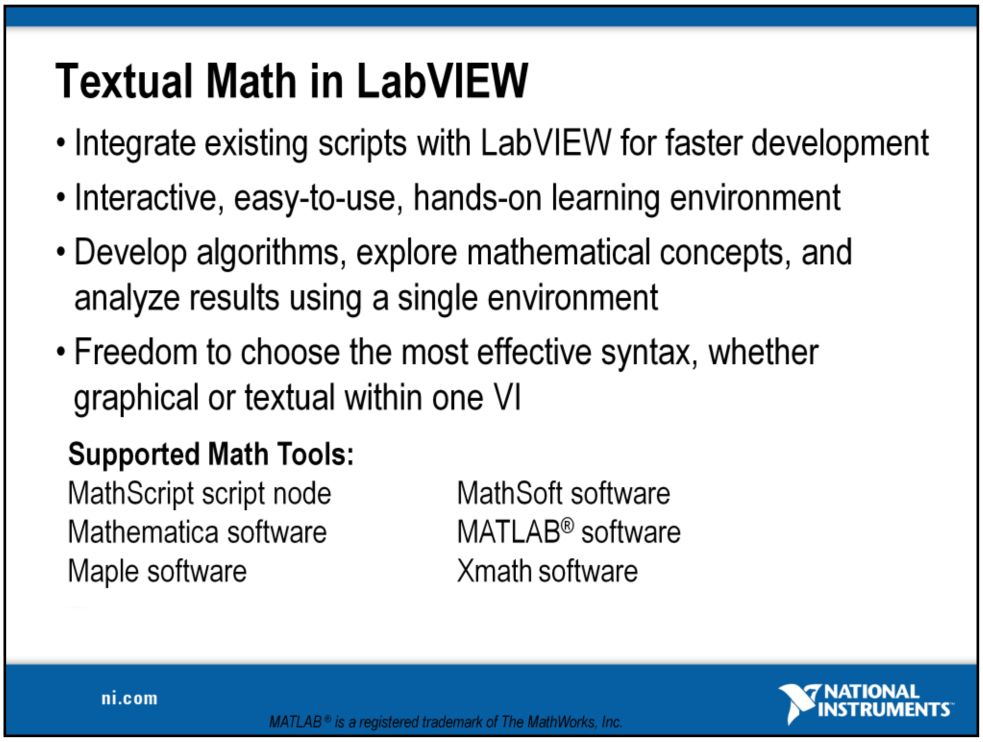
1. Open Exercise 3.1 – Simulated.vi.
2. Save the VI as "Exercise 4.1 – Manual Analysis.vi".
3. Go to the block diagram and remove the While Loop. Right-click the edge of the loop and choose **Remove While Loop** so that the code inside the loop does not get deleted.
4. Delete the stop button.
5. On the front panel, replace the waveform chart with a waveform graph. Right-click the chart and select **Replace»Modern»Graph»Waveform Graph**.
6. Make the cursor legend viewable on the graph. Right-click on the graph and select **Visible Items»Cursor Legend**.
7. Change the maximum value of the "Frequency In" dial to 100. Double-click on the maximum value and type "100" once the text is highlighted.
8. Set a default value for the "Frequency In" dial by setting the dial to the value you would like, right-clicking the dial, and selecting **Data Operations»Make Current Value Default**.
9. Run the VI and observe the signal on the waveform graph. If you cannot see the signal, you may need to turn on auto-scaling for the x-axis. Right-click on the graph and select **X Scale»AutoScale X**.
10. Change the frequency of the signal so you can see a few periods on the graph.
11. Manually measure the frequency and amplitude of the signal on the graph using cursors. To make the cursors display on the graph, click on one of the three buttons in the cursor legend. Once the cursors are displayed, you can drag them around on the graph and their coordinates will be displayed in the cursor legend.



12. Remember that the frequency of a signal is the reciprocal of its period (f = 1/T). Does your measurement match the frequency and amplitude indicators from the Tone Measurements VI?
13. Save your VI and close it.

Note: The solution to this exercise is printed in the back of this manual.

(End of Exercise)



Textual Math in LabVIEW

- Integrate existing scripts with LabVIEW for faster development
- Interactive, easy-to-use, hands-on learning environment
- Develop algorithms, explore mathematical concepts, and analyze results using a single environment
- Freedom to choose the most effective syntax, whether graphical or textual within one VI

Supported Math Tools:

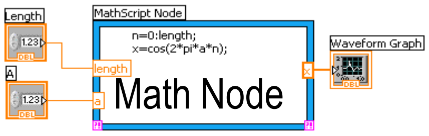
MathScript script node	MathSoft software
Mathematica software	MATLAB® software
Maple software	Xmath software

ni.com

MATLAB® is a registered trademark of The MathWorks, Inc.

NATIONAL INSTRUMENTS

Overview
With the release of National Instruments LabVIEW 8, you have new freedom to choose the most effective syntax for technical computing, whether you are developing algorithms, exploring DSP concepts, or analyzing results. You can instrument your scripts and develop algorithms on the block diagram by interacting with popular third-party math tools such as The MathWorks Inc. MATLAB software, Mathematica, Maple, Mathcad, IDL and Xmath. Use of these math tools with LabVIEW is achieved in a variety of ways depending on the vendor as listed below:



```
n=0:length;
x=cos(2*pi*a*n);
```

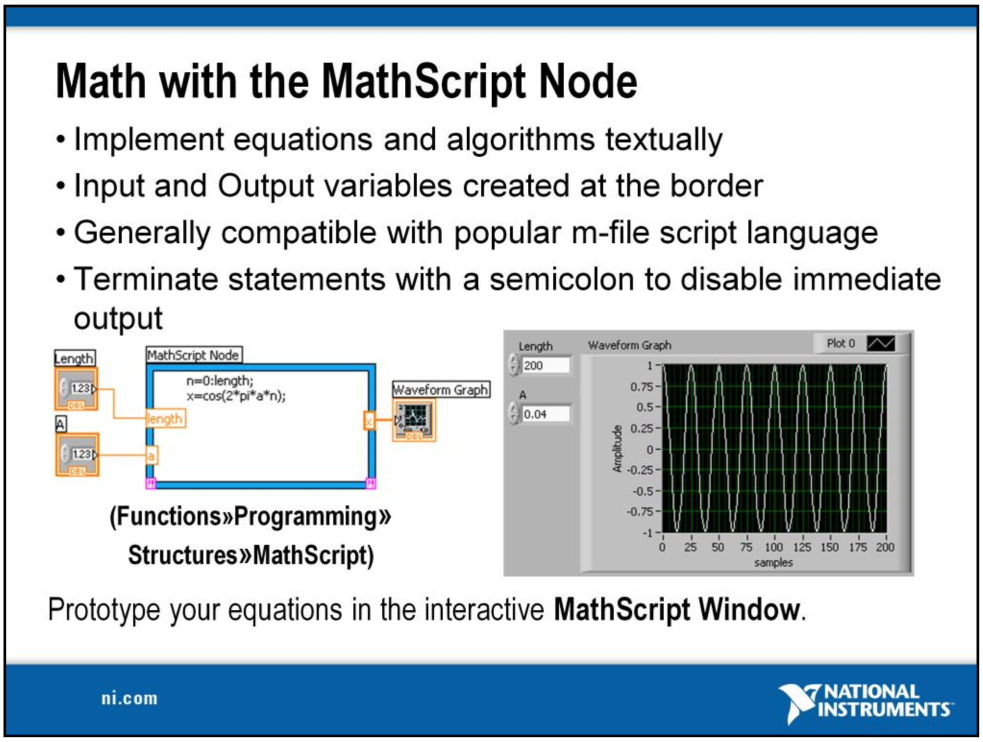
Native LabVIEW textual math node:
MathScript node, Formula node

Communication with vendor software through LabVIEW node:
Xmath node, MATLAB script node, Maple* node, IDL* node

Communication with vendor software through VI Server:
Mathematica* VIs, and Mathcad* VIs

In LabVIEW 8, you can combine the intuitive LabVIEW graphical dataflow programming with MathScript, a math-oriented textual programming language that is generally compatible with popular m-file script language.

*LabVIEW toolkit specific to the math tool must be installed.



Math with the MathScript Node

- Implement equations and algorithms textually
- Input and Output variables created at the border
- Generally compatible with popular m-file script language
- Terminate statements with a semicolon to disable immediate output

```
n=0:length;
x=cos(2*pi*a*n);
```

(Functions»Programming» Structures»MathScript)

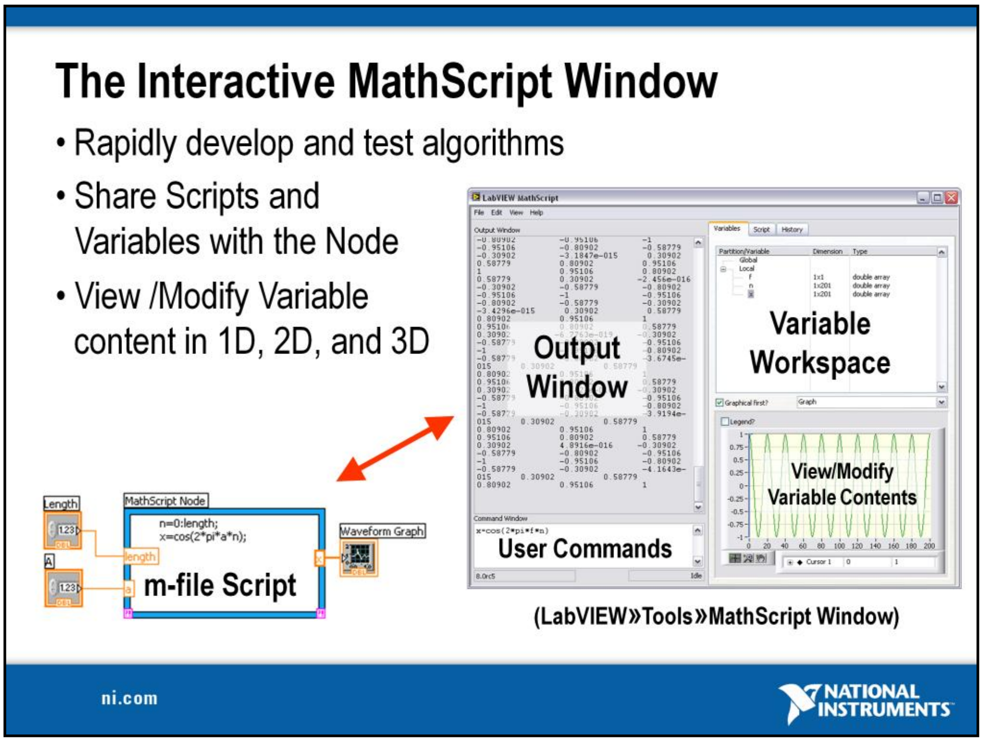
Prototype your equations in the interactive **MathScript Window**.

ni.com

The MathScript Node enhances LabVIEW by adding a native text-based language for mathematical algorithm implementation in the graphical programming environment. M-file scripts you've written and saved from the MathScript window can be opened and used in the MathScript node. M-file scripts you created in other math software will generally run as well. The MathScript allows you to pick the syntax you are most comfortable with to solve the problem. Equations can be instrumented with the MathScript Node for parameter exploration, simulation, or deployment in a final application.

The MathScript Node:

- Located in the **Programming»Structures** subpalette.
- Resizable box for entering textual computations directly into block diagrams.
- To add variables, right-click and choose **Add Input** or **Add Output**.
- Name variables as they are used in formula. (Names are case sensitive.)
- The data type of the output can be changed by right-clicking the input or output node.
- Statements should be terminated with a semicolon to suppress output.
- Ability to import & export m-files by right-clicking on the node.



The Interactive MathScript Window

- Rapidly develop and test algorithms
- Share Scripts and Variables with the Node
- View /Modify Variable content in 1D, 2D, and 3D

```
n=0:length;
x=cos(2*pi*a*n);
```

```
x=cos(2*pi*f*n)
```

(LabVIEW»Tools»MathScript Window)

ni.com

The MathScript Window provides an interactive environment where equations can be prototyped and calculations can be made. The MathScript Window and Node share a common syntax and global variables making the move from prototype to implementation seamless. The data preview pane provides a convenient way to view variable data as numbers, graphically, or audibly (with soundcard support).

Help for MathScript

Help for the environment can be accessed using the Mathscript Interactive Environment Window. Type **Help** in the command window for an introduction to MathScript help. **Help** followed by a **function** will display help specific to that function.

Features of the interactive MathScript Window:

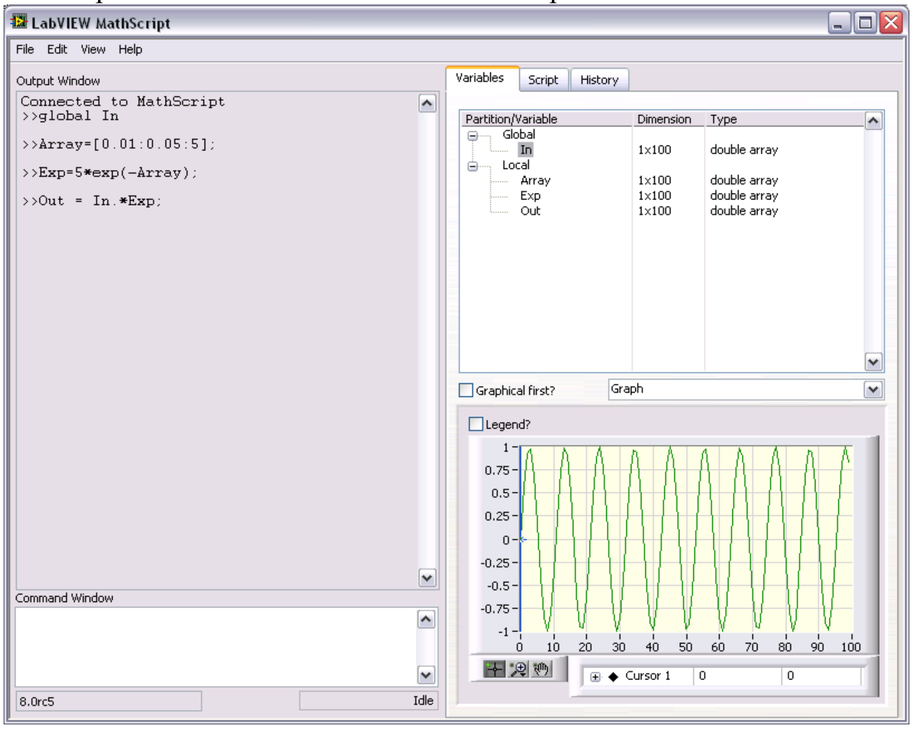
- Prototype equations and formulas through the command Window
- Easily access function help by typing **Help <function>** in the Command Window
- Select a variable to display its data in the Preview Pane and even listen to the result
- Write, Save, Load, and Run m-files using the Script tab
- Share data between the MathScript Node in LabVIEW and the MathScript Window using Global Variables
- Advanced plotting features and image export features

Exercise 4.2 – MathScript (Track A, B, & C)

Create a VI that uses the MathScript Node to alter your simulated signal and graph it. Use
the Interactive MathScript Window to view and alter the data and then load the script you
have created back into the MathScript Node.

1. Open Exercise 4.1 – Manual Analysis.vi.
2. Save the VI as "Exercise 4.2 – MathScript.vi".
3. Go to the block diagram and delete the wire connecting the Simulate Signal VI to the Waveform Graph.
4. Place down a MathScript Node (**Programming»Structures»MathScript Node**).
5. Right-click on the left border of the MathScript Node and select **Add Input**. Name this input "In" by typing while the input node is highlighted black.
6. Right-click on the right border of the MathScript Node and select **Add Output**. Name this output "Out".
7. Convert the Dynamic Data Type output of the Simulate Signals VI to a 1D Array of Scalars to input to the MathScript Node. Place a Convert from Dynamic Data Express VI on the block diagram (**Express»Signal Manipulation»Convert from Dynamic Data**). By default, the VI is configured correctly so click "OK" in the configuration window.
8. Wire the "Sine" output of the Simulate Signal VI to the "Dynamic Data" input of the Convert from Dynamic Data VI.
9. Wire the "Array" output of the Convert from Dynamic Data VI to the "In" node on the MathScript Node.
10. In order to use the data from the Simulate Signal VI in the Interactive MathScript Window it is necessary to declare the input variable as a global variable. Inside the MathScript Node type "global In;".
11. Return to the front panel and increase the frequency to be between 50 and 100. Run the VI.
12. Open the Interactive MathScript Window (**Tools»MathScript Window…**).
13. In the MathScript Window, the Command Window can be used to enter in the command that you wish to compute. In the Command Window, type "global In" and press "Enter". This will allow you to see the data passed to the variable "In" on the MathScript Node.

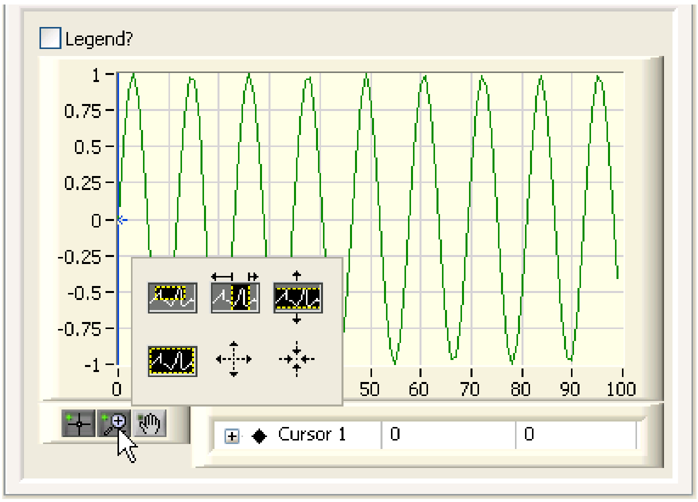
14. Notice that all declared variables in the script along with their dimensions and type are listed on the “Variables” tab. To display the graphed data, click once on the variable **In** and change the drop down menu from “Numeric” to “Graph”.



```
Connected to MathScript
>>global In

>>Array=[0.01:0.05:5];

>>Exp=5*exp(-Array);

>>Out = In.*Exp;
```

Partition/Variable	Dimension	Type
Global		
In	1x100	double array
Local		
Array	1x100	double array
Exp	1x100	double array
Out	1x100	double array

15. Use the graph palette to zoom in on your data.



16. Right-click on “Cursor 1” and choose **Bring to Center**. What does this do?

17. Drag the cursor around. The cursor will not move if the zoom option is selected.

18. Right-click on the graph and choose **Undock Window**. What does this do? Close this new window when you are finished.

19. Multiply the data by a decreasing exponential function. Follow these steps:

    a. Make a 100 element array of data that constitutes a ramp function going from 0.01 to 5 by typing “Array = [0.01:0.05:5];” in the Command Window and pressing Enter. What type of variable is “Array”?

    b. Make an array containing a decreasing exponential. Type “Exp = 5*exp(-Array);” and press Enter.

    c. Now multiply the Exp and In arrays element by element by typing “Out = In**.***Exp;” and pressing Enter.

    d. Look at the graph of the variable “Out”.

20. Go to the History tab and use Ctrl-click to choose the 4 commands you just entered. Copy those commands using <Ctrl-C>.

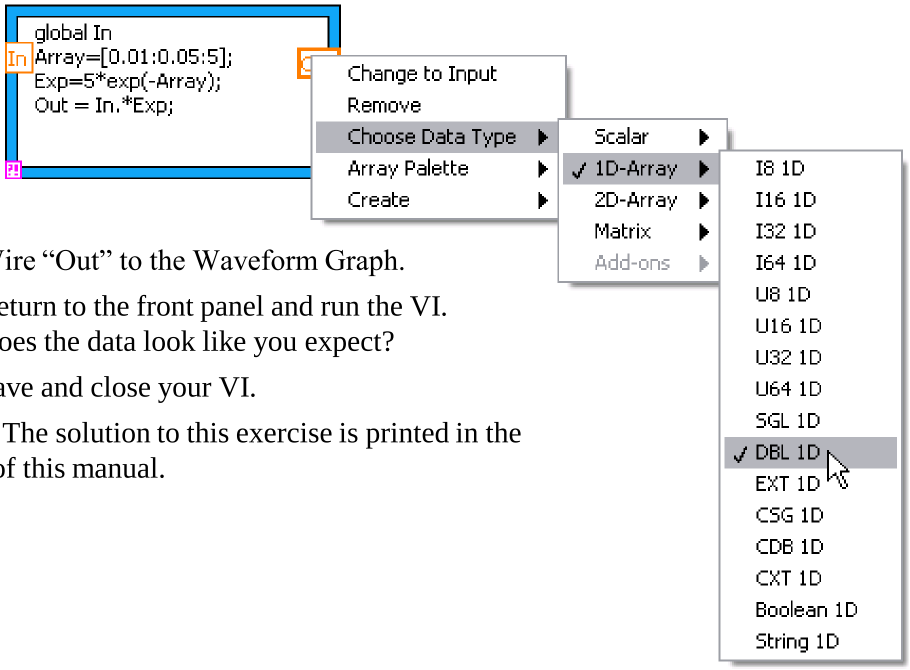
21. On the Script tab, paste the commands into the Script Editor using <Ctrl-V>.

22. Save your script by clicking “Save” at the bottom of the window. Save it as “myscript.txt”

23. Close the MathScript Window.

24. Return to the block diagram of Exercise 4.2 – MathScript. Load the script you just made by right-clicking on the MathScript Node border and selecting **Import…** Navigate to myscript.txt, select it, and click “OK”.

25. Right-click on the variable “Out” and select **Choose Data Type»1D-Array»DBL 1D**. Output data types must be set manually on the MathScript Node.



```
global In
Array=[0.01:0.05:5];
Exp=5*exp(-Array);
Out = In.*Exp;
```

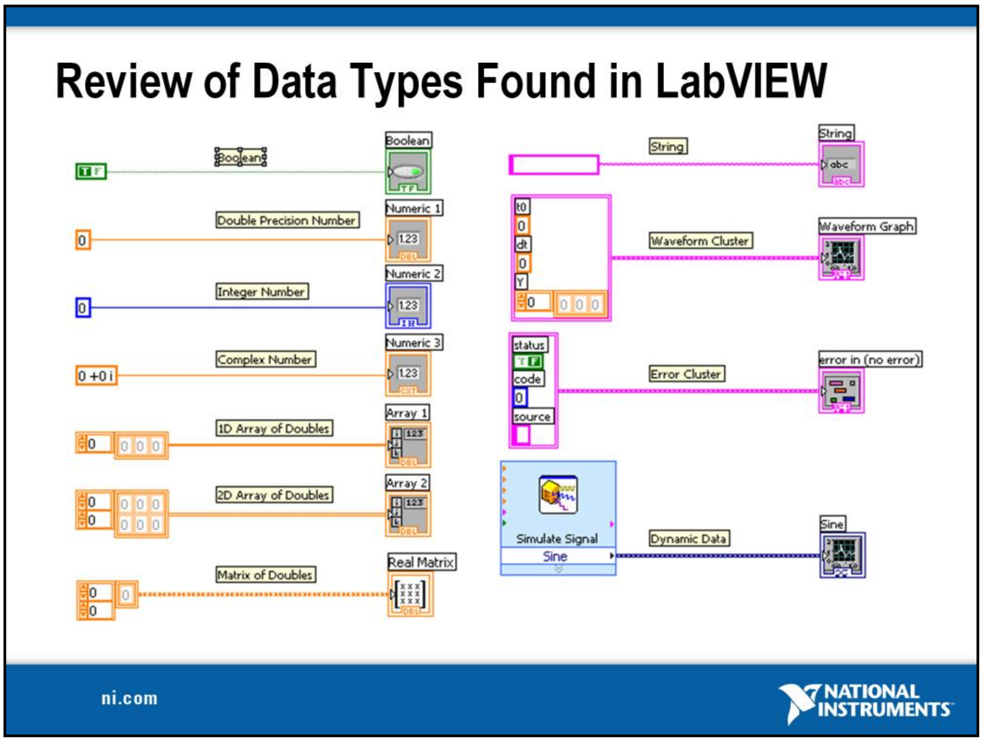
26. Wire “Out” to the Waveform Graph.

27. Return to the front panel and run the VI. Does the data look like you expect?

25. Save and close your VI.

Note: The solution to this exercise is printed in the back of this manual.

(End of Exercise)


Review of Data Types Found in LabVIEW

LabVIEW utilizes many common datatypes. These Datatypes include:

Boolean, Numeric, Arrays, Strings, Clusters, and more.

The color and symbol of each terminal indicate the data type of the control or indicator. Control terminals have a thicker border than indicator terminals. Also, arrows appear on front panel terminals to indicate whether the terminal is a control or an indicator. An arrow appears on the right if the terminal is a control, and an arrow appears on the left if the terminal is an indicator.

Definitions

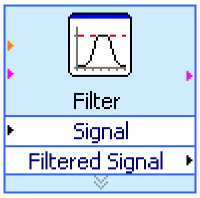
- **Array:** Arrays group data elements of the same type. An array consists of elements and dimensions. Elements are the data that make up the array. A dimension is the length, height, or depth of an array. An array can have one or more dimensions and as many as $(2^{31}) - 1$ elements per dimension, memory permitting.
- **Cluster:** Clusters group data elements of mixed types, such as a bundle of wires in a telephone cable, where each wire in the cable represents a different element of the cluster.

See **Help»Search the LabVIEW Help…** for more information. The *LabVIEW User Manual* on `ni.com` provides additional reference for data types found in LabVIEW.

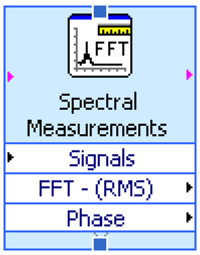
Exercise 5 – Apply What You Have Learned (Track A, B, & C)

In this exercise, you will create a VI that uses what you have learned. Design a VI that does the following:

1. Acquire data from your device and graph it (either your DAQ device, your simulated device, or your sound card).
2. Filter that data using the Filter Express VI (**Functions»Express»Signal Analysis»Filter**). There should be a front panel control for a user configurable cut-off frequency.

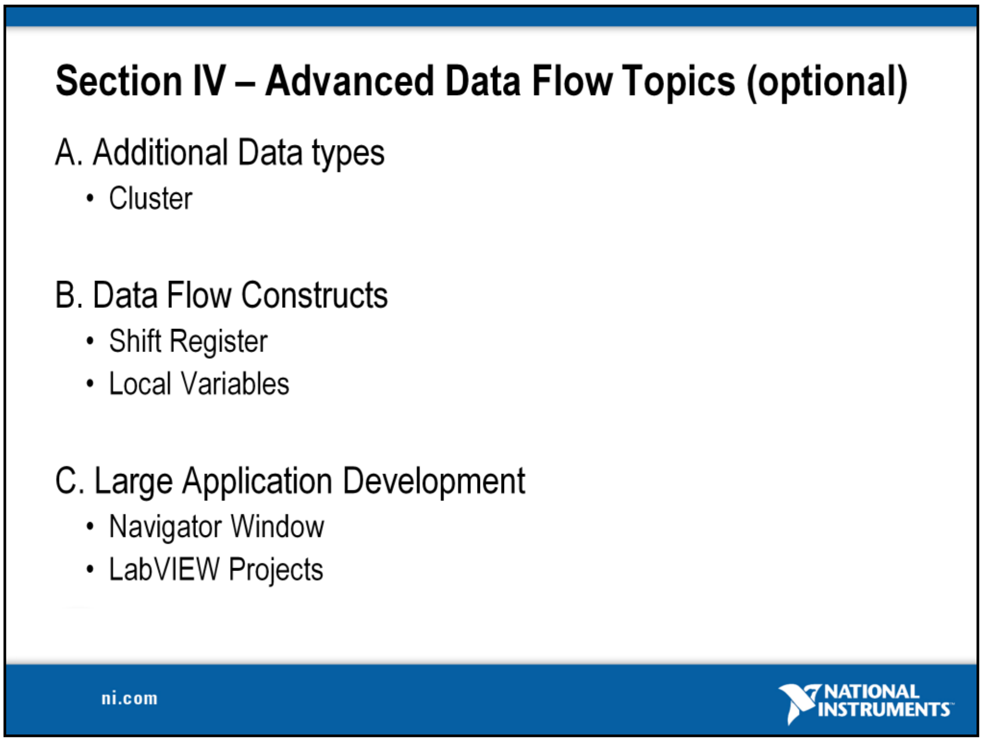


3. Take a Fast Fourier Transform to get the frequency information from the filtered data and graph the result. Use the Spectral Measurements Express VI (**Functions»Express»Signal Analysis»Spectral**).



4. Find the dominant frequency of the filtered data using the Tone Measurements Express VI.
5. Compare that frequency to a user inputted limit. If the frequency is over that limit, light up an LED. If you have a USB-6009, light up the LED on your hardware using the DAQ Assistant. You will need to invert the digital line for the LED to light up when over the limit. You can specify this in the configuration window of the DAQ Assistant or with a "not" boolean function.
6. If you get stuck, open up the solution or view it at the end of this manual.

(End of Exercise)



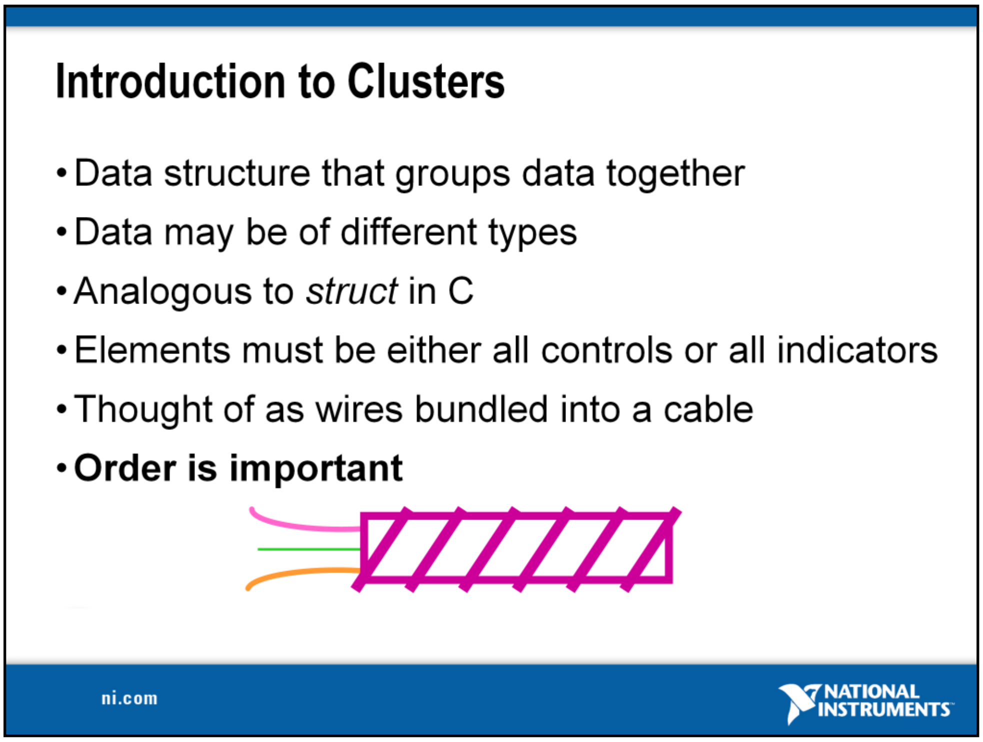
Section IV – Advanced Data Flow Topics (optional)

A. Additional Data types
- Cluster

B. Data Flow Constructs
- Shift Register
- Local Variables

C. Large Application Development
- Navigator Window
- LabVIEW Projects

ni.com



Introduction to Clusters

- Data structure that groups data together
- Data may be of different types
- Analogous to *struct* in C
- Elements must be either all controls or all indicators
- Thought of as wires bundled into a cable
- **Order is important**

ni.com

NATIONAL INSTRUMENTS

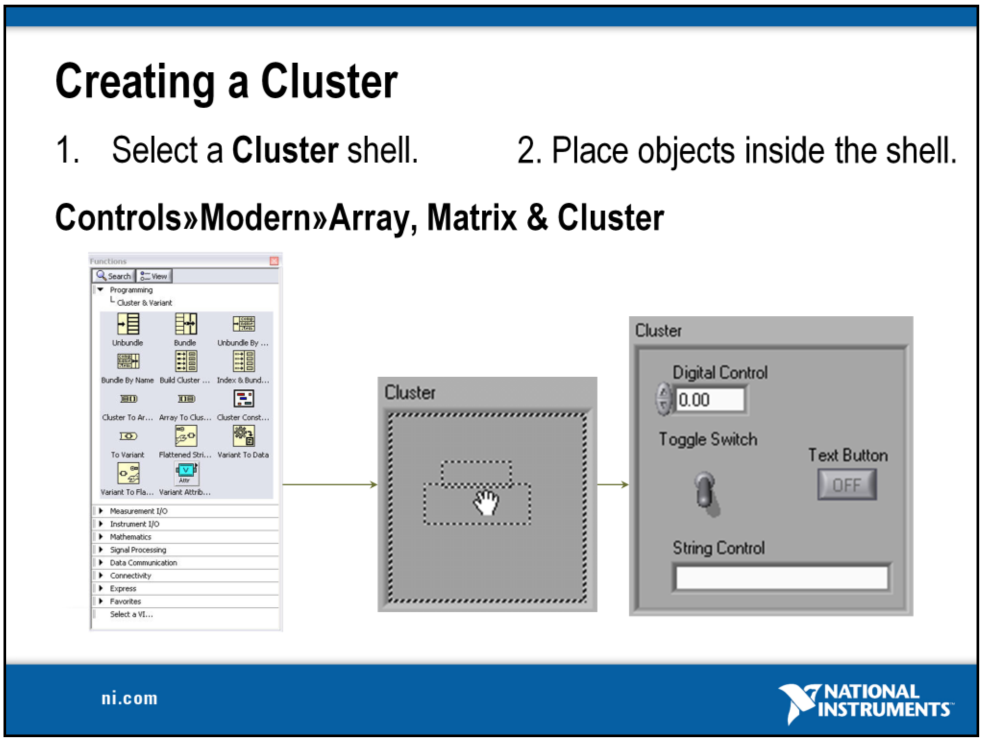
Clusters group like or unlike components together. They are equivalent to a *record* in Pascal or a *struct* in C.

Cluster components may be of different data types.

Examples:

- Error information—Grouping a Boolean error flag, a numeric error code, and an error source string to specify the exact error.
- User information—Grouping a string indicating a user's name and an ID number specifying their security code.

All elements of a cluster must be either controls or indicators. You cannot have a string control and a Boolean indicator. Clusters can be thought of as grouping individual wires (data objects) together into a cable (cluster).



Creating a Cluster

1. Select a **Cluster** shell. 2. Place objects inside the shell.

Controls»Modern»Array, Matrix & Cluster

ni.com

Cluster front panel object can be created by choosing **Cluster** from the **Controls»Modern»Array, Matrix & Cluster** palette.

- This option gives you a shell (similar to the array shell when creating arrays).
- You can size the cluster shell when you drop it.
- Right-click inside the shell and add objects of any type.

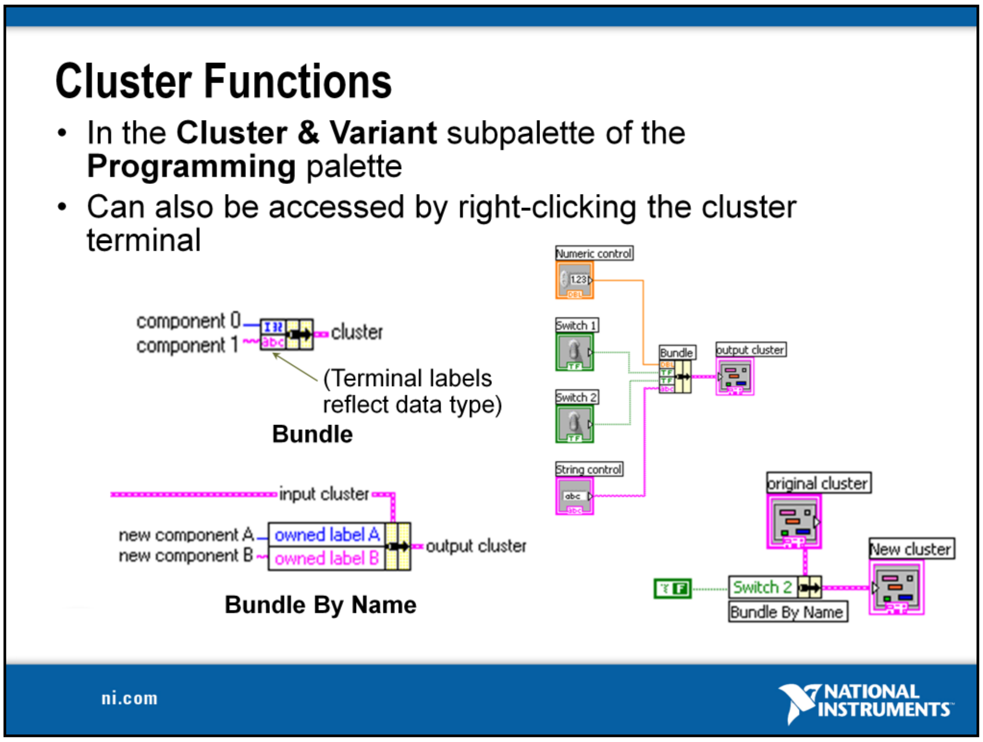
Note: You can even have a cluster inside of a cluster.

The cluster becomes a control or an indicator cluster based on the first object you place inside the cluster.

You can also create a cluster constant on the block diagram by choosing **Cluster Constant** from the **Cluster** palette.

- This gives you an empty cluster shell.
- You can size the cluster when you drop it.
- Put other constants inside the shell.

Note: You cannot place terminals for front panel objects in a cluster constant on the block diagram, nor can you place "special" constants like the Tab or Empty String constant within a block diagram cluster shell.



Cluster Functions

- In the **Cluster & Variant** subpalette of the **Programming** palette
- Can also be accessed by right-clicking the cluster terminal

component 0
component 1
cluster
(Terminal labels reflect data type)
Bundle
input cluster
new component A
new component B
owned label A
owned label B
output cluster
Bundle By Name
Numeric control
Switch 1
Switch 2
String control
Bundle
output cluster
original cluster
New cluster
Switch 2
Bundle By Name
ni.com
NATIONAL INSTRUMENTS

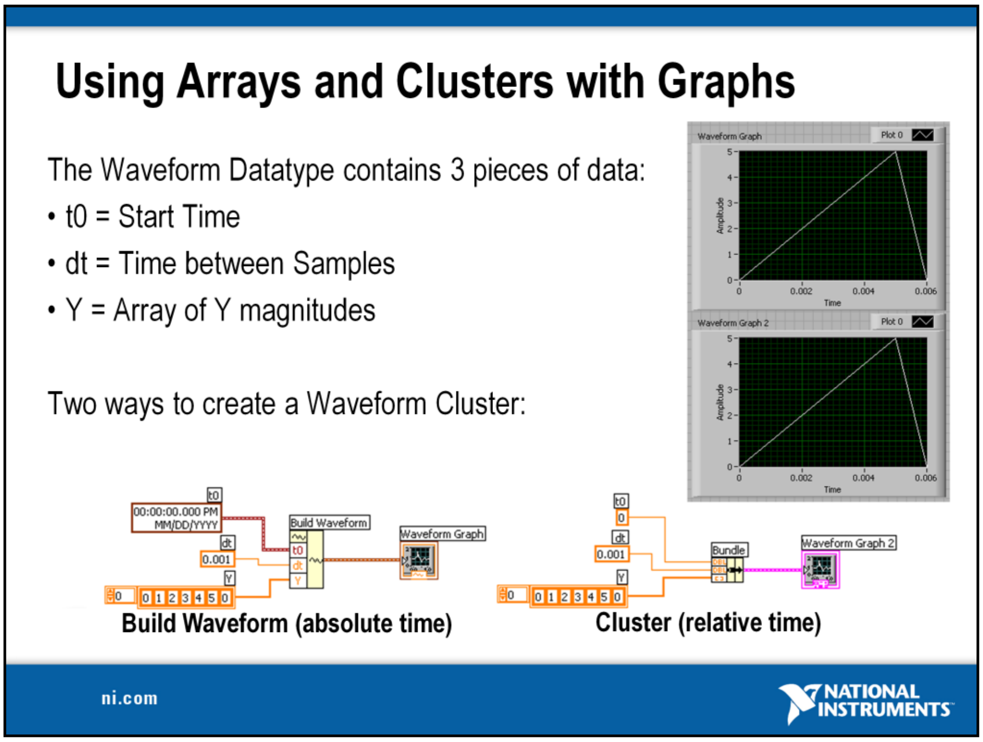
The terms Bundle and Cluster are closely related in LabVIEW.

Example: You use a Bundle Function to create a Cluster. You use an Unbundle function to extract the parts of a cluster.

Bundle function—Forms a cluster containing the given objects (explain the example).

Bundle by Name function—Updates specific cluster object values (the object must have an owned label).

Note: You must have an existing cluster wired into the middle terminal of the function to use Bundle By Name.



Using Arrays and Clusters with Graphs

The Waveform Datatype contains 3 pieces of data:

- t0 = Start Time
- dt = Time between Samples
- Y = Array of Y magnitudes

Two ways to create a Waveform Cluster:

Waveform Graph
Plot 0
Amplitude
Time
Waveform Graph 2
t0
00:00:00.000 PM MM/DD/YYYY
dt
0.001
Y
0 1 2 3 4 5 0
Build Waveform
Bundle

Build Waveform (absolute time) **Cluster (relative time)**

ni.com

NATIONAL INSTRUMENTS

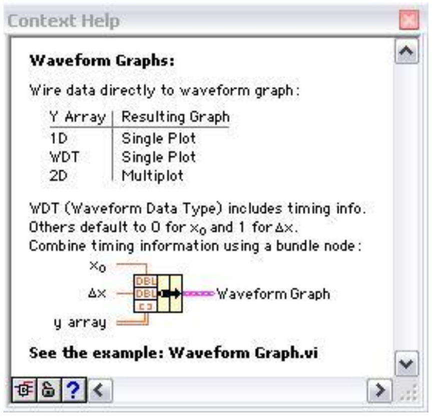
The waveform data type carries the data, start time, and Δt of a waveform. You can create waveforms using the Build Waveform function. Many of the VIs and functions you use to acquire or analyze waveforms accept and return the waveform data type by default. When you wire a waveform data type to a waveform graph or chart, the graph or chart automatically plots a waveform based on the data, start time, and Δx of the waveform. When you wire an array of waveform data types to a waveform graph or chart, the graph or chart automatically plots all the waveforms.

Build Waveform

Builds a waveform or modifies an existing waveform with the start time represented as an absolute TimeStamp. Time Stamps are accurate to real-world time & date and are very useful for real-world data recording.

Bundle

Builds a waveform or modifies an existing waveform with a relative time stamp. The input to t_0 is a DBL. Building waveforms using the bundle allows data to be plotted on the negative X (time) axis.



Context Help

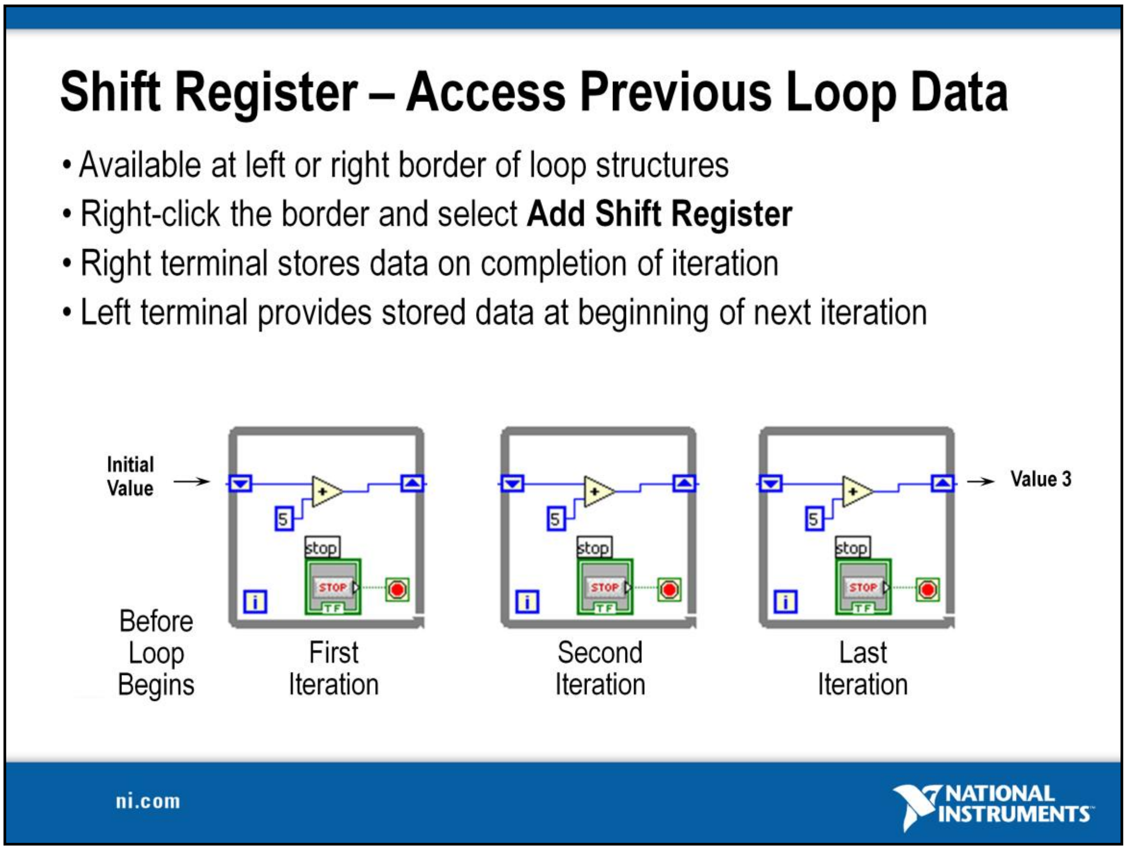
Waveform Graphs:

Wire data directly to waveform graph:

Y Array	Resulting Graph
1D	Single Plot
WDT	Single Plot
2D	Multiplot

WDT (Waveform Data Type) includes timing info. Others default to 0 for x_0 and 1 for Δx. Combine timing information using a bundle node:

See the example: Waveform Graph.vi



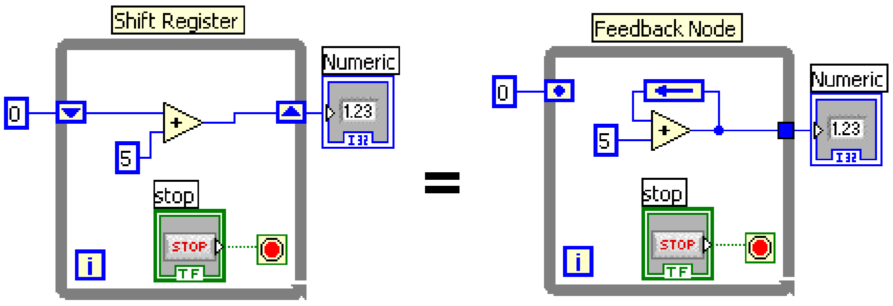
Shift Register – Access Previous Loop Data

- Available at left or right border of loop structures
- Right-click the border and select **Add Shift Register**
- Right terminal stores data on completion of iteration
- Left terminal provides stored data at beginning of next iteration

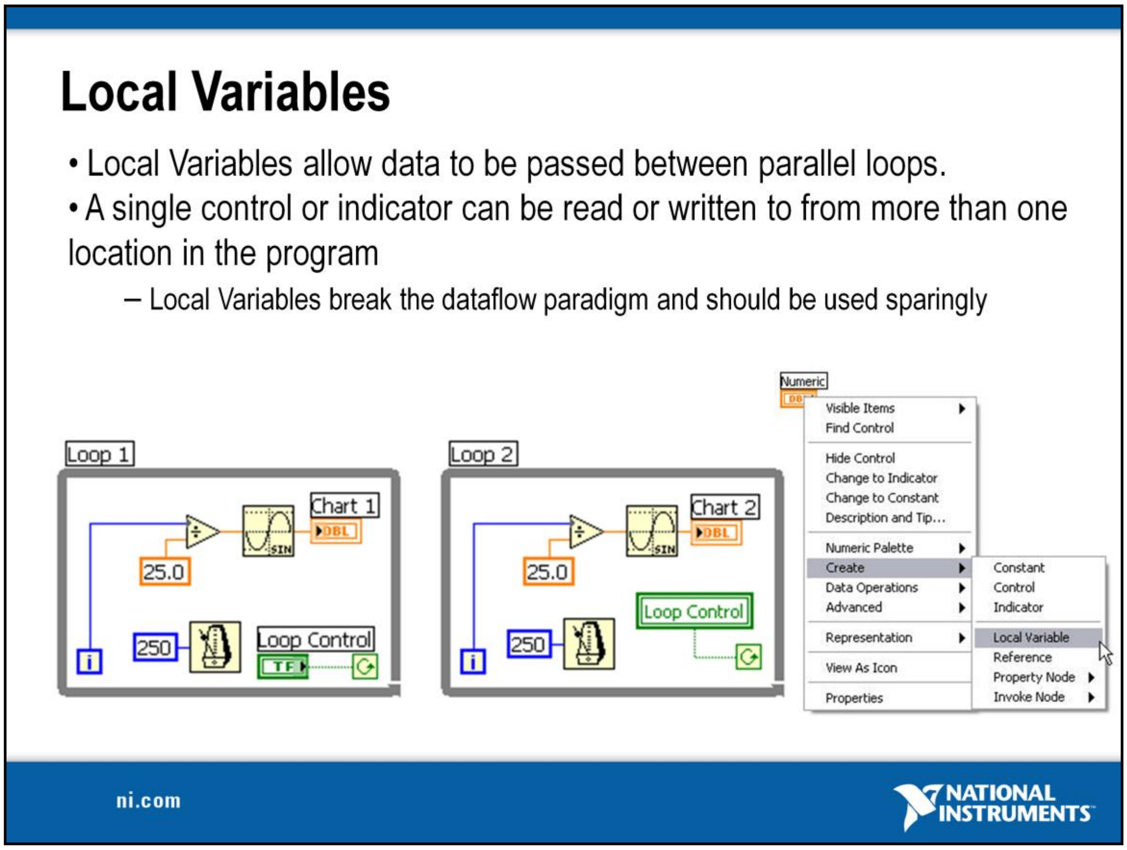
Shift registers transfer data from one iteration to the next:

- Right-click on the left or right side of a For Loop or a While Loop and select Add Shift Register.
- The right terminal stores data at the end of an iteration. Data appears at the left terminal at the start of the next iteration.
- A shift register adapts to any data type wired into it.

An input of 0 would result in an output of 5 the first iteration, 10 the second iteration and 15 the third iteration. Said another way, shift registers are used to retain values from one iteration to the next. They are valuable for many applications that have memory or feedback between states. The feedback node is another representation of the same concept. (pictured below) Both programs pictured behave the same.



See **Help»Search the LabVIEW Help…** for more information.



Local Variables

- Local Variables allow data to be passed between parallel loops.
- A single control or indicator can be read or written to from more than one location in the program
  - Local Variables break the dataflow paradigm and should be used sparingly

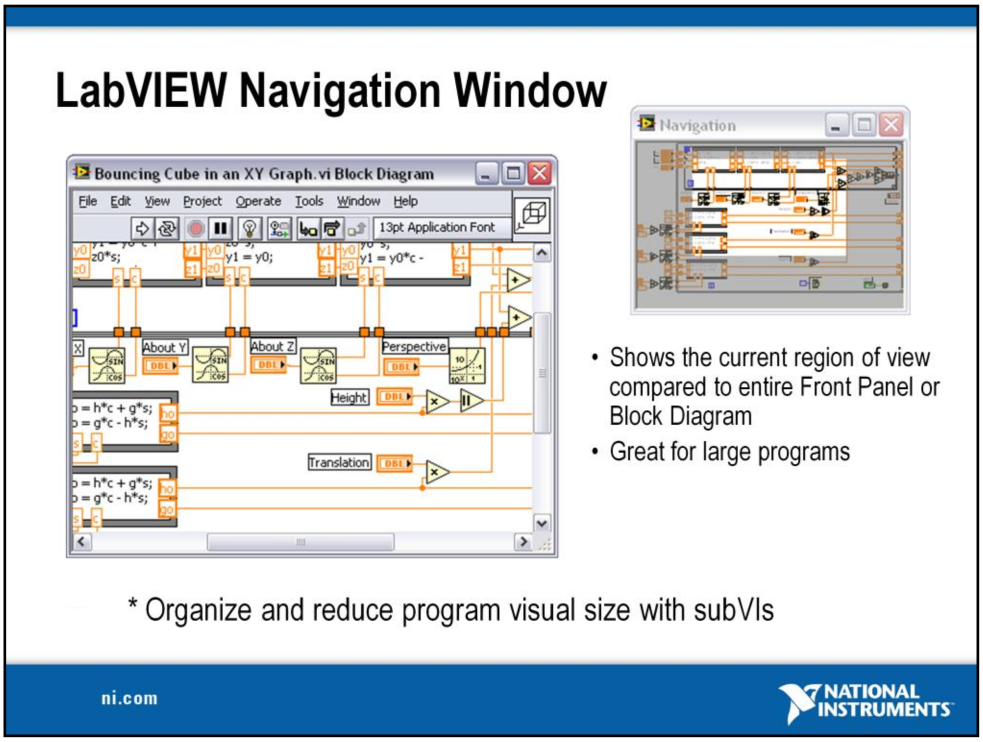
Sometimes you may need to access a front panel object from more than one place on the block diagram or to pass data between structures that cannot be connected by a wire. To accomplish these tasks, you would use a *local variable*.

Local variables are located in the **Structures** subpalette of the **Functions** palette.

When you place a local variable on the diagram, it contains by default the name (owned label) of the first object you placed on the front panel.

You use a local variable by first selecting the object you want to access. You can either click on the local variable with the Operating tool and select the object (by owned label) you want to access, or pop up on the local variable and choose the object from the **Select Item** menu.

Next, you must decide to either read or write to the object. Right click on the local variable and choose **Change To Read** or **Change to Write**.



LabVIEW Navigation Window

Bouncing Cube in an XY Graph.vi Block Diagram
File Edit View Project Operate Tools Window Help
13pt Application Font
About Y
About Z
Perspective
Height
Translation
Navigation

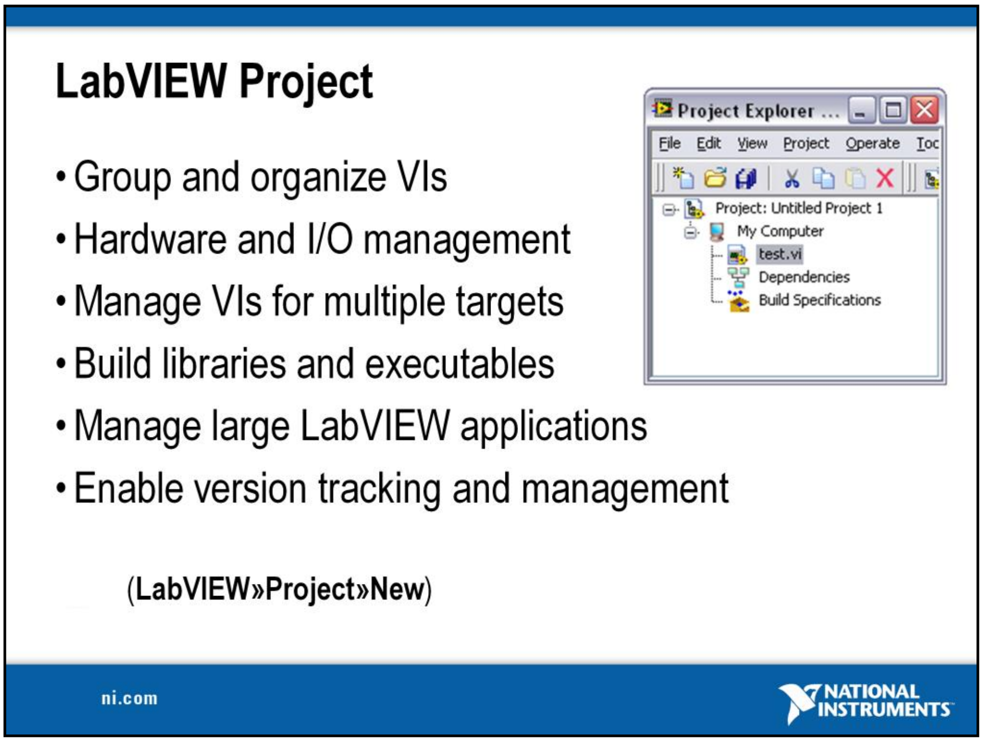
- Shows the current region of view compared to entire Front Panel or Block Diagram
- Great for large programs

* Organize and reduce program visual size with subVIs

ni.com

Select **View»Show Navigation Window** to display this window.

Use the window to navigate large front panels or block diagrams. Click an area of the image in the **Navigation Window** to display that area in the front panel or block diagram window. You also can click and drag the image in the **Navigation Window** to scroll through the front panel or block diagram.



LabVIEW Project

- Group and organize VIs
- Hardware and I/O management
- Manage VIs for multiple targets
- Build libraries and executables
- Manage large LabVIEW applications
- Enable version tracking and management

(**LabVIEW»Project»New**)

Project Explorer ...
File Edit View Project Operate Too
Project: Untitled Project 1
My Computer
test.vi
Dependencies
Build Specifications
ni.com
NATIONAL INSTRUMENTS

LabVIEW Project

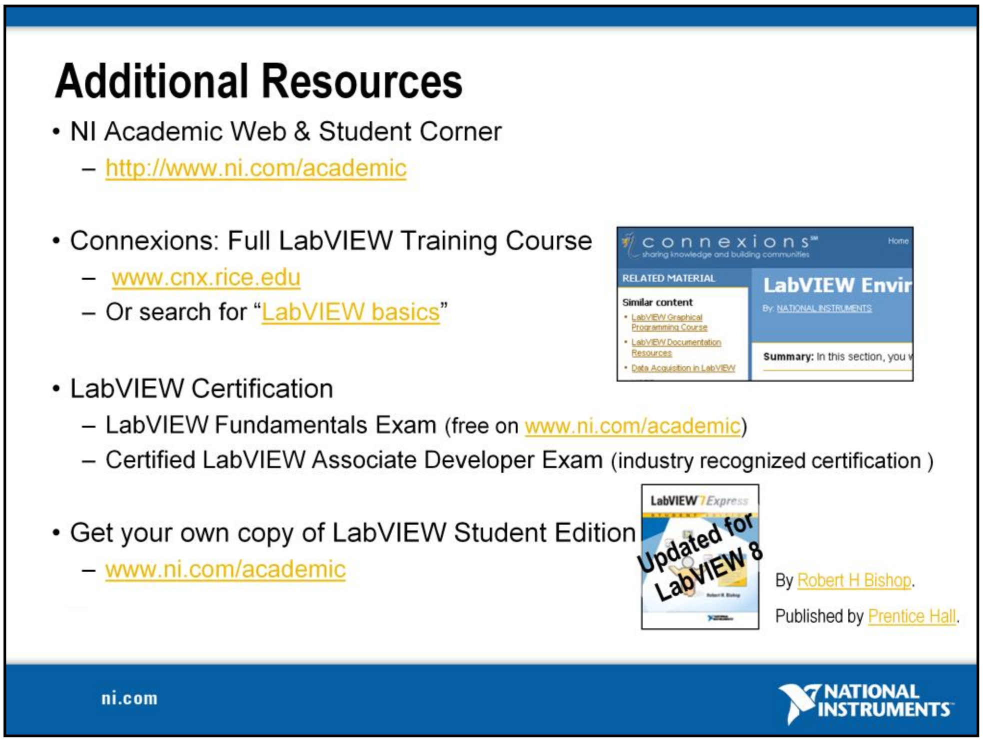
Use projects to group together LabVIEW files and non-LabVIEW files, create build specifications, and deploy or download files to targets. A target is a device or machine on which a VI runs. When you save a project, LabVIEW creates a project file (.lvproj), which includes configuration information, build information, deployment information, references to files in the project, and so on.

You must use a project to build stand-alone applications and shared libraries. You also must use a project to work with an RT, FPGA, or PDA target. Refer to the specific module documentation for more information about using projects with the LabVIEW Real-Time, FPGA, and PDA Modules.

Project-style LabVIEW Plug and Play instrument drivers use the project and project library features in LabVIEW 8.0. You can use project-style drivers in the same way as previous LabVIEW Plug and Play drivers.

Project Explorer Window

Use the Project Explorer window to create and edit projects. Select **File»New Project** to display the Project Explorer window. You also can select **Project»New Project** or select **File»New** and then select **Empty Project** in the New dialog box to display the Project Explorer window.



Additional Resources

- NI Academic Web & Student Corner
  - http://www.ni.com/academic
- Connexions: Full LabVIEW Training Course
  - www.cnx.rice.edu
  - Or search for "LabVIEW basics"
- LabVIEW Certification
  - LabVIEW Fundamentals Exam (free on www.ni.com/academic)
  - Certified LabVIEW Associate Developer Exam (industry recognized certification)
- Get your own copy of LabVIEW Student Edition
  - www.ni.com/academic

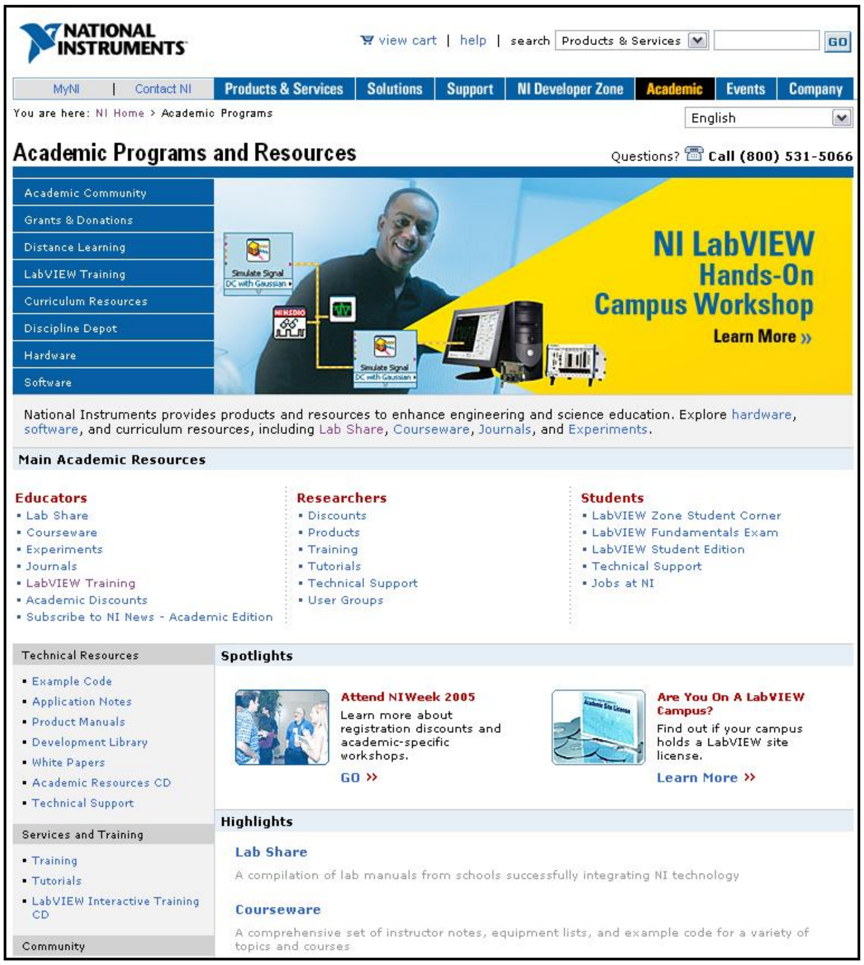
connexions
sharing knowledge and building communities
Home
RELATED MATERIAL
Similar content
LabVIEW Graphical Programming Course
LabVIEW Documentation Resources
Data Acquisition in LabVIEW
LabVIEW Envir
By: NATIONAL INSTRUMENTS
Summary: In this section, you
LabVIEW 7 Express
Updated for LabVIEW 8

By Robert H Bishop.

Published by Prentice Hall.

ni.com

NATIONAL INSTRUMENTS



You are here: NI Home > Academic Programs

Academic Programs and Resources

Questions? Call (800) 531-5066

National Instruments provides products and resources to enhance engineering and science education. Explore hardware, software, and curriculum resources, including Lab Share, Courseware, Journals, and Experiments.

Main Academic Resources

Educators
- Lab Share
- Courseware
- Experiments
- Journals
- LabVIEW Training
- Academic Discounts
- Subscribe to NI News - Academic Edition

Researchers
- Discounts
- Products
- Training
- Tutorials
- Technical Support
- User Groups

Students
- LabVIEW Zone Student Corner
- LabVIEW Fundamentals Exam
- LabVIEW Student Edition
- Technical Support
- Jobs at NI

Technical Resources
- Example Code
- Application Notes
- Product Manuals
- Development Library
- White Papers
- Academic Resources CD
- Technical Support

Services and Training
- Training
- Tutorials
- LabVIEW Interactive Training CD

Community

Spotlights

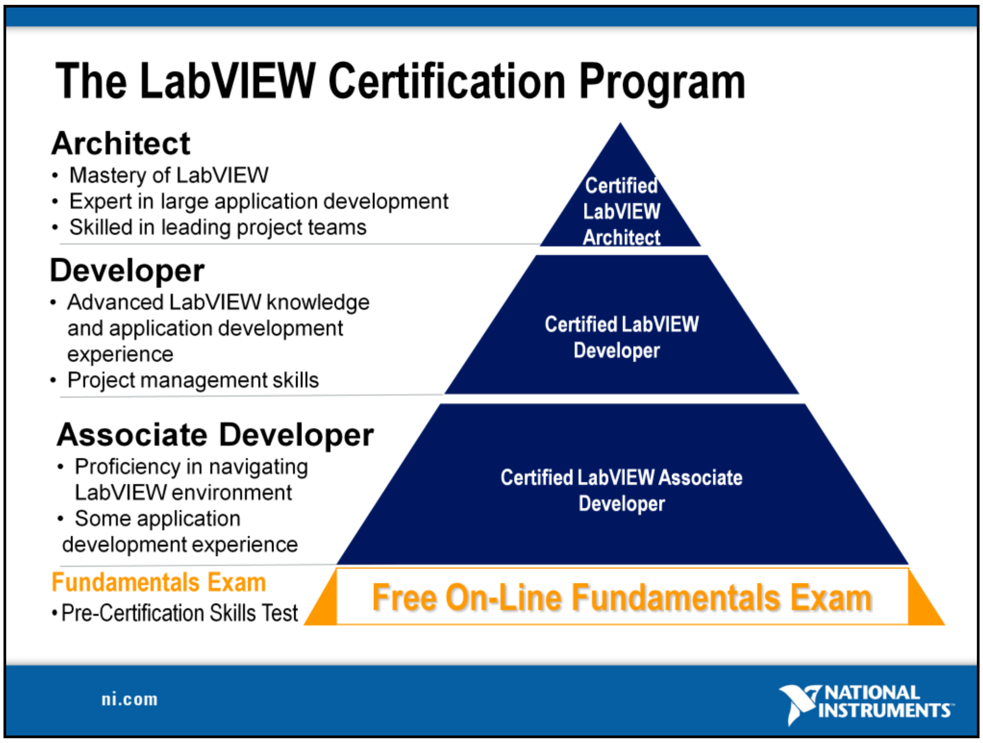
Attend NIWeek 2005
Learn more about registration discounts and academic-specific workshops.
GO »

Are You On A LabVIEW Campus?
Find out if your campus holds a LabVIEW site license.
Learn More »

Highlights

Lab Share
A compilation of lab manuals from schools successfully integrating NI technology

Courseware
A comprehensive set of instructor notes, equipment lists, and example code for a variety of topics and courses



The LabVIEW Certification Program

Architect
- Mastery of LabVIEW
- Expert in large application development
- Skilled in leading project teams

Developer
- Advanced LabVIEW knowledge and application development experience
- Project management skills

Associate Developer
- Proficiency in navigating LabVIEW environment
- Some application development experience

Fundamentals Exam
- Pre-Certification Skills Test

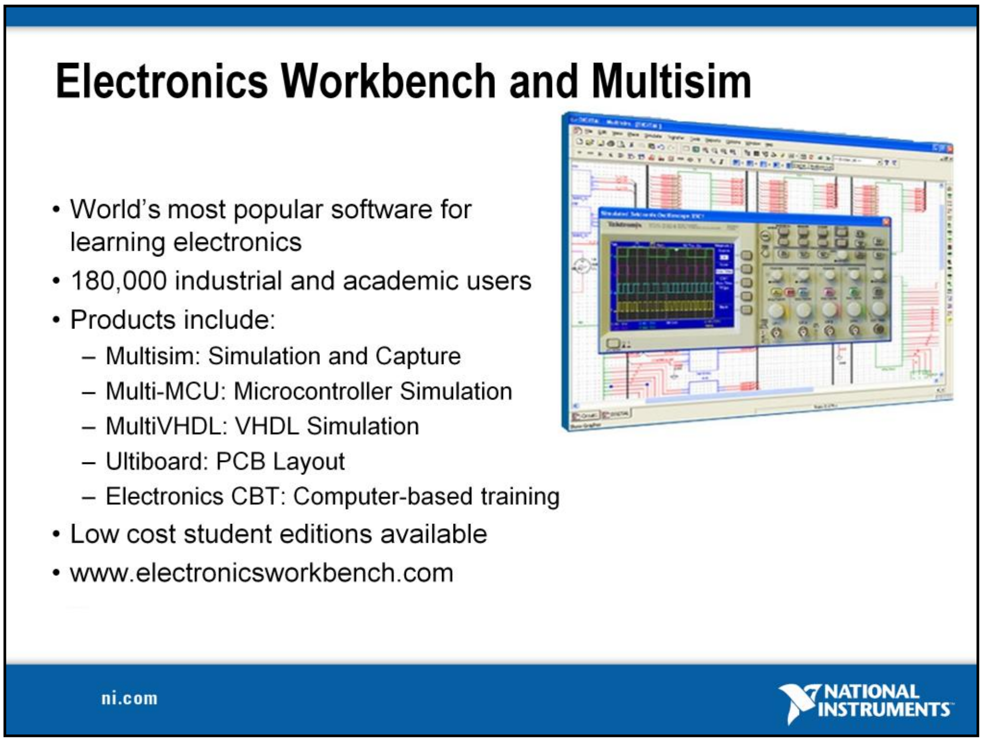
Certified LabVIEW Architect
Certified LabVIEW Developer
Certified LabVIEW Associate Developer
Free On-Line Fundamentals Exam
ni.com
NATIONAL INSTRUMENTS

Today, more and more companies and hiring managers are requesting for LabVIEW expertise in their job interviews. The LabVIEW Certification Program is built on a series of professional exams. LabVIEW Certifications are used to validate LabVIEW expertise and skills for employment opportunities and for project bids.

The Certified LabVIEW Associate Developer is the first-step for LabVIEW certification and it demonstrates a strong foundation in using LabVIEW and the LabVIEW environment. As students, your Certified LabVIEW Associate Developer certification differentiates your LabVIEW skill for employment opportunities and also gets you recognition for your LabVIEW expertise. The CLAD is a 1-hour multiple choice exam conducted at Pearson VUE testing centers around the country. The exam covers multiple topics on the LabVIEW environment including dataflow concepts, programming structures, Advanced file I/O techniques, Modular programming practices, VI object properties and control references.

Thinking about getting your CLAD certification? Take the free online LabVIEW Fundamentals Exam as a sample test.

The Certified LabVIEW Developer and Architect are professional certifications that validate Advanced LabVIEW knowledge and application development experience. Additionally, the Architect certification also demonstrates skills in leading project teams and large application development experience. These exams are 4-hour practical exams conducted by National Instruments.



Electronics Workbench and Multisim

- World's most popular software for learning electronics
- 180,000 industrial and academic users
- Products include:
  - Multisim: Simulation and Capture
  - Multi-MCU: Microcontroller Simulation
  - MultiVHDL: VHDL Simulation
  - Ultiboard: PCB Layout
  - Electronics CBT: Computer-based training
- Low cost student editions available
- www.electronicsworkbench.com

ni.com

NATIONAL INSTRUMENTS

Electronics Workbench products are the most widely used electronics software in electrical engineering and electronics technology departments around the globe. As the only company to design our products specifically for the education market, our software has become the teaching and learning tool of choice for thousands of educators.

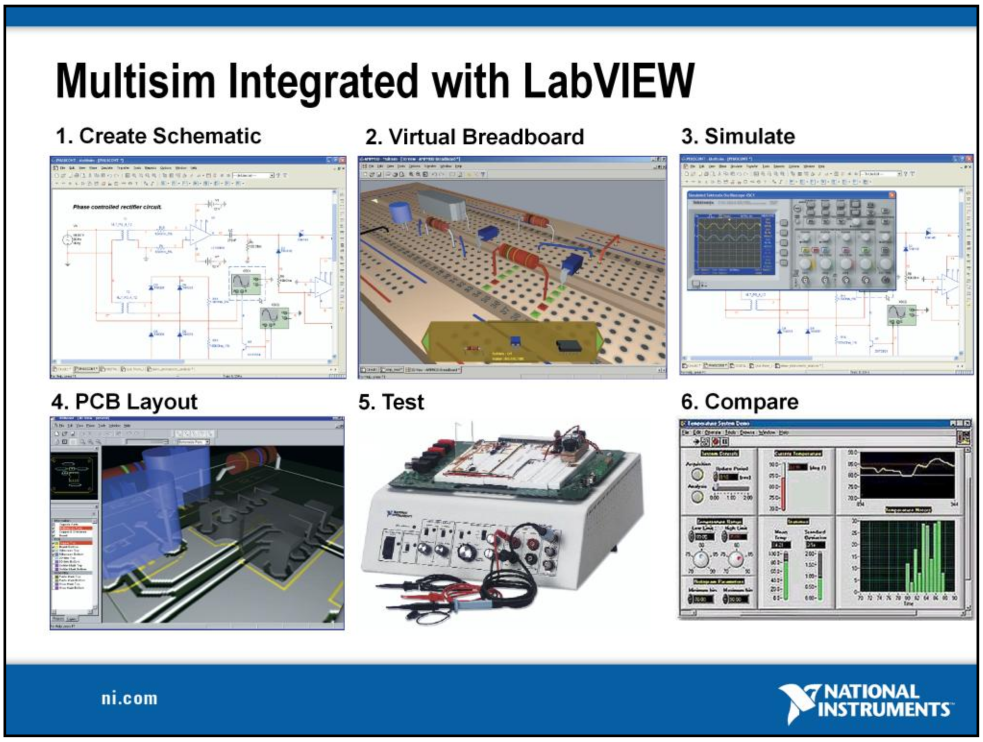
MULTISIM — SIMULATION AND CAPTURE
Multisim is an intuitive, drag-and-drop schematic capture and simulation program that allows educators and students to quickly create complete circuits containing both analog and digital components.

MULTIMCU— MICROCONTROLLER CO-SIMULATION
MultiMCU adds microcontroller unit co-simulation capabilities to Multisim, allowing you to include an MCU, programmed in assembly code, within your SPICE (and optionally VHDL) modeled circuit.

MULTIVHDL — VHDL CO-SIMULATION
MultiVHDL adds patented VHDL co-simulation capabilities to Multisim. It is a powerful yet easy-to-use application that is perfect for teaching students about HDL programming, or for including VHDL-programmed devices in a Multisim project.

ULTIBOARD — PCB LAYOUT
Ultiboard allows students to gain exposure to the physical implementation and manufacturing of circuits on PCBs. Their Multisim schematic can be imported into Ultiboard with a single mouse-click.

ELECTRONICS CBT — COMPUTER-BASED TRAINING
Electronics CBT offers a complete, standalone introductory electronics curriculum to support your lectures or to act as the centerpiece of your course delivery. E-CBT is enhanced with over 400 exercises and experiments that run directly in Multisim's virtual lab environment.

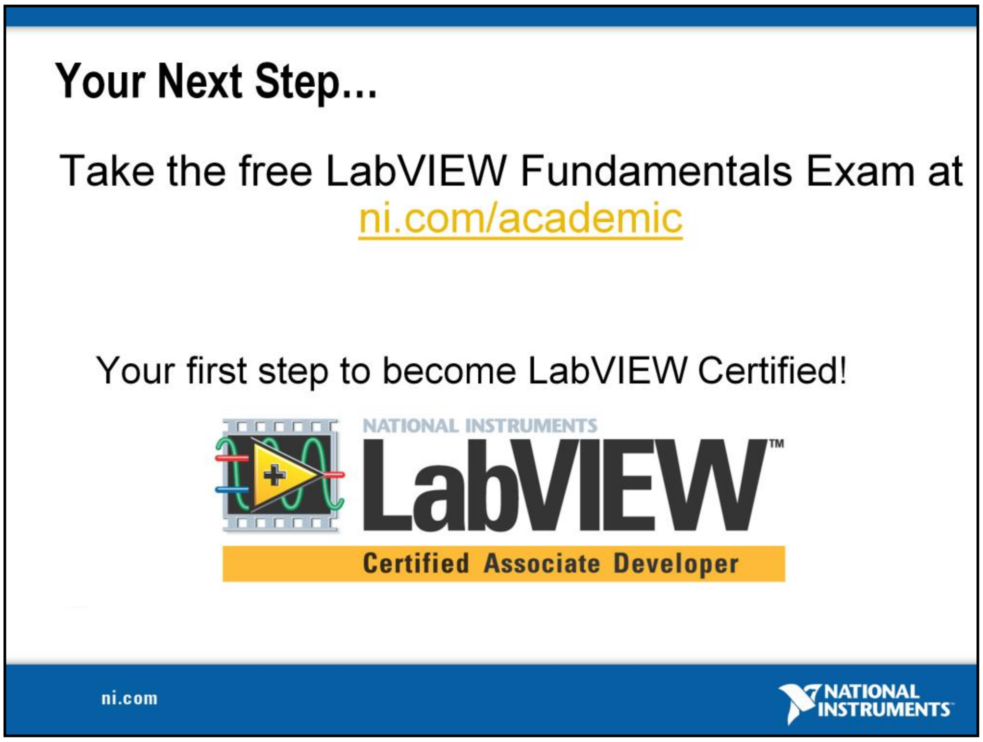


Multisim Integrated with LabVIEW

1. Create Schematic
2. Virtual Breadboard
3. Simulate
4. PCB Layout
5. Test
6. Compare

ni.com

1. Multisim - Schematics
    - Easy-to-use schematics
    - Simply click and drag
    - 3D animated parts
    - Wire drag without breaking connections
2. Multisim – Virtual breadboard
    - Breadboarding techniques
    - Synchronized with schematic
    - Wiring report for Elvis (step 5)
3. Multisim – Simulation
    - 13.000 part library
    - 20 virtual instruments
    - Changes on-the-fly
    - New microcontroller simulation
    - Animated parts (LEDs, and 7-segment displays)
4. Ultiboard – PCB Layout
    - Integrated with Multisim
    - User-friendly interface
    - 3D view
    - Design rule check
    - Built-in autorouting
5. Elvis – Test
    - Instrumentation
    - Data acquisition
    - Prototyping
6. LabVIEW – Compare
    - Automatically import:
        - Multisim virtual data
        - Elvis real data
    - Compare ideal and real data



Your Next Step…

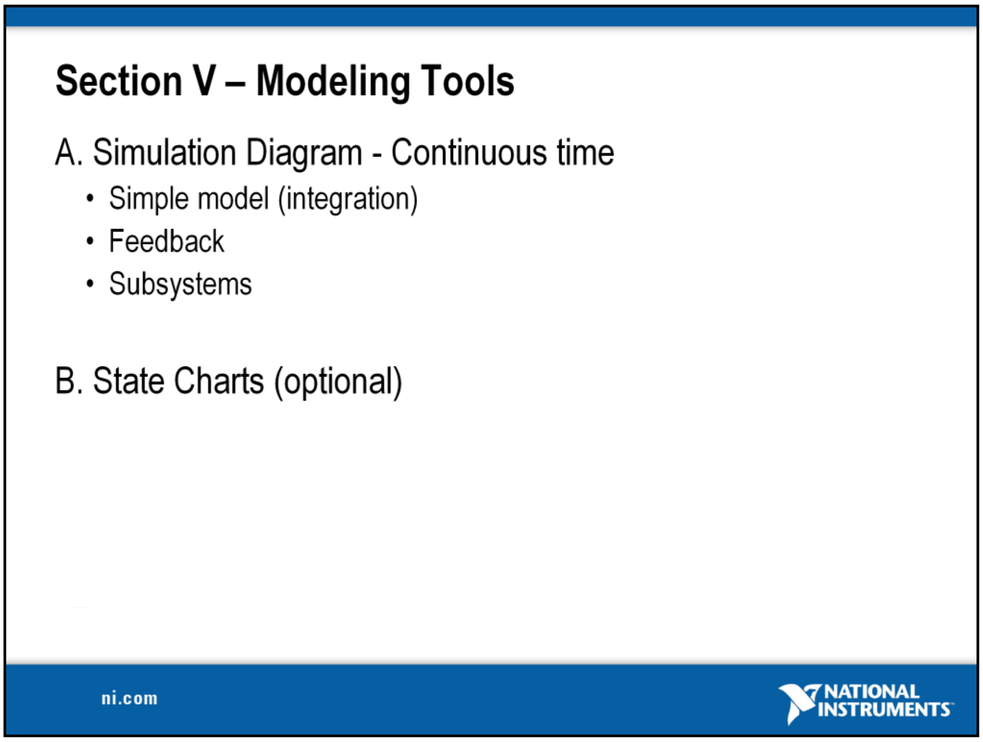
Take the free LabVIEW Fundamentals Exam at
ni.com/academic

Your first step to become LabVIEW Certified!

NATIONAL INSTRUMENTS LabVIEW™ Certified Associate Developer

ni.com

NATIONAL INSTRUMENTS



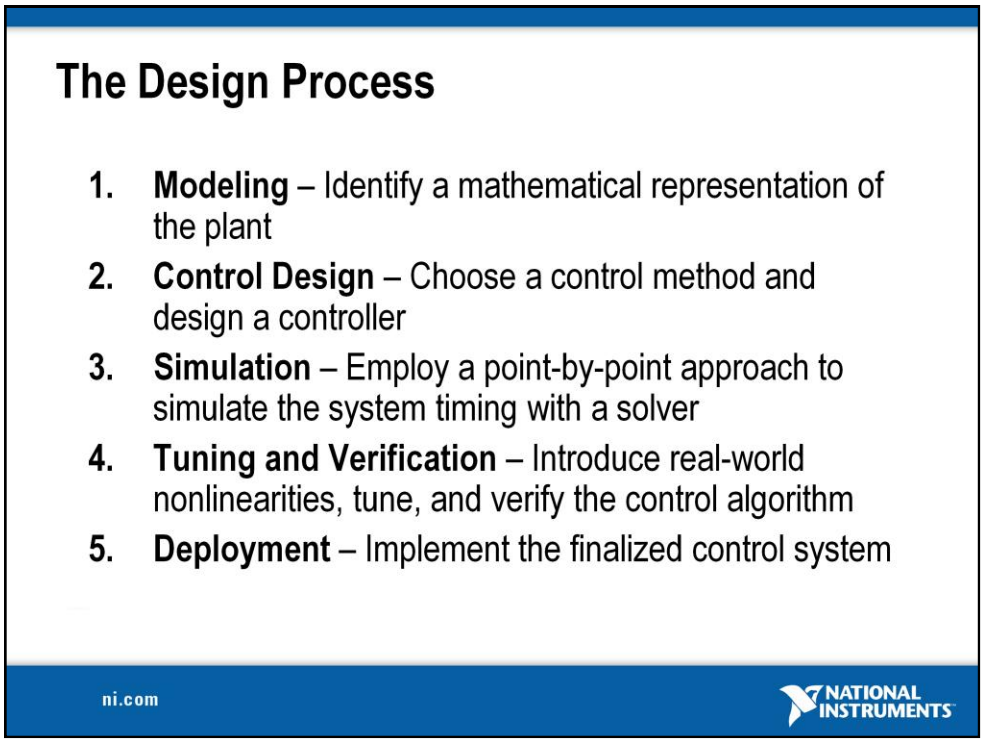
Section V – Modeling Tools

A. Simulation Diagram - Continuous time

- Simple model (integration)
- Feedback
- Subsystems

B. State Charts (optional)

ni.com



The Design Process

1. **Modeling** – Identify a mathematical representation of the plant
2. **Control Design** – Choose a control method and design a controller
3. **Simulation** – Employ a point-by-point approach to simulate the system timing with a solver
4. **Tuning and Verification** – Introduce real-world nonlinearities, tune, and verify the control algorithm
5. **Deployment** – Implement the finalized control system

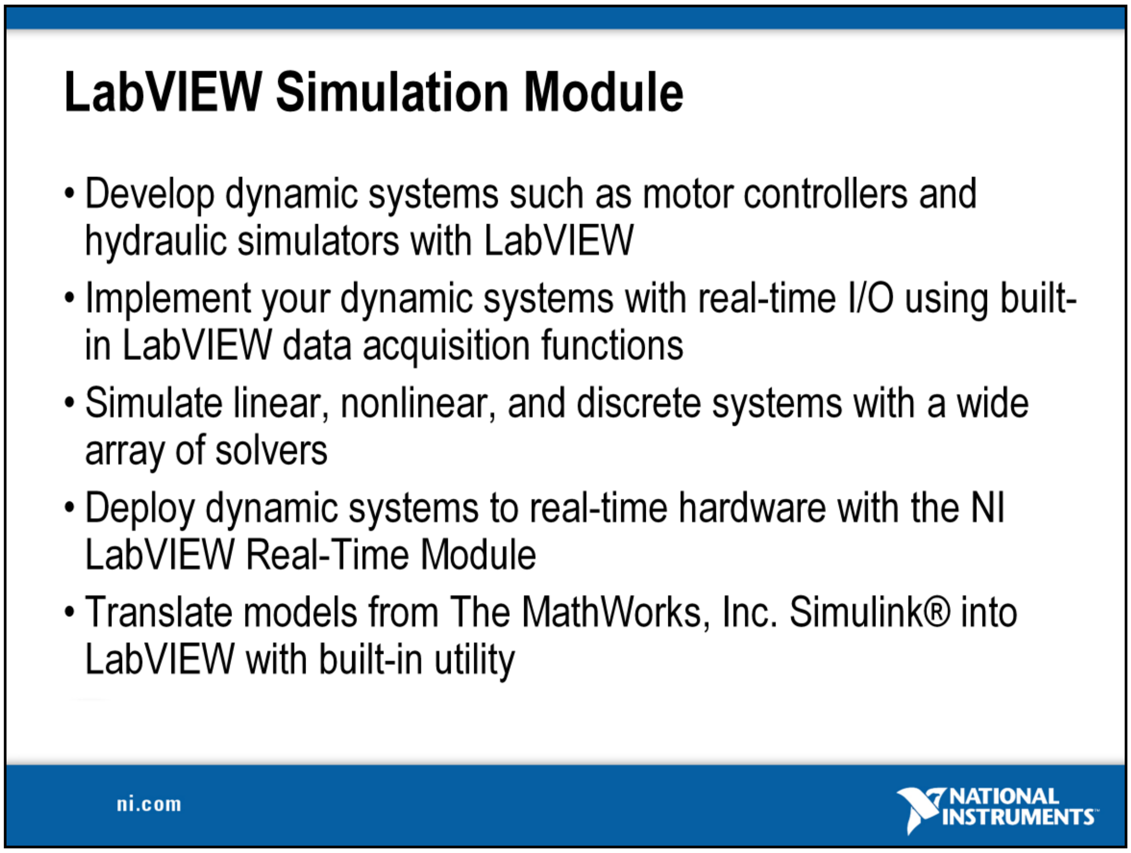
ni.com

NATIONAL INSTRUMENTS

The controller can be optimized by fully understanding the plant. This understanding comes from analysis done in the LabVIEW Control Design Toolkit. Specialized graphs, such as Bode, root-locus, and Nyquist plot, builds intuition of how the plant will behave. Further analysis using these plots with an open loop system with the controller allow stability and other characteristics to be calculated. In this way, we use Control Design to optimize the parameters, in our case the gains P, I, and D. Graphs in the time domain, such as the step response, provide immediate feedback on the ideal behavior of the system, such as rise time, overshoot, settling time, and steady-state error.

We then use the Simulation Module to validate these parameters under real world constraints. One prime example is the maximum and minimum voltage and current available to be supplied to the motor. The voltage limits of the motor define the linear range of operation. Outside of that range, the motor might be damaged or become unpredictable. This is an example of nonlinear behavior, which cannot be modeled using LTI analysis found in the Control Design Toolkit. The Simulation Module allows nonlinearities to be represented in the model. Voltage limits can be modeled using the Saturation block. Other nonlinear behaviors can be written in LabVIEW, MathScript, and the LabVIEW Formula Node, and integrated into the model as a Sub VI.

The optimization process usually includes several iterations between Control Design and Simulation to achieve the desired performance of the system.



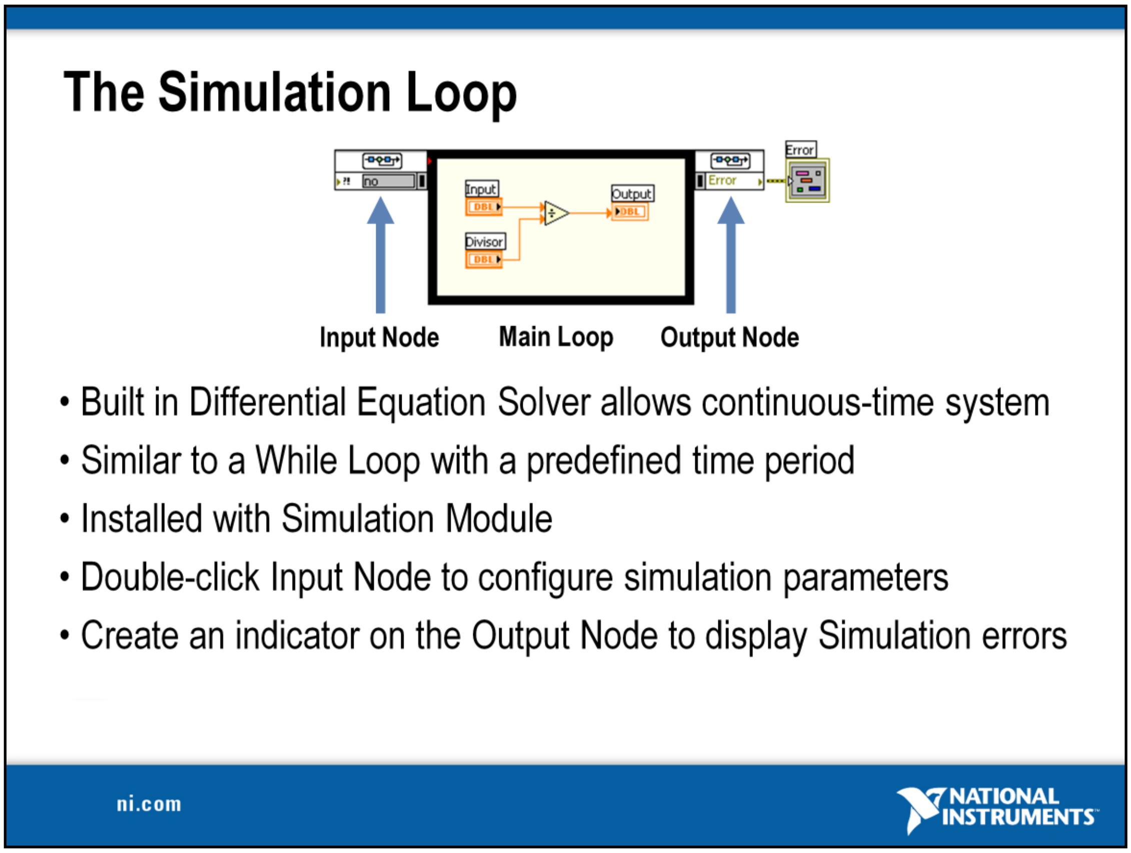
LabVIEW Simulation Module

- Develop dynamic systems such as motor controllers and hydraulic simulators with LabVIEW
- Implement your dynamic systems with real-time I/O using built-in LabVIEW data acquisition functions
- Simulate linear, nonlinear, and discrete systems with a wide array of solvers
- Deploy dynamic systems to real-time hardware with the NI LabVIEW Real-Time Module
- Translate models from The MathWorks, Inc. Simulink® into LabVIEW with built-in utility

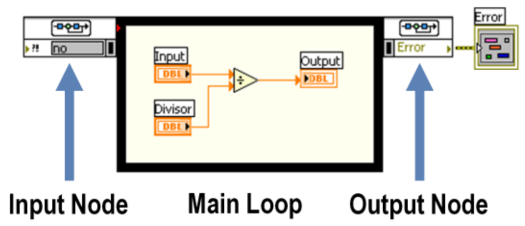
ni.com

NATIONAL INSTRUMENTS

The National Instruments LabVIEW Simulation Module integrates dynamic system simulation into the LabVIEW environment. As a module, the Simulation Module adds new functionality to core LabVIEW. The simulation loop can model linear, nonlinear, discrete, and continuous plant or control systems in block diagram form. Create models from blocks such as integrator, derivative, and transfer function blocks, and then add graphs and controls to test out the models. Alternatively, you can import models developed in the NI LabVIEW Control Design Toolkit or NI LabVIEW System Identification Toolkit. The interactive nature of the LabVIEW tools allow you to modify parameters while logging the results of the simulation. Models built with the simulation node can also seamlessly download to a real-time target with the LabVIEW Real-Time Module for control prototyping and hardware-in-the-loop (HIL) simulation.



The Simulation Loop



- Built in Differential Equation Solver allows continuous-time system
- Similar to a While Loop with a predefined time period
- Installed with Simulation Module
- Double-click Input Node to configure simulation parameters
- Create an indicator on the Output Node to display Simulation errors

ni.com

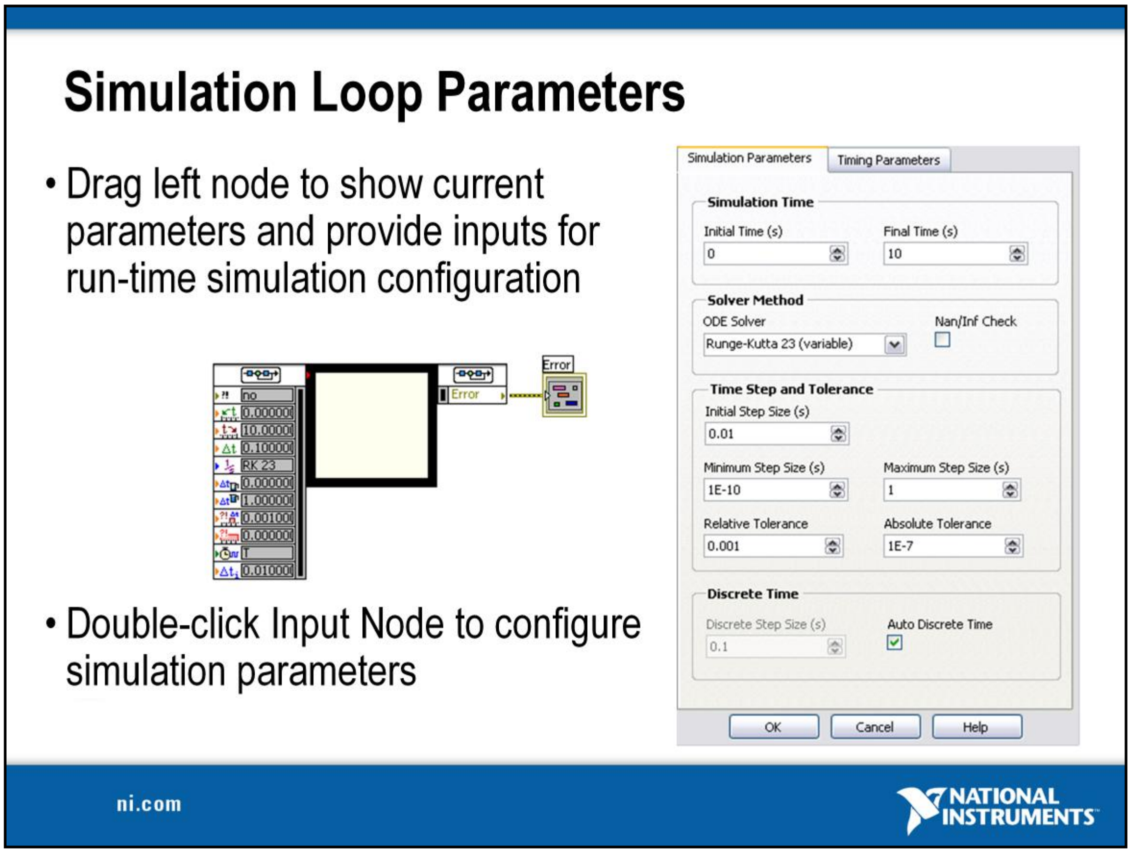
NATIONAL INSTRUMENTS

The simulation loop is the core component of the Simulation Module. It is an “upgraded” version of the while loop which integrates powerful differential equation solvers, internal or external timing features, and cross-platform capabilities. With the Simulation Module, continuous time systems can be run in the discrete digital world. The loop consists of three main parts:

Input node (Left)- allows simulation parameters to be programmatically defined. By default, these parameters are static and can be configured by double-clicking the input node. This node can be expanded by clicking on the bottom border of the node to “grab” the handle and dragging down to show additional parameters.

Main loop – The system to be simulated is placed here.

Output node (Right) – returns any errors that may have happened in the loop, such as an improper transfer function.



Simulation Loop Parameters

- Drag left node to show current parameters and provide inputs for run-time simulation configuration
- Double-click Input Node to configure simulation parameters

Simulation Parameters | Timing Parameters

Simulation Time
Initial Time (s): 0
Final Time (s): 10

Solver Method
ODE Solver: Runge-Kutta 23 (variable)
Nan/Inf Check: ☐

Time Step and Tolerance
Initial Step Size (s): 0.01
Minimum Step Size (s): 1E-10
Maximum Step Size (s): 1
Relative Tolerance: 0.001
Absolute Tolerance: 1E-7

Discrete Time
Discrete Step Size (s): 0.1
Auto Discrete Time: ✓

OK | Cancel | Help

ni.com

Configuring a Simulation

There are many parameters that can be configured for a given simulation loop.

Simulation Parameters Tab

Simulation Time – Specifies for what period of “simulation time” how long the simulation should run. This time doesn’t necessarily dictate the computation time of the simulation. See Timing Parameters below.

Solver Method – Specifies what Ordinary Differential Equation ODE) solver is used to solve integral and differential type blocks in the simulation. A wide variety of solvers are available.

Time Step and Tolerance – These settings control the window of time steps used by LabVIEW. Typically, the default settings will suffice, but adjust them if necessary.

Discrete Time – While the Default Auto Discrete Time option will typically work for most simulations, you can force LabVIEW to use a specific step size here.

Timing Parameters Tab

Timing – The software timing option is always used when performing simulations. This will solve the equation as fast as the CPU can. To observe a simulation run in “real time”, disable the software timing option and set the timing parameters below.

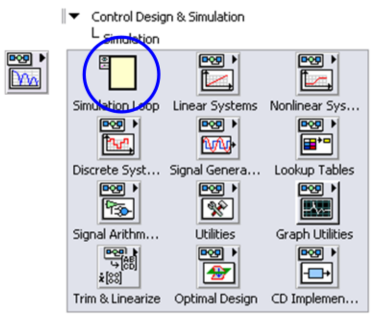
Loop Timing Source – When implementing a simulation in hardware, change the loop timing source to an available hardware timing. This option can be used to sync the simulation execution with an external time source, such as the operating system clock or even a Data Acquisition board’s clock.

Loop Timing Parameters – These options control how the loop executes with respect to the selected timing source.

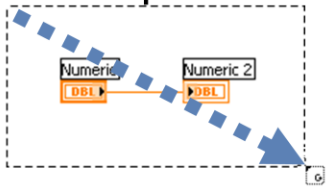
For more detailed information on these options, consult the LabVIEW help by clicking the Help button.

Drawing a Simulation Loop

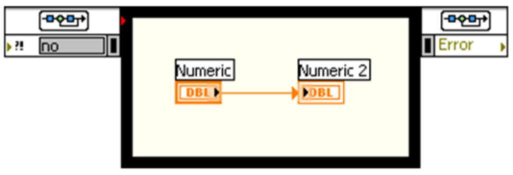
1. Select the structure from the Functions Palette



2. Left-click at the top left point



...and drag to the bottom right to enclose code to be looped



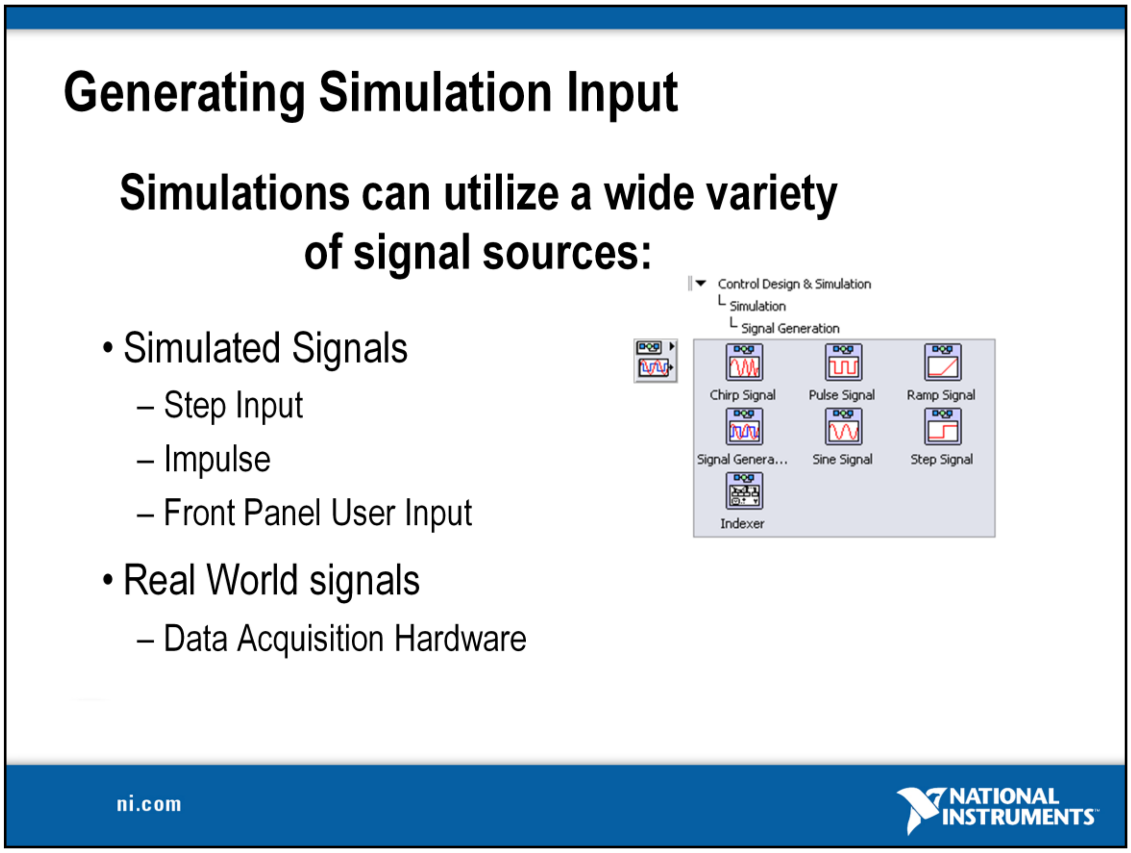
ni.com

Place simulation loops in your diagram by selecting them from the Simulation Palette in the Functions palette:

When selected, the mouse cursor becomes a special pointer that you use to enclose the section of code you want to repeat.

Click the mouse button to define the top-left corner, click the mouse button again at the bottom-right corner, and the Simulation Loop boundary is created around the selected code.

Drag or drop additional blocks or functions in the Simulation Loop if needed.



Generating Simulation Input

Simulations can utilize a wide variety of signal sources:

- Simulated Signals
  - Step Input
  - Impulse
  - Front Panel User Input
- Real World signals
  - Data Acquisition Hardware

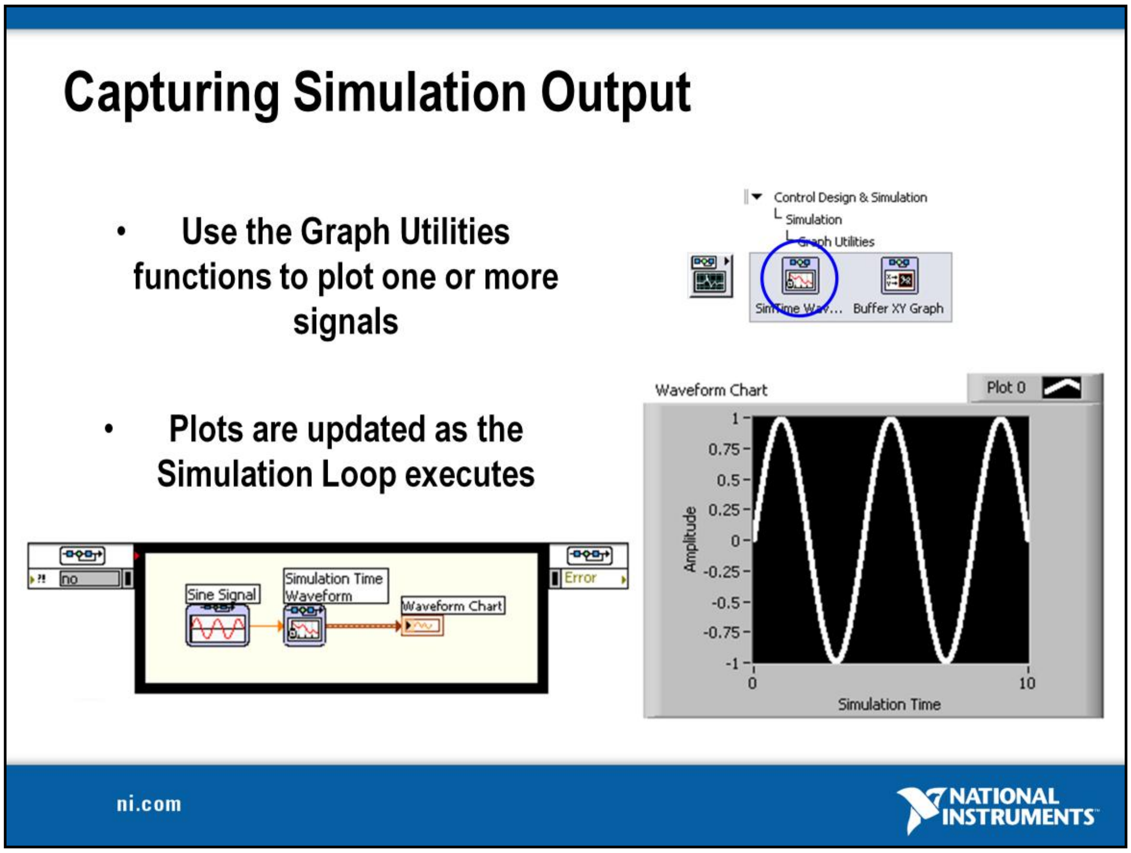
Control Design & Simulation
Simulation
Signal Generation
Chirp Signal
Pulse Signal
Ramp Signal
Signal Genera...
Sine Signal
Step Signal
Indexer

ni.com

NATIONAL INSTRUMENTS

Simulations can utilize a wide variety of signals sources. Simulated signals are useful for characterizing system response and testing corner cases. Chirps provide a useful frequency sweep, and pulses provide good step response information. The indexer is useful for using a predetermined arbitrary signal, and can interpolate between samples for better resolution.

Once the simulation is confirmed, a real world signal can be substituted for the simulation signal. In this configuration, the simulation is now performing calculations based on actual data and is very good for testing the system before controlling the output.



Capturing Simulation Output

- Use the Graph Utilities functions to plot one or more signals
- Plots are updated as the Simulation Loop executes

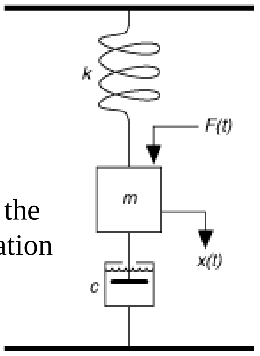
ni.com

NATIONAL INSTRUMENTS

The Simulation Time Waveform chart is a special numeric indicator that displays one or more plots. The waveform chart is located on the Functions » Control Design & Simulation » Simulation » Graph Utilities » SimTime Waveform palette. Waveform charts can display single or multiple plots. The following front panel shows an example of a multi-plot waveform chart.

You can change the min and max values of either the x or y axis by double clicking on the value with the labeling tool and typing the new value. Similarly, you can change the label of the axis. You can also right click the plot legend and change the style, shape, and color of the trace that is displayed on the chart.

Exercise 6 – Simulation of a dynamic system



Background Information

A dynamic system model is a differential or difference equation that describes the behavior of the dynamic system. In this tutorial, the following differential equation describes the dynamic system.

$F(t) - cx'(t) - kx(t) = mx''(t)$

where t is the simulation time, $F(t)$ is an external force applied to the system, c is the damping constant of the spring, k is the stiffness of the spring, m is a mass, and $x(t)$ is the position of the mass. x' is the first derivative of the position, which equals the velocity of the mass. x'' is the second derivative of the position, which equals the acceleration of the mass.

The following figure shows this dynamic system.

The goal of this tutorial is to use the LabVIEW Simulation Module to view the position $x(t)$ of the mass m with respect to time t. You can calculate the position by integrating the velocity of the mass. You can calculate the velocity by integrating the acceleration of the mass. If you know the force and mass, you can calculate this acceleration by using Newton's Second Law of Motion, given by the following equation:

Force = Mass × Acceleration

Therefore,

Acceleration = Force / Mass

Substituting terms from the differential equation above yields the following equation:

$x''(t) = (F(t) - cx'(t) - kx(t))/m$

In this tutorial you will construct a simulation diagram that iterates the following steps over a period of time.

Divides a known force by a known mass to calculate the acceleration of the mass.

Integrates acceleration to calculate the velocity of the mass.

Integrates velocity to calculate the position of the mass.

Performing these integrations over a period of time requires an ordinary differential equation (ODE) solver. LabVIEW includes several ODE solvers you can use in a simulation. Each ODE solver has properties that make it suitable for different tasks. For example, some ODE solvers have variable step sizes. During a simulation, these ODE solvers can adjust the points in time at which the Simulation Module evaluates the simulation diagram. Conversely, ODE solvers with fixed step sizes cannot make this change.

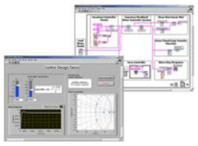
Complete the steps in the following sections to build a simulation diagram that simulates the spring-mass damper dynamic system model.

Please refer to separate file for details of lab

Tutorial Getting Started with the LabVIEW Simulation Module - LabVIEW 8_5 Simulation Module Help.pdf

Where Can I Learn More?

We have only begun to explore the many opportunities for control and simulation within LabVIEW. Learn more by visiting the following links:



System Identification Toolkit:
- http://sine.ni.com/nips/cds/view/p/lang/en/nid/13853

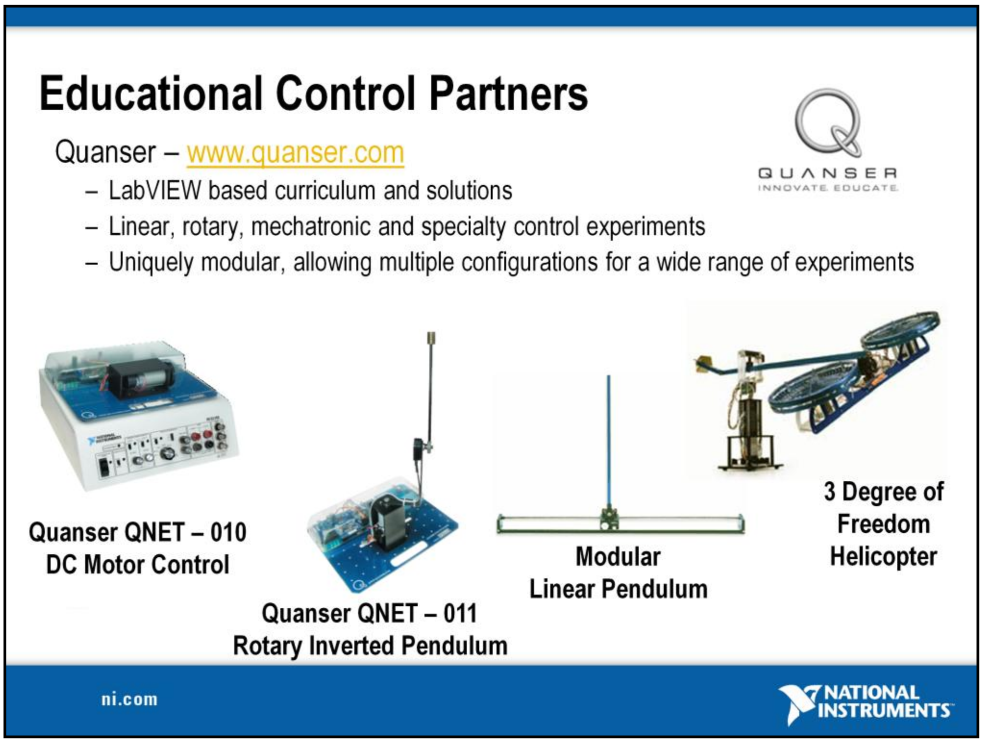
Control Design Toolkit:
- http://sine.ni.com/nips/cds/view/p/lang/en/nid/13854

Simulation Module:
- http://sine.ni.com/nips/cds/view/p/lang/en/nid/13852

LabVIEW Real-Time Module:
- http://www.ni.com/realtime

Data Acquisition and Control Hardware:
- http://www.ni.com/dataacquisition

CompactRIO Real-Time Platform:
- http://www.ni.com/compactrio



Educational Control Partners

Quanser – www.quanser.com

- LabVIEW based curriculum and solutions
- Linear, rotary, mechatronic and specialty control experiments
- Uniquely modular, allowing multiple configurations for a wide range of experiments

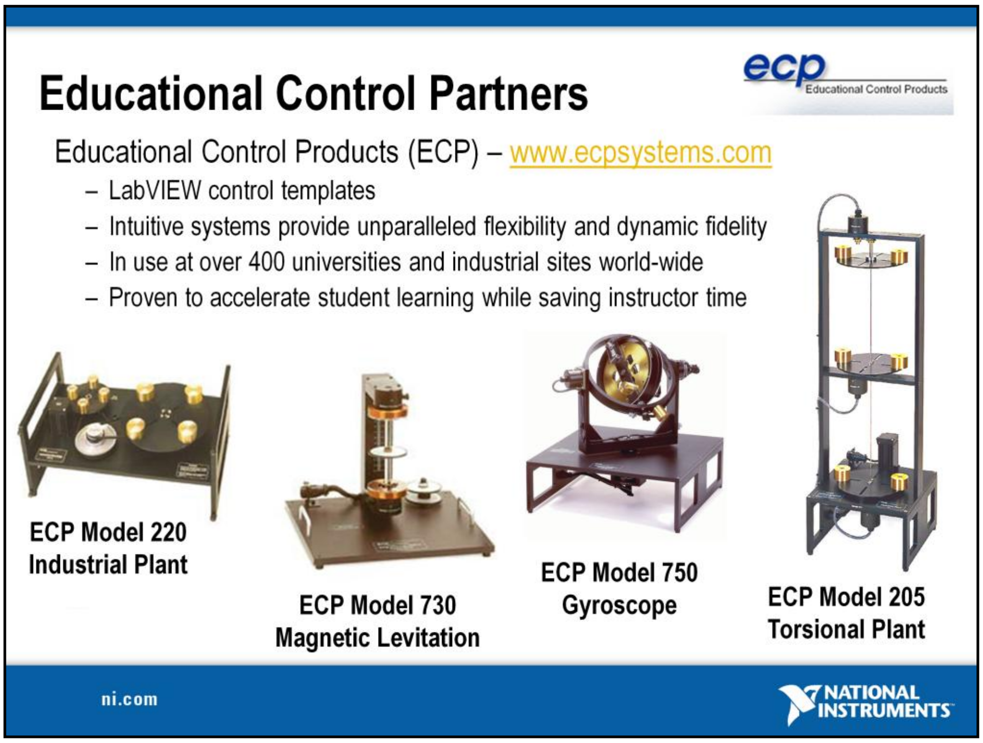
QUANSER
INNOVATE. EDUCATE.
Quanser QNET – 010
DC Motor Control
Quanser QNET – 011
Rotary Inverted Pendulum
Modular
Linear Pendulum
3 Degree of
Freedom
Helicopter
ni.com
NATIONAL INSTRUMENTS

Quanser Control Challenges

Implement and evaluate feedback strategies such as PID, LQC, adaptive or nonlinear controllers with Quanser's linear, rotary, specialty, mechatronics and custom control experiments. Our systems are uniquely modular and use the same base components that can be multi-purposed throughout a wide range of experiment configurations. So by adding and removing various modules you extend the functionality of your investment and more importantly create a greater variety of experiments and challenges to explore – no other solution provider comes close to providing this kind of flexibility.

Quanser Engineering Trainers For NI ELVIS (QNET)

The Quanser Engineering Trainers For NI ELVIS (QNET) Series is a new cost-effective line of controls education experiments that considerably increases the value of your investment in NI ELVIS & LabVIEW software. The boards are easily connected to the NI ELVIS workstation to facilitate a diverse range of controls education experiments, thereby extending the functionality of the NI platform.



Educational Control Partners

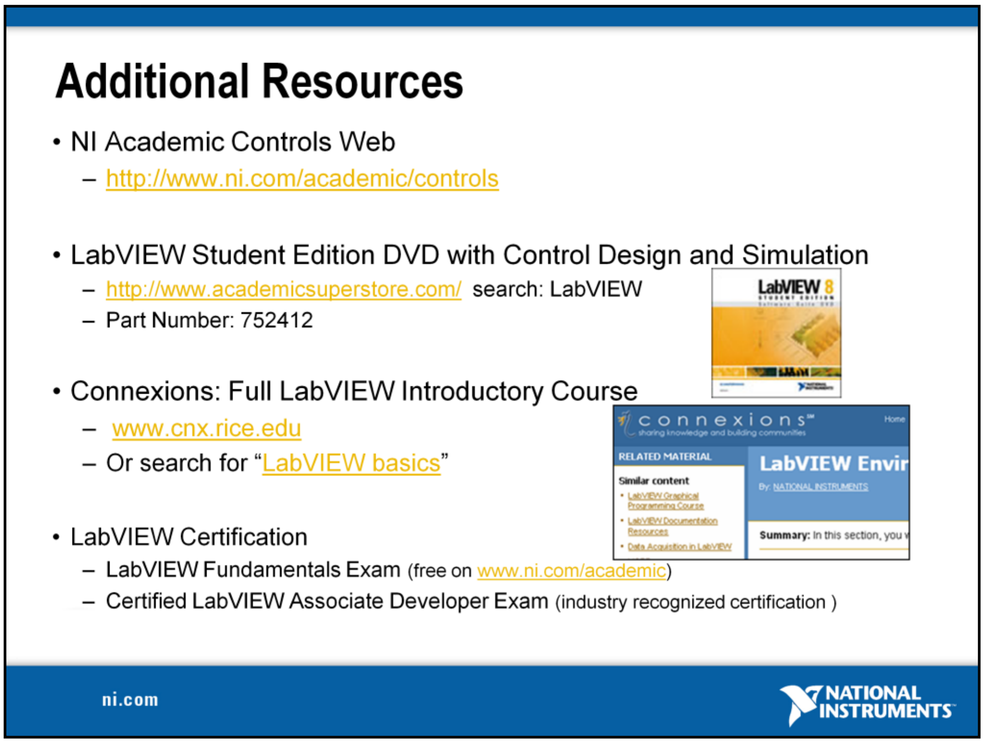
Educational Control Products (ECP) – www.ecpsystems.com

- LabVIEW control templates
- Intuitive systems provide unparalleled flexibility and dynamic fidelity
- In use at over 400 universities and industrial sites world-wide
- Proven to accelerate student learning while saving instructor time

ecp Educational Control Products
ECP Model 220 Industrial Plant
ECP Model 730 Magnetic Levitation
ECP Model 750 Gyroscope
ECP Model 205 Torsional Plant
ni.com
NATIONAL INSTRUMENTS

LabVIEW ECP Real-Time Control Option

In addition to the fully integrated ECP Executive® programs, National Instruments LabVIEW and NI data acquisition hardware can be used to develop and deploy real-time control algorithms to control select ECP electromechanical plants. The LabVIEW® environment allows for the design and simulation of control algorithms with seamless real-time deployment and execution. This approach allows for multiple levels of learning by allowing students to simply modify controller parameters on a graphical user interface or to design and create their own controller. It allows students to gain familiarity with the LabVIEW software and NI hardware that is widely used in industrial control applications.

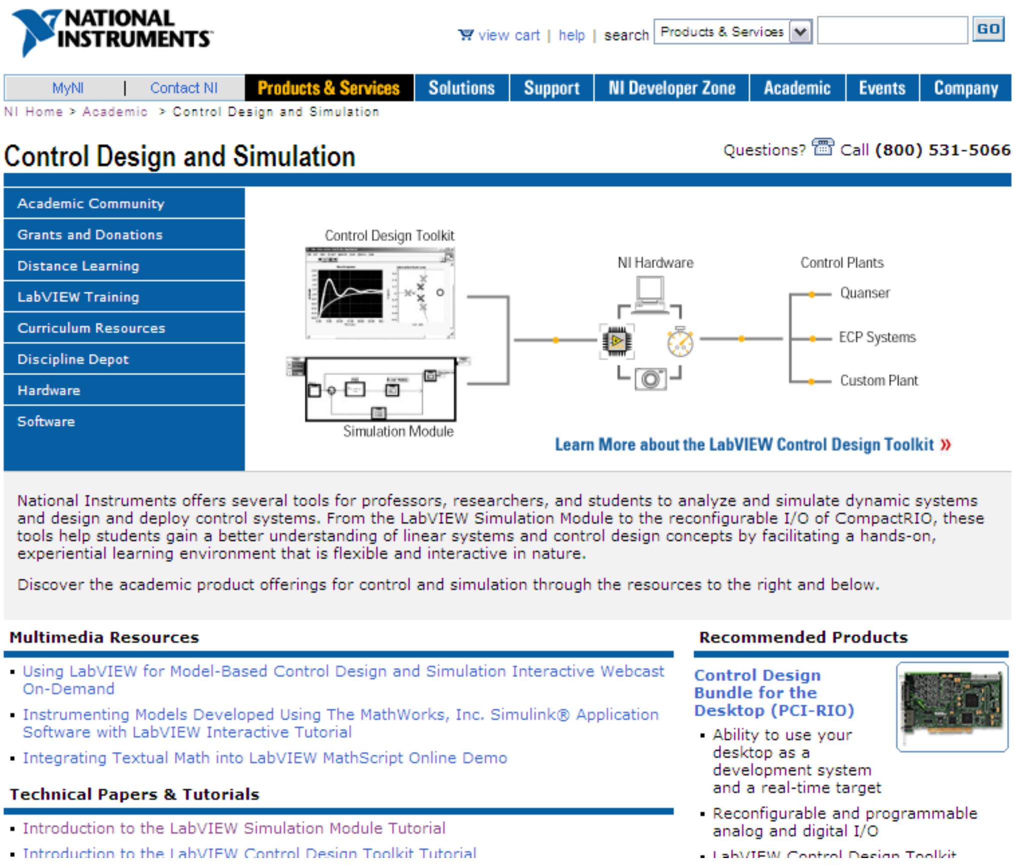


Additional Resources

- NI Academic Controls Web
  - http://www.ni.com/academic/controls
- LabVIEW Student Edition DVD with Control Design and Simulation
  - http://www.academicsuperstore.com/ search: LabVIEW
  - Part Number: 752412
- Connexions: Full LabVIEW Introductory Course
  - www.cnx.rice.edu
  - Or search for "LabVIEW basics"
- LabVIEW Certification
  - LabVIEW Fundamentals Exam (free on www.ni.com/academic)
  - Certified LabVIEW Associate Developer Exam (industry recognized certification)

ni.com

http://www.ni.com/academic/controls



view cart | help | search Products & Services GO

MyNI | Contact NI | Products & Services | Solutions | Support | NI Developer Zone | Academic | Events | Company

NI Home > Academic > Control Design and Simulation

Control Design and Simulation

Questions? Call (800) 531-5066

- Academic Community
- Grants and Donations
- Distance Learning
- LabVIEW Training
- Curriculum Resources
- Discipline Depot
- Hardware
- Software

Learn More about the LabVIEW Control Design Toolkit »

National Instruments offers several tools for professors, researchers, and students to analyze and simulate dynamic systems and design and deploy control systems. From the LabVIEW Simulation Module to the reconfigurable I/O of CompactRIO, these tools help students gain a better understanding of linear systems and control design concepts by facilitating a hands-on, experiential learning environment that is flexible and interactive in nature.

Discover the academic product offerings for control and simulation through the resources to the right and below.

Multimedia Resources

- Using LabVIEW for Model-Based Control Design and Simulation Interactive Webcast On-Demand
- Instrumenting Models Developed Using The MathWorks, Inc. Simulink® Application Software with LabVIEW Interactive Tutorial
- Integrating Textual Math into LabVIEW MathScript Online Demo

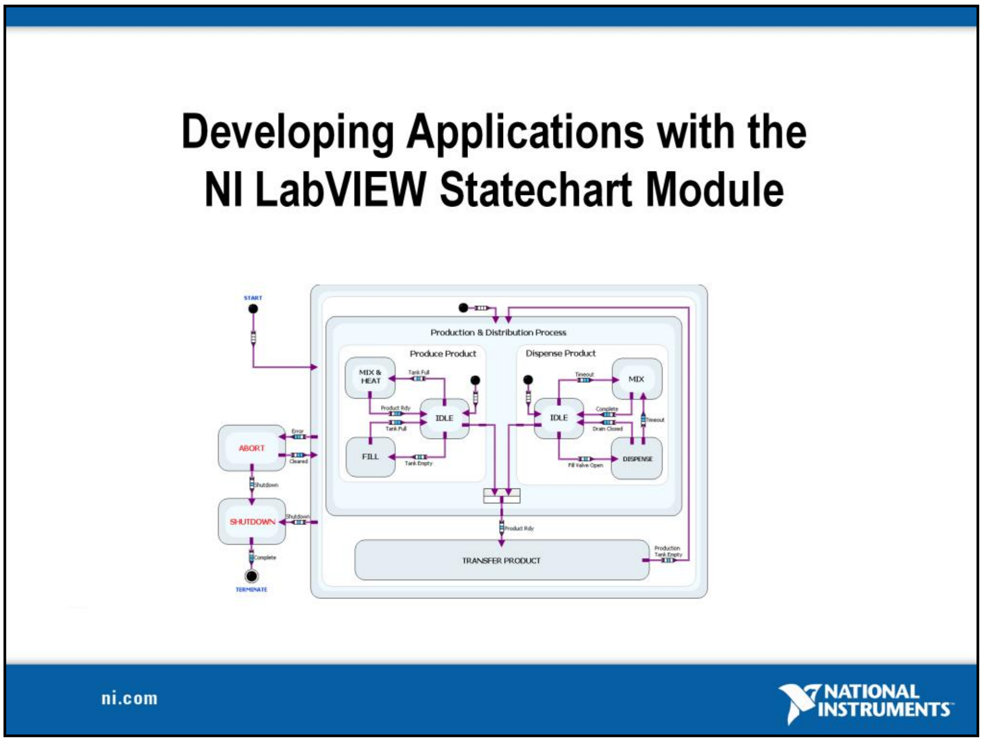
Technical Papers & Tutorials

- Introduction to the LabVIEW Simulation Module Tutorial
- Introduction to the LabVIEW Control Design Toolkit Tutorial

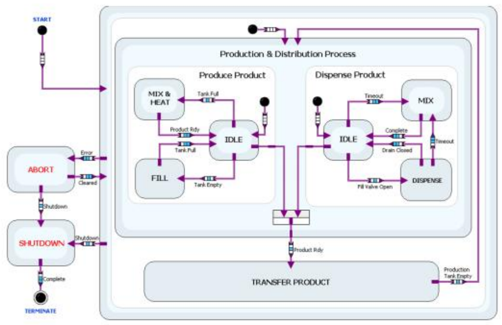
Recommended Products

Control Design Bundle for the Desktop (PCI-RIO)

- Ability to use your desktop as a development system and a real-time target
- Reconfigurable and programmable analog and digital I/O
- LabVIEW Control Design Toolkit

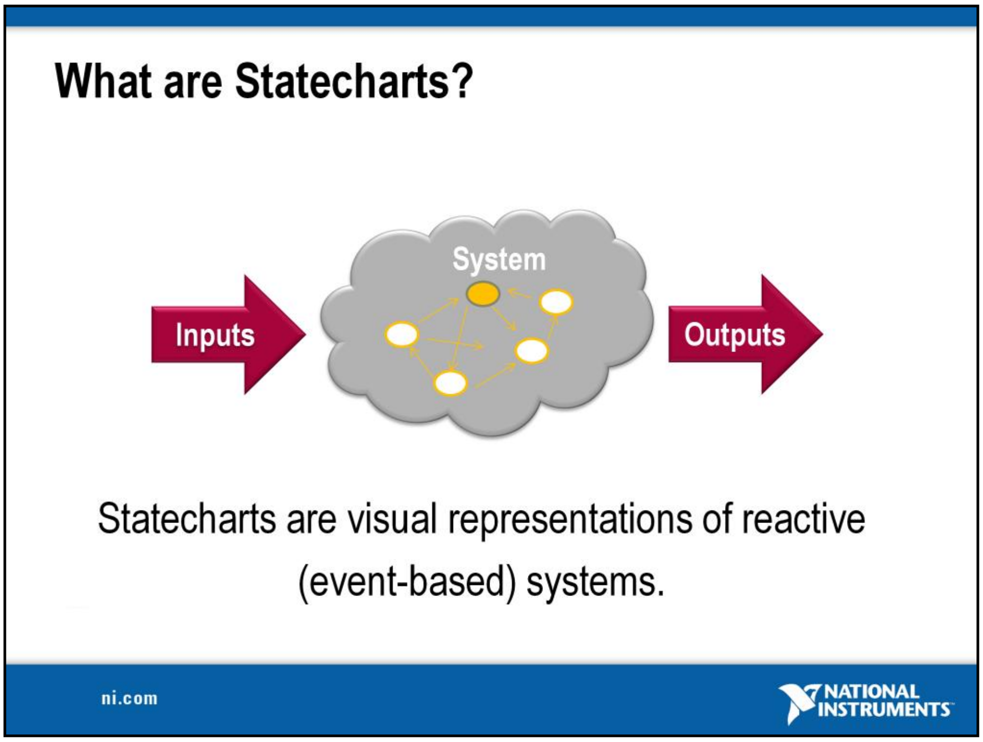


Developing Applications with the NI LabVIEW Statechart Module

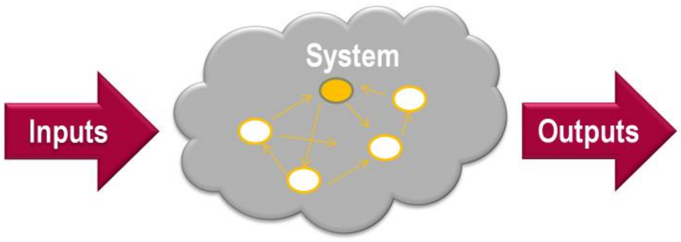


ni.com

With the release of LabVIEW 8.5, we have added another development technique to the LabVIEW Graphical System Design platform.



What are Statecharts?

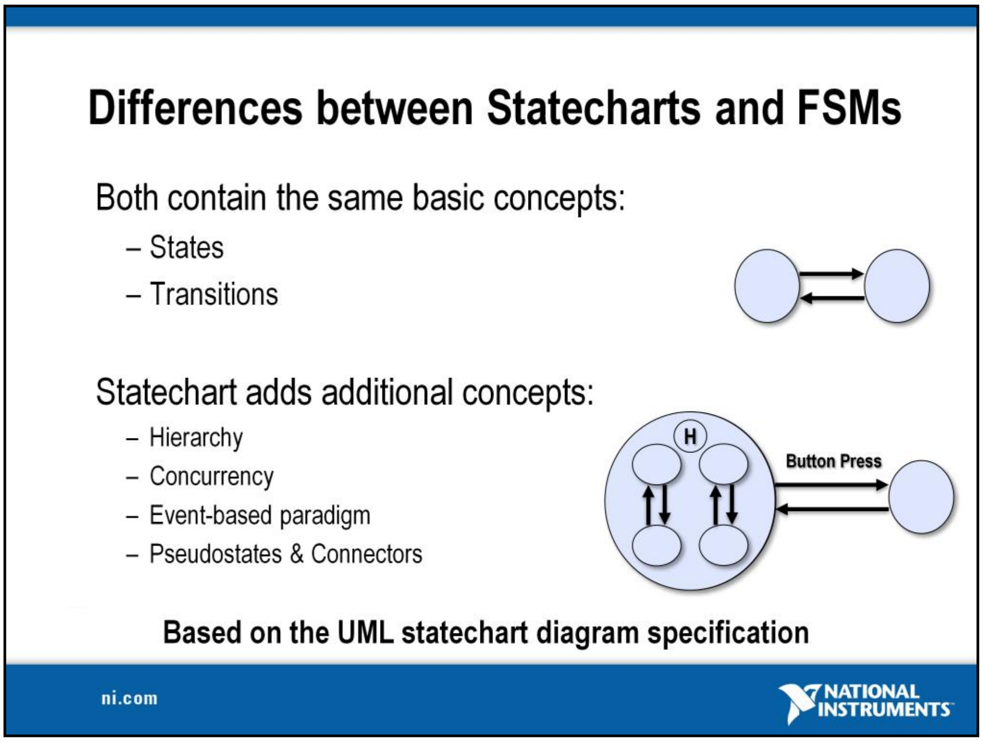


Statecharts are visual representations of reactive (event-based) systems.

ni.com

So, what is a statechart…

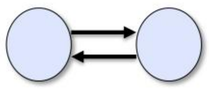
A statechart is a visual representation of a reactive system. Reactive systems, or event-driven systems, are system/applications that continuously respond to their inputs to determine the state of their outputs.



Differences between Statecharts and FSMs

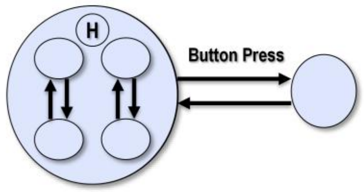
Both contain the same basic concepts:

- States
- Transitions



Statechart adds additional concepts:

- Hierarchy
- Concurrency
- Event-based paradigm
- Pseudostates & Connectors



Based on the UML statechart diagram specification

ni.com

Most of you have probably been exposed to state machines and are probably familiar with finite state machines (*this is what our state diagram toolkit creates)*. Statecharts are an extension of finite state machines that are defined as a diagram in the UML (Unified modeling language) specification.

Statecharts add the folowing concepts on top of the finite state machine…

[mutliple transistion to step through the different additions]

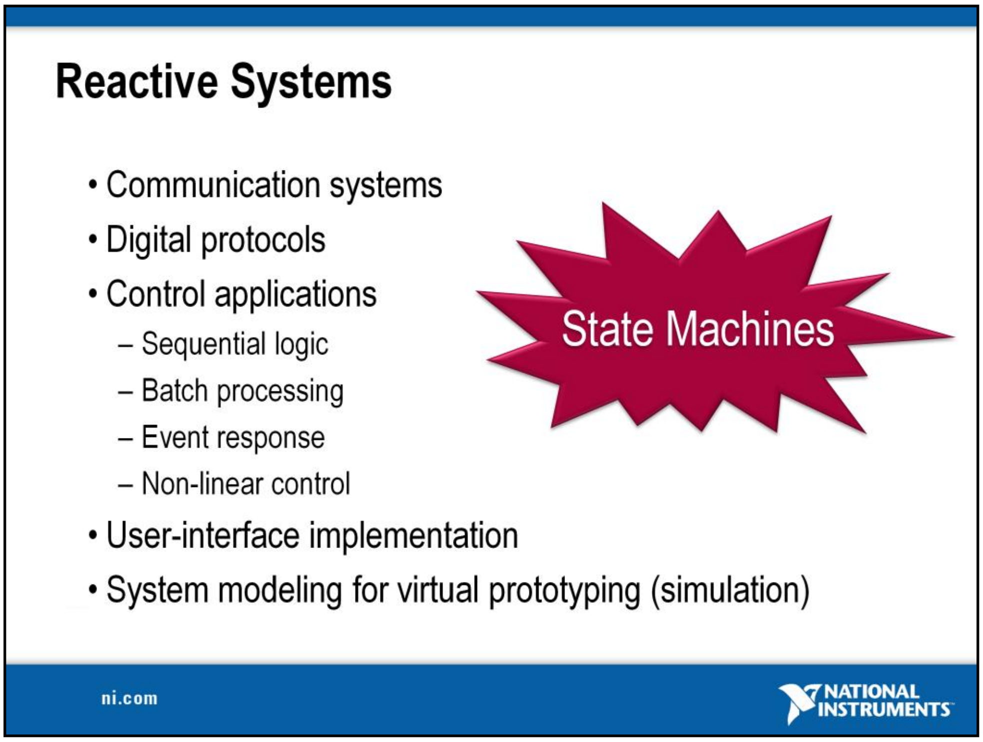
Hierarchy – ability to nest sub-statecharts (same benefits as subVIs in LV)

Concurrency – parallel states

Event-based paradigm – a representation that is truly dependent on events

Pseudostates & connectors – there are many addition semantics for statecharts as opposed to simply the states and transitions that are available in finite state machines (ie history, join, split, etc)

UML is popular in the embedded developer community and has been around since the 80's. For more information on UML, see the Statechart Wiki


Reactive Systems

- Communication systems
- Digital protocols
- Control applications
  - Sequential logic
  - Batch processing
  - Event response
  - Non-linear control
- User-interface implementation
- System modeling for virtual prototyping (simulation)

State Machines
ni.com
NATIONAL INSTRUMENTS

Statecharts can be used in a variety of applications…..

Comments you can mention:

- the original GPIB spec was represented in a statechart

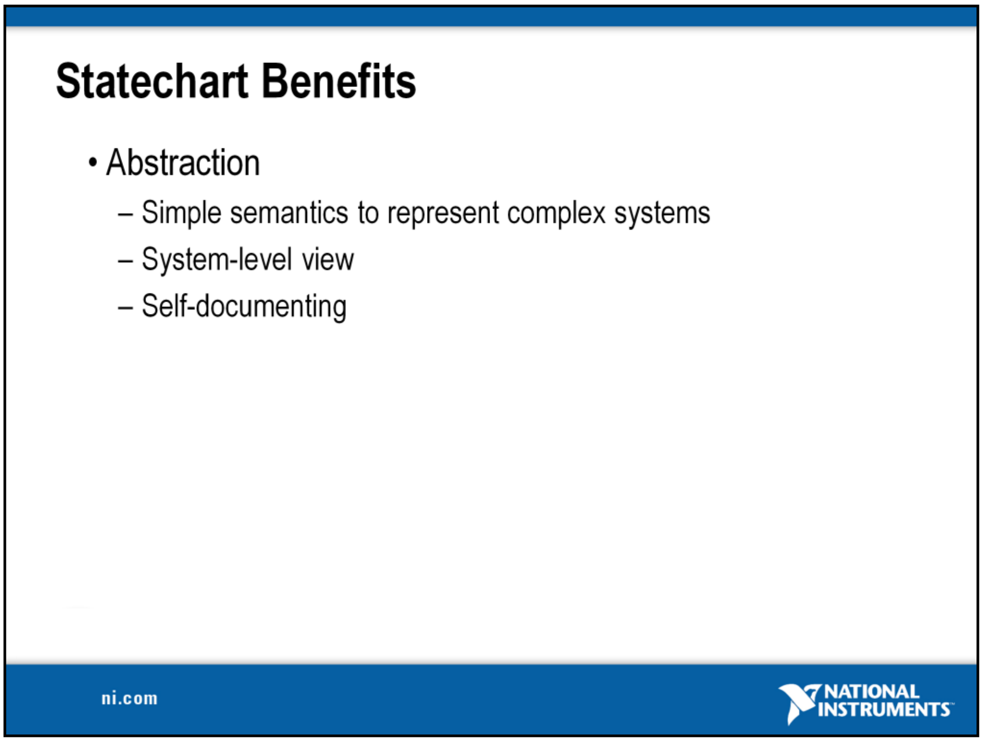
- control systems from simple on/off controllers to the most advanced modern control techniques can utilize statecharts

- any event driven system such as user-interfaces may benefit from the use of statecharts

[Transition – "state machines" appears]

Essentially, any application that uses a state machine architecture can benefit from the use of statecharts.

You might want to remind the audience of the state machine architecture we teach in LabVIEW training that consists of a case structure in a while loop with a shift register that holds the next state. Any application that would use this architecture could benefit from the statechart module.



Statechart Benefits

- Abstraction
  - Simple semantics to represent complex systems
  - System-level view
  - Self-documenting

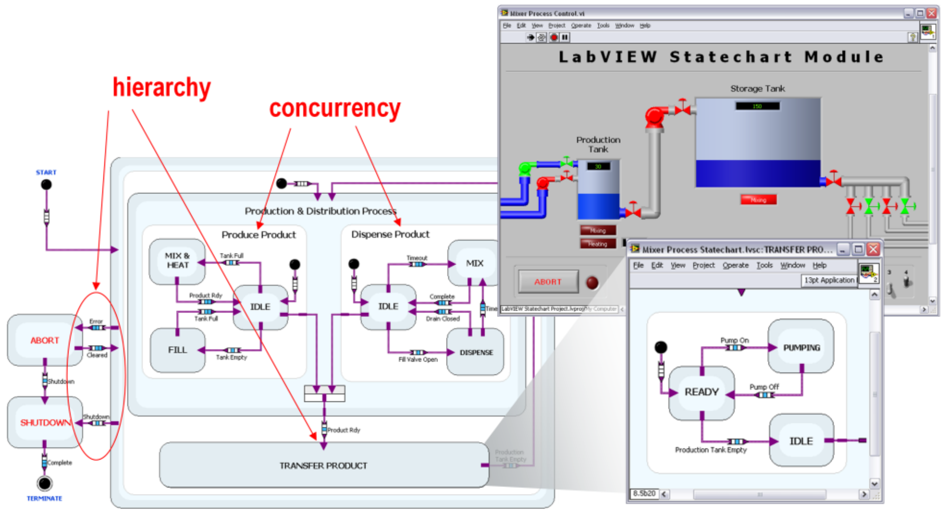
ni.com

So what are those benefits…

First, statecharts allow you to create a system-level view of your application that provides abstraction to your application. This abstraction simplifies development and the transfer of knowledge between developers. Essentially, statecharts can provide an executable specification that can be easily understood by management, domain experts and developers alike.

Let’s take a look at some examples of the abstraction provided by statecharts…

Machine & Process Control



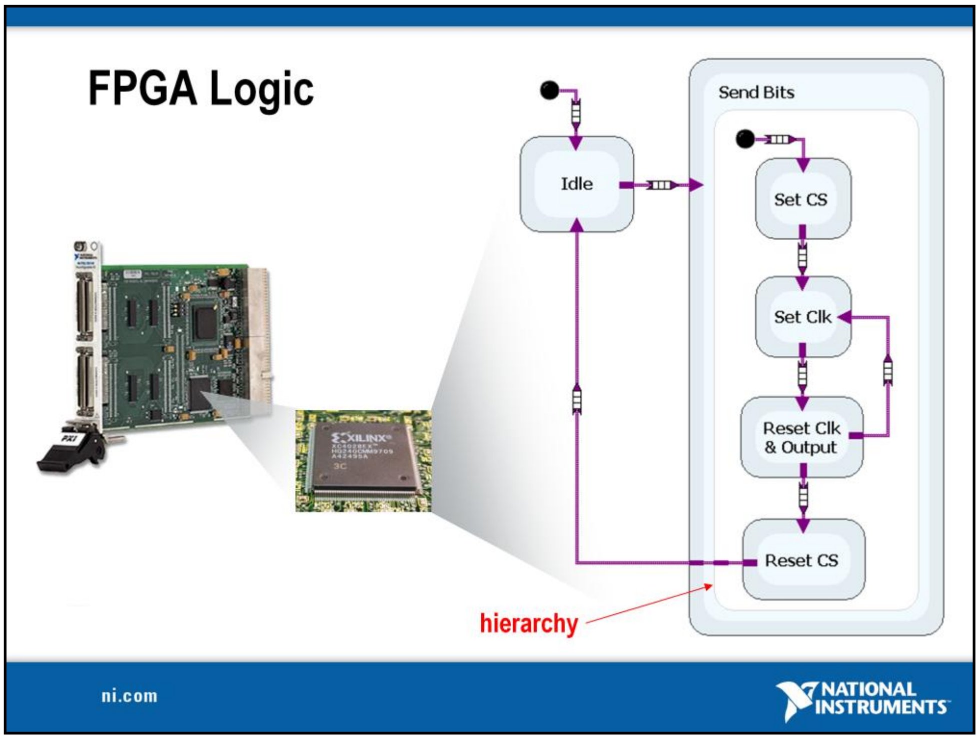
ni.com

NATIONAL INSTRUMENTS

This is a statechart used to define the logic of a system that produces, stores, and distributes a product. We can see the parallel operation of the produce and dispense systems. The “join” object denotes that both of these systems must be in an idle state before the statechart can transition from the “Production & Distribution Process” super-state to the “Transfer Product” super-state.

[transition]

Here we can see examples of the concepts of hierarchy and concurrency provided by statecharts.



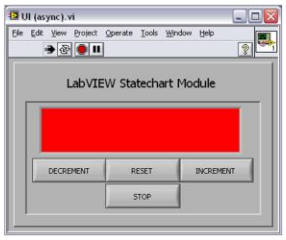
FPGA Logic

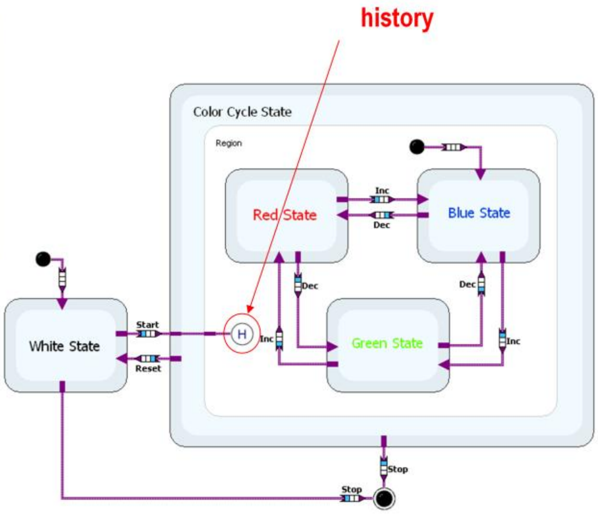
Briefly explain what an FPGA is , the benefits of RIO, and how you can use LabVIEW to create FPGA personalities.

Most digital logic designs use a state machine architecture which make statecharts an ideal way to represent their logic at a high-level.

This is a state machine representation of the logic for a SPI (serial peripheral interface) digital communication protocol that has been implemented using LabvIEW FPGA. Even with no knowledge of LabVIEW, the process executing on the FPGA can be easily understood because of the statechart representation.

User Interfaces


UI (async).vi
File Edit View Project Operate Tools Window Help
LabVIEW Statechart Module
DECREMENT
RESET
INCREMENT
STOP

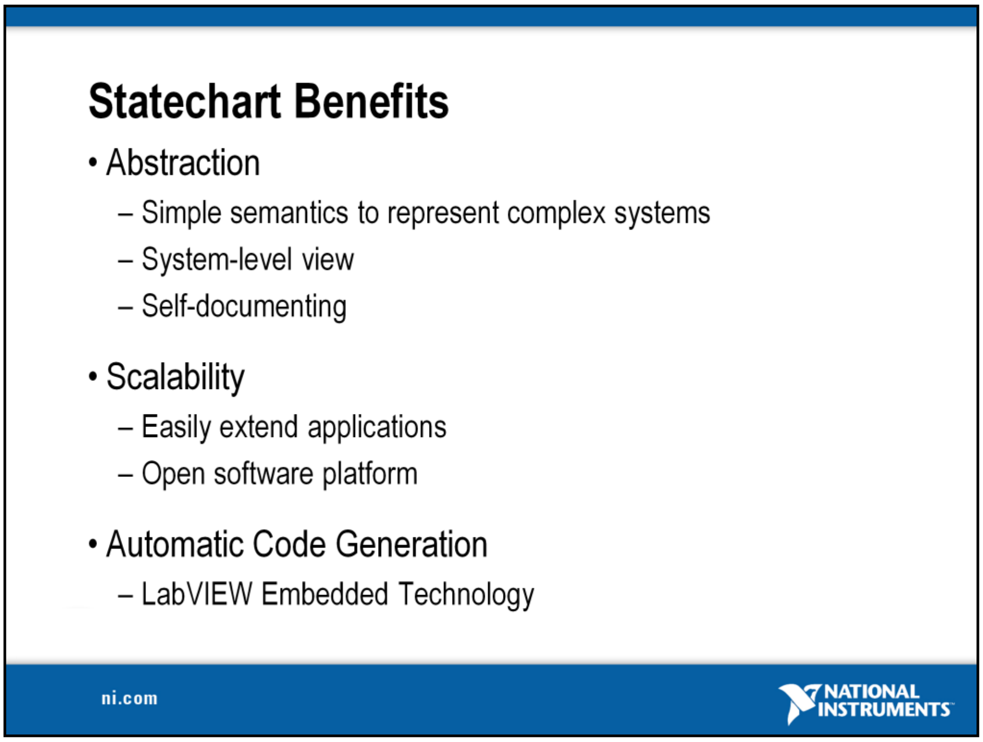


ni.com

Statecharts are ideal for representing asynchronous/event-driven applications/system. User-interfaces are inherently asynchronous and in complex implementation can benefit greatly from the use of statecharts.

[transition]

In this example, the “history” pseudostate has been used to specify that the application should “remember” the last state it was in when it returns to the “color cycle state” from the “white state”.



Statechart Benefits

- Abstraction
  - Simple semantics to represent complex systems
  - System-level view
  - Self-documenting
- Scalability
  - Easily extend applications
  - Open software platform
- Automatic Code Generation
  - LabVIEW Embedded Technology

ni.com

The implementation of the LabVIEW Statechart module also helps to ensure that you are creating applications that can easily scale to adapt to changing needs. The design architecture it enforces lends itself to easily expanding or changing applications without significant architecture changes.

Examples you can bring up:

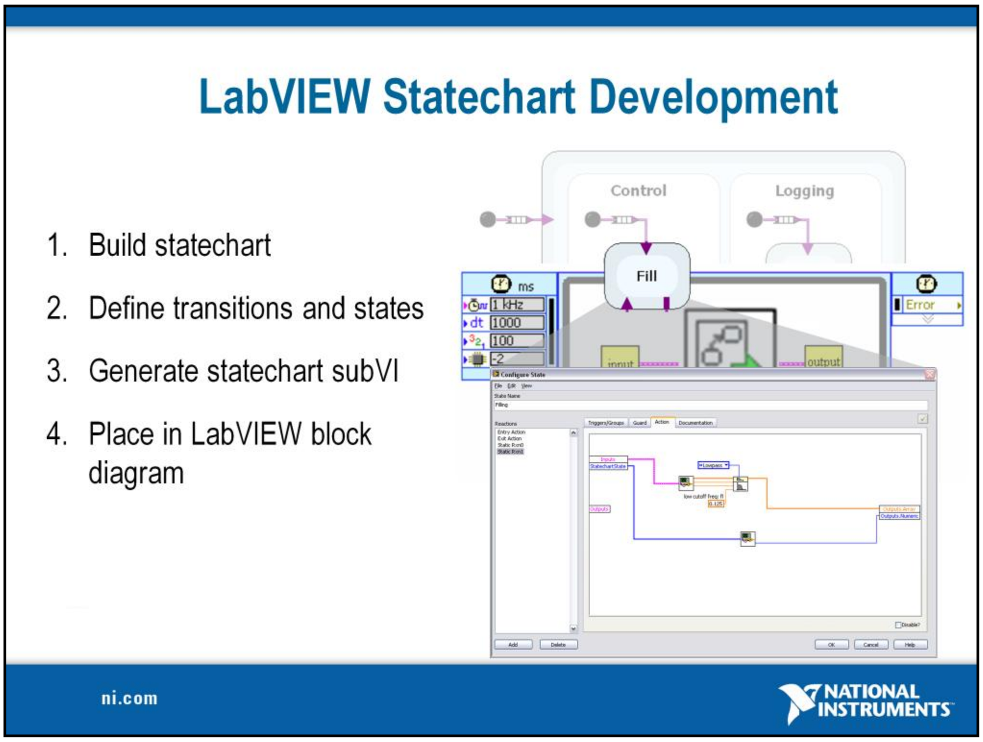
- all interfaces to the statechart subVI are 'strict type defs' so changing the number/type of inputs or outputs to your statechart will propagate to all uses of that data-type throughout your application.

- Transition logic is encapsulated in the statechart, so changes the paths through your statechart simply requires re-connecting the transition as opposed to digging through a case structure to update the logic of your state machine

- Expanding you system can be easily accomplished by adding additional states or converting states to super-states and adding additional statechart logic within it.

- In general, many concepts that you will need throughout a state machine implementation (transition logic, variable management, etc) are abstracted when using the statechart module reducing opportunities for error and making development and maintenance of your application easier.

LabVIEW Embedded Technology provides the ability to not only generate C-code for deployment to custom hardware, but seamless deployment to 100's of off-the-shelf, modular hardware targets on a variety of processor and FPGA platforms designed to meet different application requirements (performance, environmental, cost, size, etc)



LabVIEW Statechart Development

1. Build statechart
2. Define transitions and states
3. Generate statechart subVI
4. Place in LabVIEW block diagram

Control
Logging
Fill
ms
1 kHz
dt 1000
100
-2
input
output
Error
Configure State

ni.com

To develop with the LabVIEW Statechart Module, you begin by designing the states and transitions of your application with the same drag and drop ease-of-use that you are accustom to in the LabVIEW environment.

[transition]

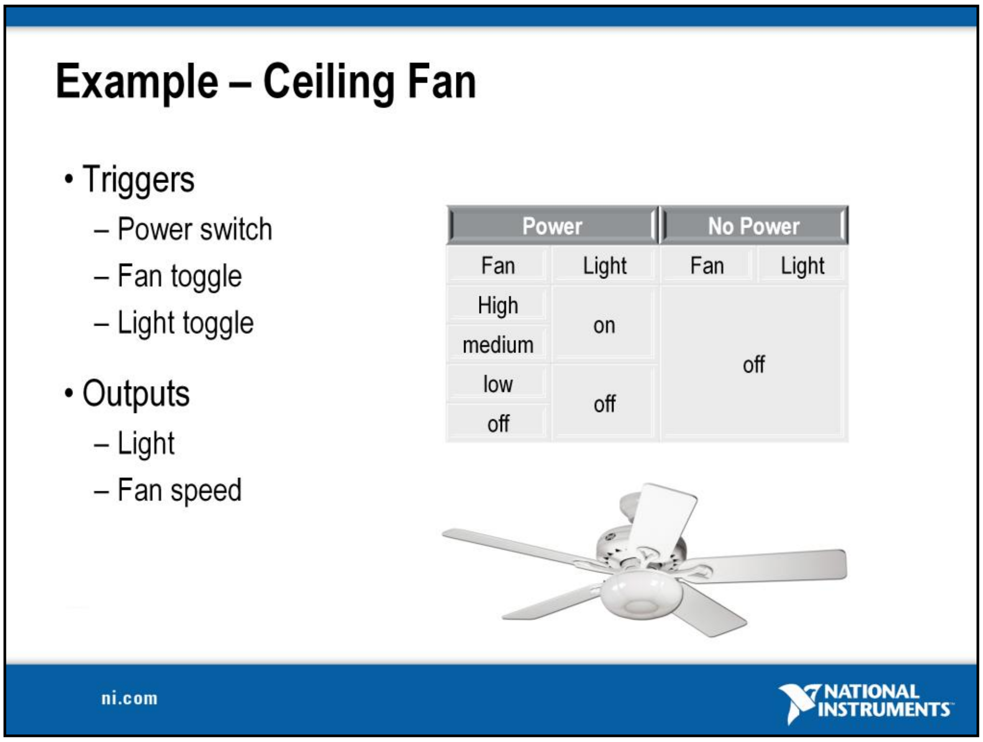
Next you define the state and transion behavior/logic for your statechart.

[transition]

Once you statechart is complete, the state chart logic is encapsulated inside a LabVIEW subVI.

[transition]

Finally, you utilize the subVI in the rest of the LabVIEW application.



Example – Ceiling Fan

- Triggers
  - Power switch
  - Fan toggle
  - Light toggle
- Outputs
  - Light
  - Fan speed

Power		No Power	
Fan	Light	Fan	Light
High	on	off	
medium			
low	off		
off			

ni.com

So let’s build a statechart.

A simple system that we are all familiar with is a ceiling fan. It has a light that is either on or off and a fan that can be in various speeds or off. So what are the event/triggers in this system?

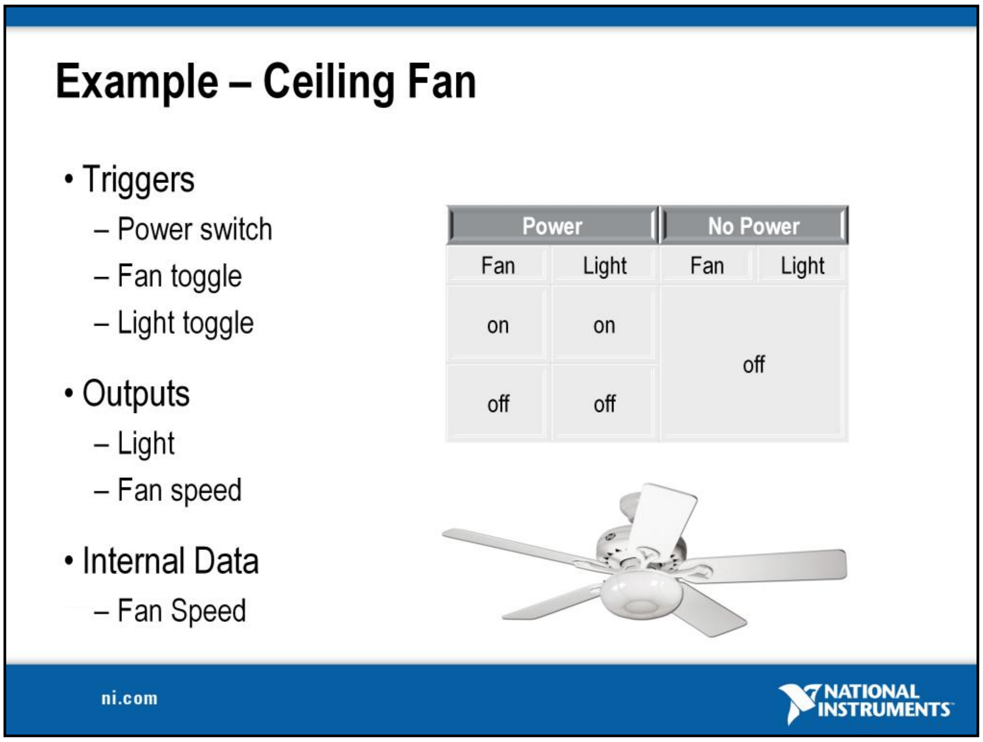
[wait for audience to respond]

And what are our outputs?

[wait for audience to respond]

[transition]

So, one way we could represent the states of the system could be this (table). OR, we could utilize an internal variable in our statechart and represent it like this…



Example – Ceiling Fan

- Triggers
  - Power switch
  - Fan toggle
  - Light toggle
- Outputs
  - Light
  - Fan speed
- Internal Data
  - Fan Speed

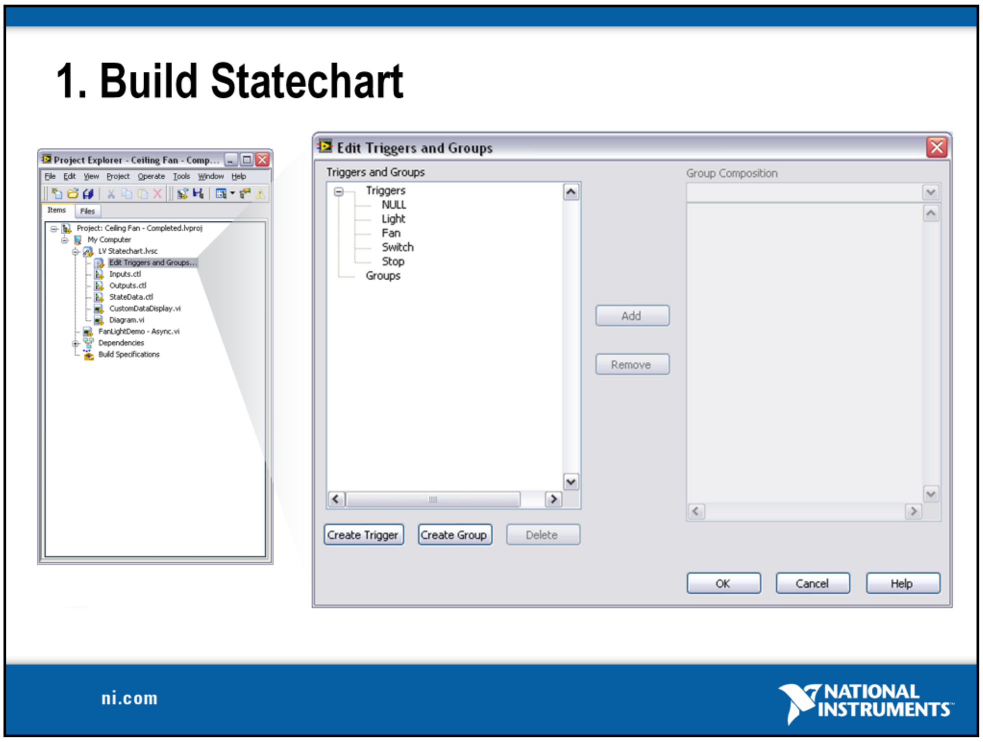
Power		No Power	
Fan	Light	Fan	Light
on	on	off	
off	off		

ni.com

NATIONAL INSTRUMENTS

Now we have eliminate the "speed" states of our fan and replaced them with "on" / "off" states and a "fan speed" state variable.

Let's build this application using LabVIEW and LabVIEW Statecharts…



1. Build Statechart

To build a statechart in LabVIEW, we will begin by adding a statechart to a LabVIEW project. A LabVIEW statechart consists of several elements that we will define as we create our statechart.

They are:

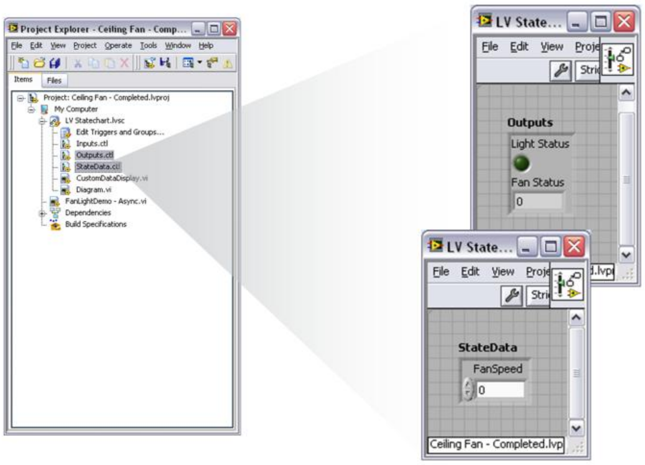
- Triggers or events for your system
- Inputs and outputs to the statechart
- Internal state data for your statechart
- The actual statechart diagram where you build the statechart
- and a custom data display that can be used to debug your statechart

The first element that you would typically define is the set of triggers or events for your application. In this application, the events are:

- fan pull
- light pull
- power switch

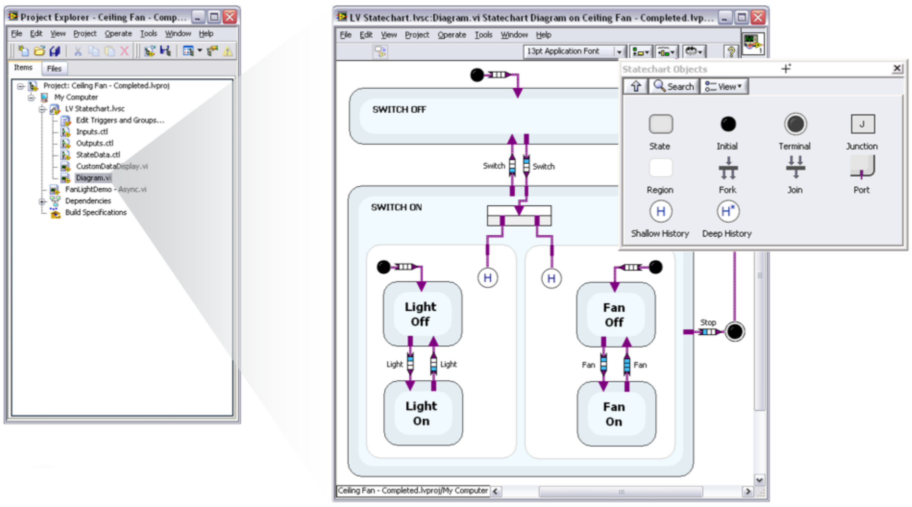
- stop has also been added in order to terminate the application

1. Build Statechart



Next we will specify the inputs, outputs, and internal state data for our statechart. In this example, the outputs are the light state (on, off) and the fan speed (off, low, med, high). We also have a state variable for the current fan speed.

1. Build Statechart

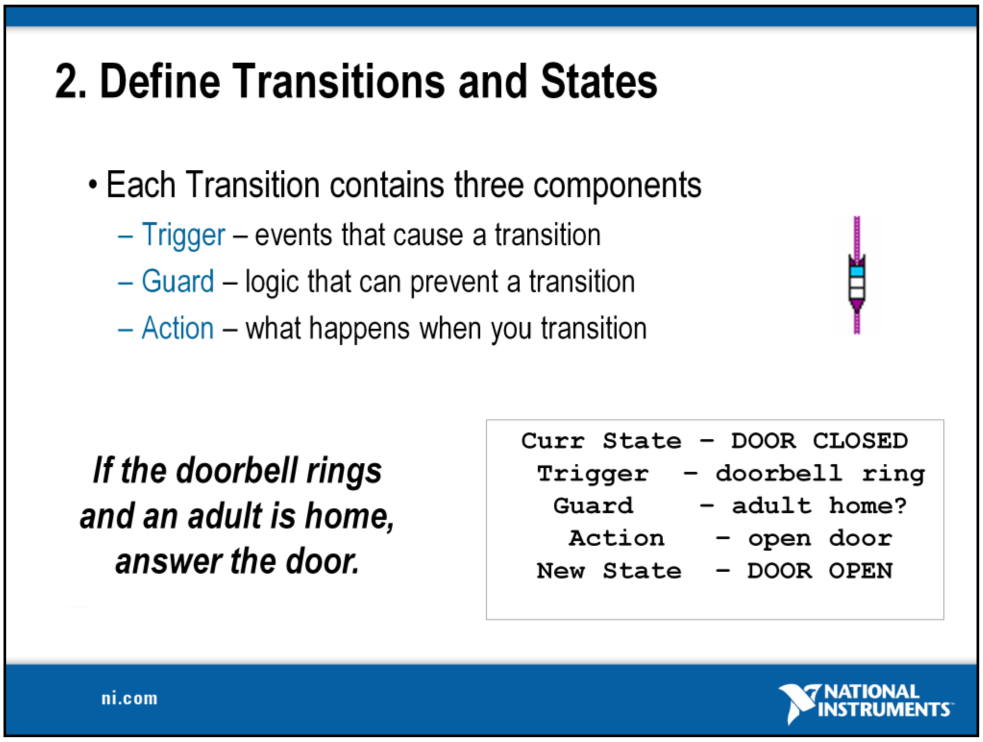


ni.com

NATIONAL INSTRUMENTS

Now we can begin building our statechart.

When you create the actual statechart, there is a new pallet that is only used to create LabVIEW statecharts. (these objects do not work in LV block diagrams and other LV functions do not work in LV statechart diagrams).



2. Define Transitions and States

- Each Transition contains three components
  - Trigger – events that cause a transition
  - Guard – logic that can prevent a transition
  - Action – what happens when you transition

If the doorbell rings and an adult is home, answer the door.

```
Curr State - DOOR CLOSED
 Trigger   - doorbell ring
  Guard     - adult home?
   Action    - open door
 New State   - DOOR OPEN
```

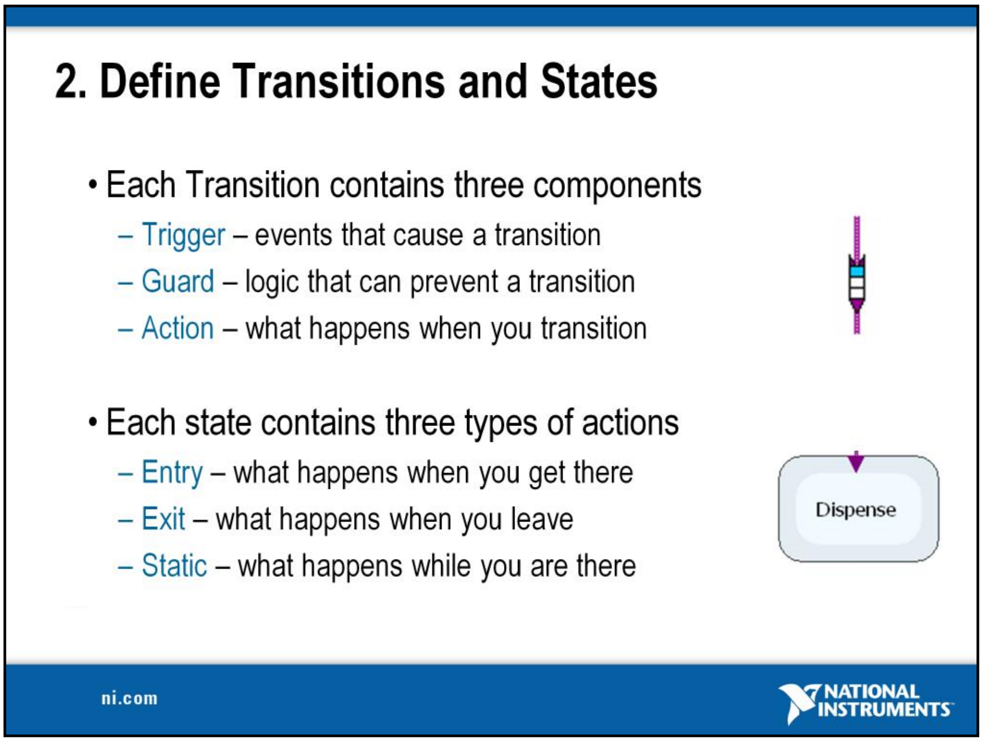
ni.com

NATIONAL INSTRUMENTS

Once the statechart has been created, we will define the logic and functionality of our statechart using LabVIEW graphical programming. The two types of objects in a statechart that need to be further specified are states and transitions.

A transition determines what states your application can move into from the current state. You can configure a trigger, guard, and action for each transition. The trigger signals that a transition should take place, the guard code can be used to “block” the transition even though the trigger occurred, and the action is a task that will be completed before entering the next state.

EXAMPLE *– This example illustrates the trigger-guard-action concept used in transitions and static reactions of a statechart. In this example, a child has be instructed that they are not to open the door unless an adult is home. The box to the right breaks down the element of this behavior/system as it would be described using statechart notation.*



2. Define Transitions and States

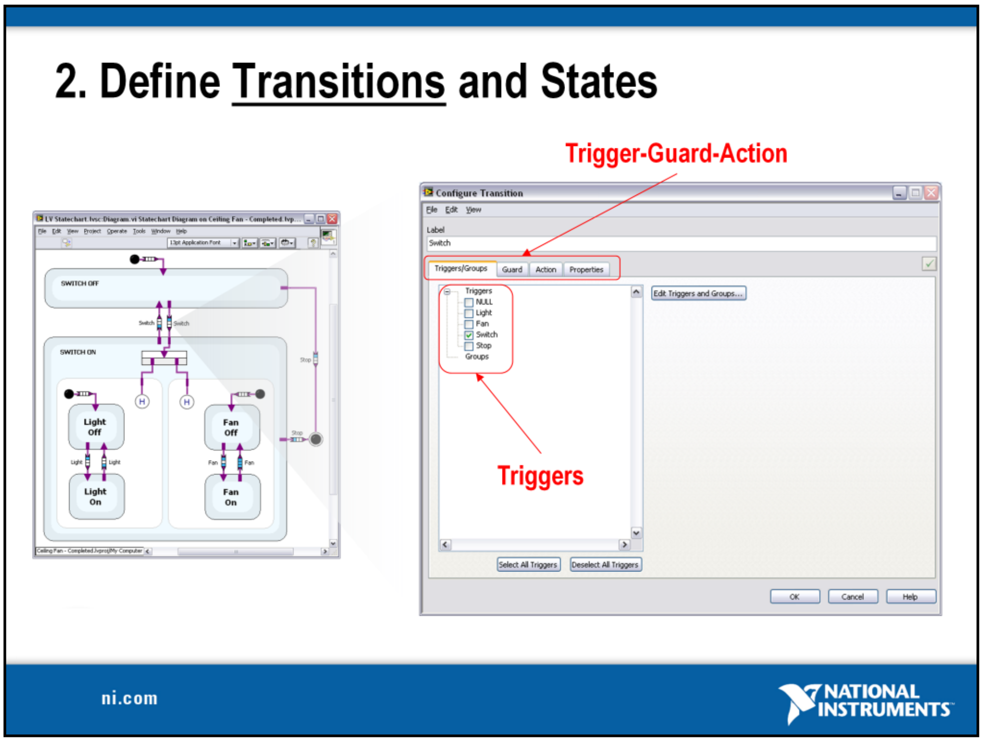
- Each Transition contains three components
  - Trigger – events that cause a transition
  - Guard – logic that can prevent a transition
  - Action – what happens when you transition
- Each state contains three types of actions
  - Entry – what happens when you get there
  - Exit – what happens when you leave
  - Static – what happens while you are there

ni.com

A state also contains 3 components.

The entry and exit actions will always execute when entering or leaving a state. Because of this, they have no trigger or guard associated with them.

An arbitrary number of static actions (or reactions) may also be specified for a each state. Static reactions have a trigger and guard associated with them like transitions do (entry and exit actions do not because they always and only occur at entry/exit) and are only evaluated if there are no valid transitions out of the state (if the state cannot be left, then the reactions are evaluated).



2. Define Transitions and States

Label: Switch

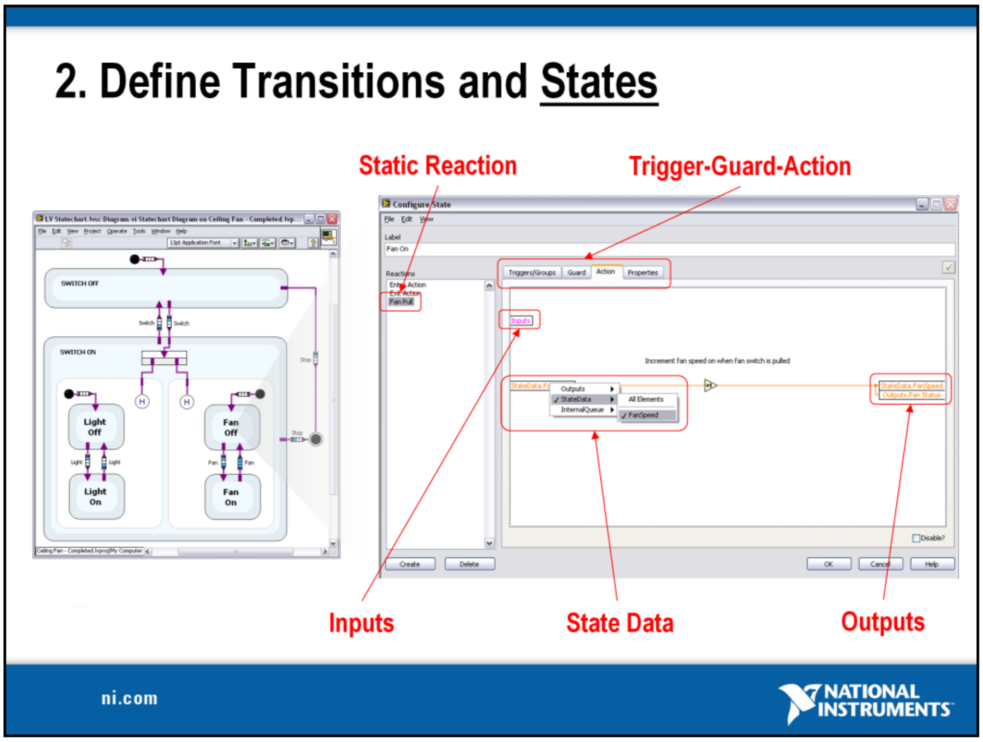
Triggers
- NULL
- Light
- Fan
- [✓ Switch]
- Stop

Groups

To define a transition, you right-click on the transition node to bring up the transition configuration dialog. This is where you will use LV code to define the logic for your transitions.

[transition]

Notice the trigger-guard-action tabs which will also take us to the different windows for selecting our trigger/triggers and specifying our guard and action logic with LabVIEW.



2. Define Transitions and States

ni.com

NATIONAL INSTRUMENTS

To define a state, you right-click on the state to bring up the state configuration dialog. This is where you will use LV code to define the logic for your states.

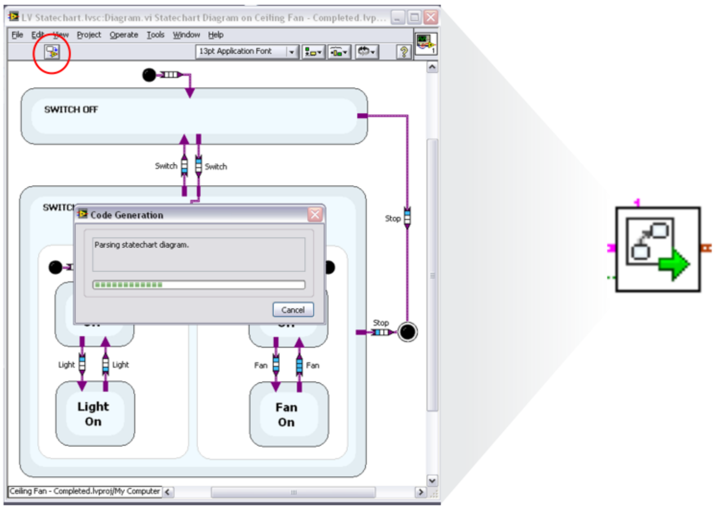
[transition]

Recall that there are 3 components to a state (entry/exit actions and static reactions). This state has a static reaction configured that will increment the internal state variable “fan speed” when it occurs. The tabs across the top take us to different “block diagrams” for specifying each part of the action or reaction using LabVIEW.

[transition]

The inputs, outputs, and state data we previously defined for our statechart are available on the left and right data nodes and can be selected in the same way you would select an element using the ‘bundle/unbundle by name’ function.

3. Build Statechart SubVI


LV Statechart.lvsc:Diagram.vi Statechart Diagram on Ceiling Fan - Completed.lvp...
File Edit View Project Operate Tools Window Help
13pt Application Font
SWITCH OFF
Switch
Switch
SWITCH
Stop
Stop
Code Generation
Parsing statechart diagram.
Cancel
Light
Light
Fan
Fan
Light On
Fan On
Ceiling Fan - Completed.lvproj/My Computer

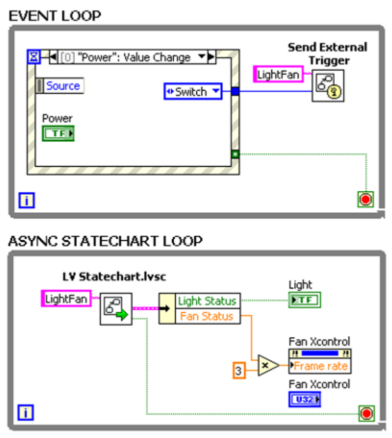
ni.com

NATIONAL INSTRUMENTS

Once we have completed our statechart, we will automatically generate LabVIEW code and encapsulate it in a subVI that we can use in our block diagram.

NOTE – the code in the subVI cannot be accessed

4. Place in LabVIEW Block Diagram



Asynchronous Usage

- User interface
- Interruption handling
- Modeling event driven systems

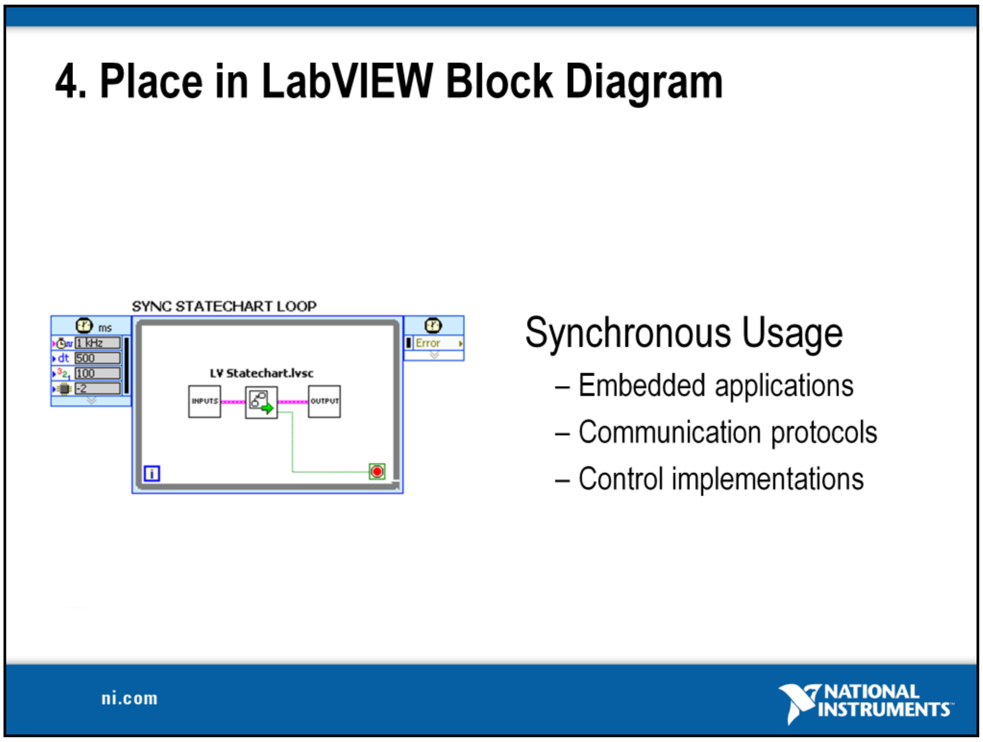
ni.com

NATIONAL INSTRUMENTS

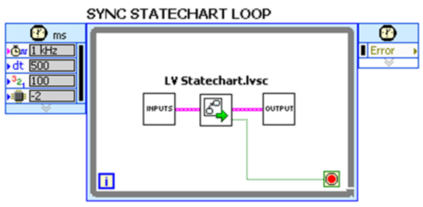
Statecharts can be used in two modes within LabVIEW, asynchronous and synchronous.

In the async case, the statechart waits in a separate loop for another process or task to send an asynchronous trigger using a special LabVIEW Queue. This communication is implemented using the ‘Send External Trigger’ subVI.

This architecture is best for developing truly asynchronous systems or applications such as user-interfaces.



4. Place in LabVIEW Block Diagram



Synchronous Usage

- Embedded applications
- Communication protocols
- Control implementations

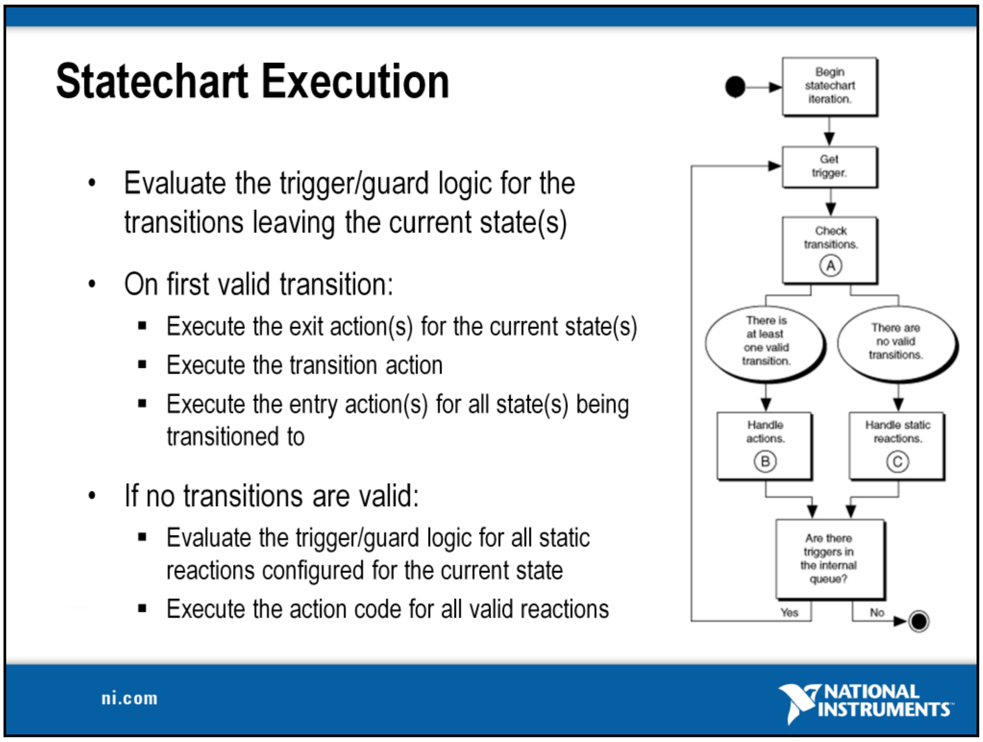
ni.com

NATIONAL INSTRUMENTS

In the synchronous case, the statechart is executed continuously and would typically reside in a timed-loop.

This architecture is commonly used for embedded applications, implementing communication protocols, and closed-loop control applications.

NOTE: Async triggers can still be used in the sync mode, but the 'Send External Trigger' VI would not be used. The user would need to implement the logic themselves.



Statechart Execution

- Evaluate the trigger/guard logic for the transitions leaving the current state(s)
- On first valid transition:
  - Execute the exit action(s) for the current state(s)
  - Execute the transition action
  - Execute the entry action(s) for all state(s) being transitioned to
- If no transitions are valid:
  - Evaluate the trigger/guard logic for all static reactions configured for the current state
  - Execute the action code for all valid reactions

Begin statechart iteration.
Get trigger.
Check transitions.
A
There is at least one valid transition.
There are no valid transitions.
Handle actions.
B
Handle static reactions.
C
Are there triggers in the internal queue?
Yes
No

ni.com

NATIONAL INSTRUMENTS

So what happens when the statechart subVI is called? The chart on the right is taken from the documentation for the statechart module. Essentially, the statechart module takes 1 step each time it is called. 1 step involves:

[there are 3 transitions to help summarize what happens on each call of the SC subVI]

Items to note:

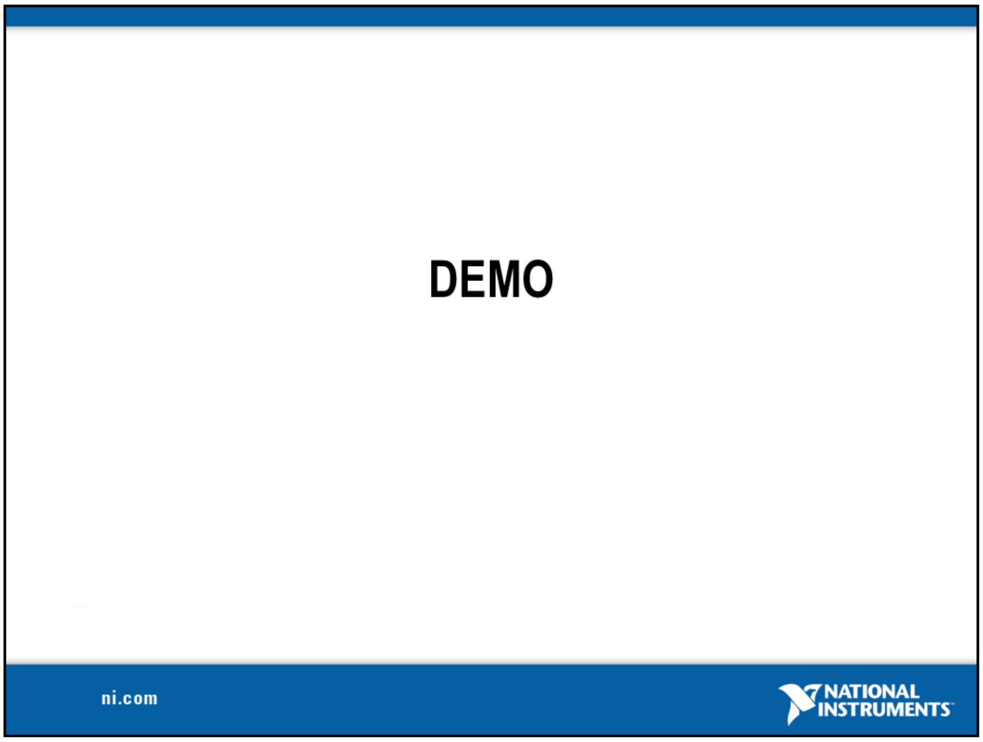
• For a transition or static reaction to be valid, both the trigger and guard must evaluate to "TRUE"

• If you have a valid transition, you will not evaluate static reactions for the state you are leaving

• Valid transitions for a state are all exit transitions directly connected to the current state and all exit transitions connected to states that contain the current state (superstates)

• You can specify a priority for transitions, but transitions from a superstate will always be higher priority than transitions from a state within a superstate

(the chart on the right can be found in the documentation for the statechart module)



DEMO

ni.com

Open the xxxxx project

Expand the statechart to show the components

Open the Events & Triggers window

Open the outputs component

Open the diagram

Right click on a state to show the logic defined with LabVIEW

Double-click on a transition to show how it is configured

Close the statechart (if you made changes you may need to re-generate code)

Open the VI and goto the block diagram

Right click on the statechart subVI to show that it is linked to the statechart in the project

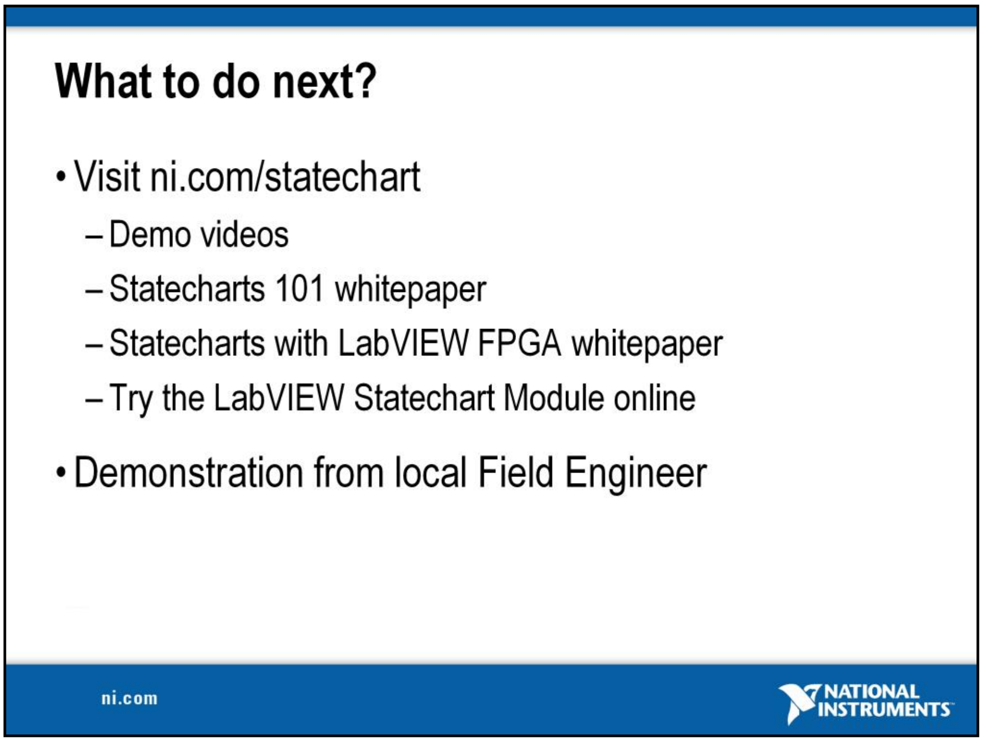
Explain the asynchronous architecture again

Right click on the statechart subVI and select “Debug statechart” and enable highlight execution on the statechart

Return to the FP and run the VI in highlight execution mode

Show how the statechart loop waits for an event/trigger from the loop above

Click the light button to show the statechart waking up to execute



What to do next?

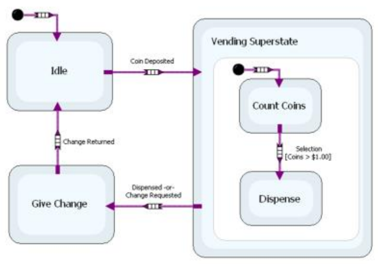
- Visit ni.com/statechart
  - Demo videos
  - Statecharts 101 whitepaper
  - Statecharts with LabVIEW FPGA whitepaper
  - Try the LabVIEW Statechart Module online
- Demonstration from local Field Engineer

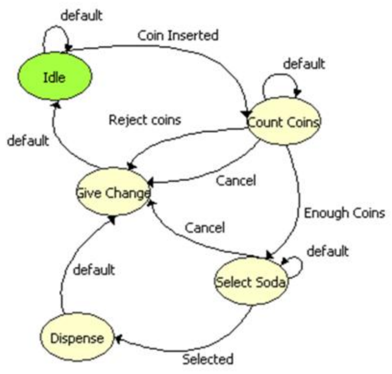
ni.com

NATIONAL INSTRUMENTS

There is no evaluation version of the LV Statechart module, so encourage attendees to use the online eval (link at ni.com/statechart) or request a visit from their local field engineer.

Statecharts vs. State Diagram Toolkit

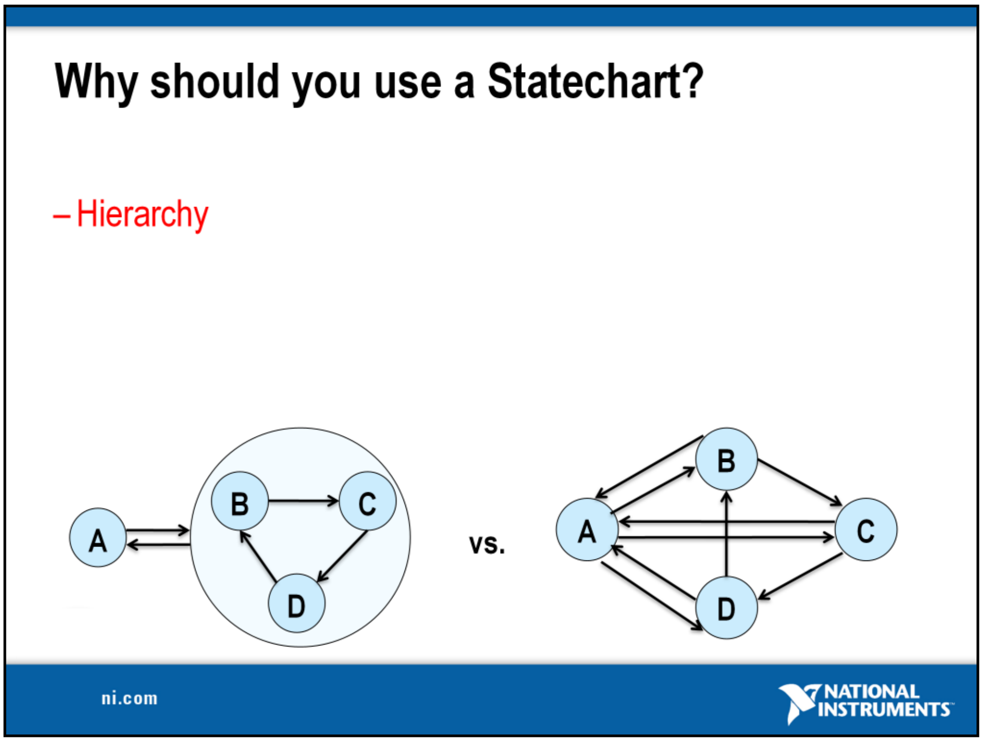




ni.com

NATIONAL INSTRUMENTS

Statecharts are an extension of the finite state machine. Here is a simple example of a FSM and statechart representation of the logic of a vending machine controller.


Why should you use a Statechart?

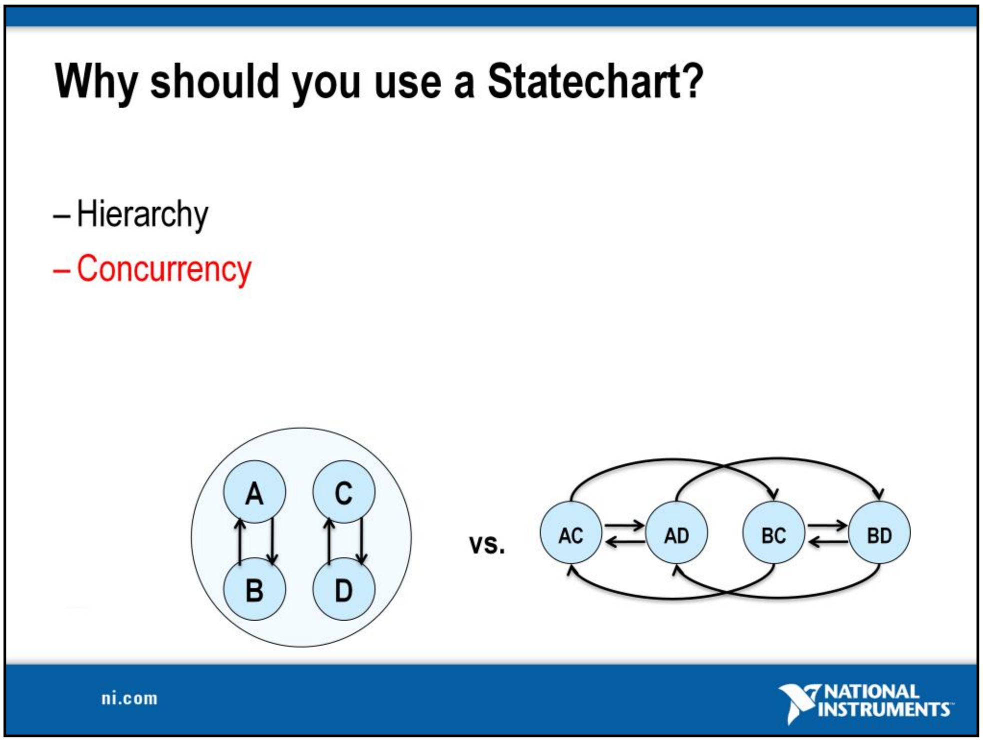
– Hierarchy

NATIONAL INSTRUMENTS

We talked about what the differences between an FSM and a Statechart – but how do these differences translate into benefits to your application compared to a finite state machine implementation?

Hierarchy –

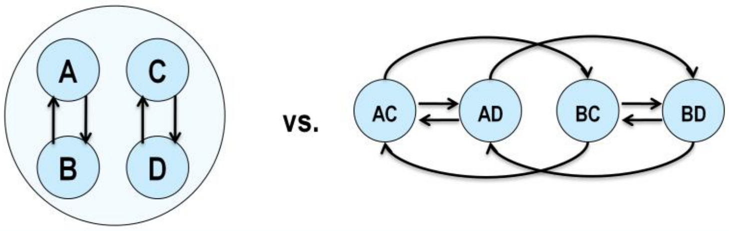
See here how hierarchy makes the flow of the program a lot clearer to the user. It eliminates many of the transitions required by a finite state machine representation, simplifying the implementation. This architecture is particularly useful for error, abort, emergency, and other preemptive situations.



Why should you use a Statechart?

– Hierarchy

– Concurrency

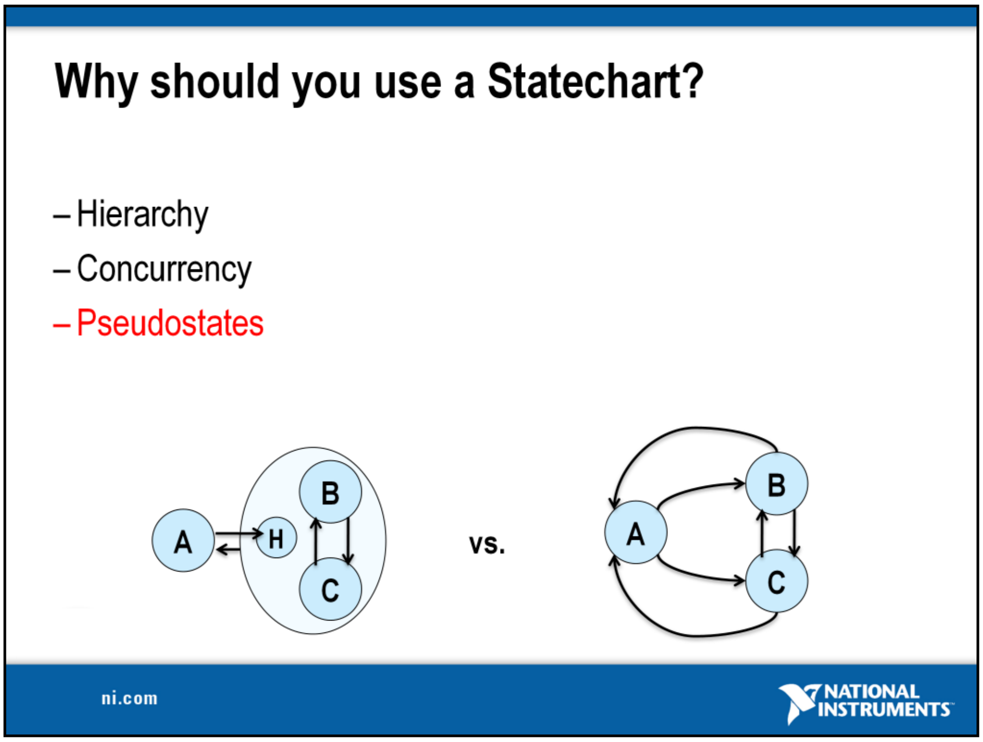


ni.com

NATIONAL INSTRUMENTS

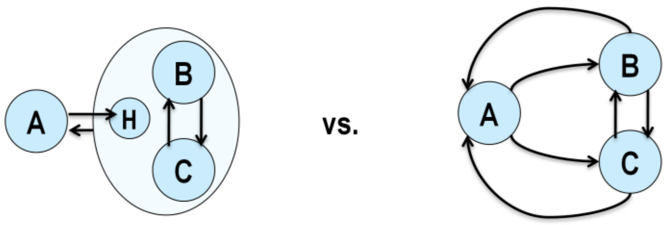
Concurrency –

This is very useful when separate logic needs to happen at the same time. Here you can see how the flow of the program is more obvious on a conceptual level.



Why should you use a Statechart?

- Hierarchy
- Concurrency
- Pseudostates

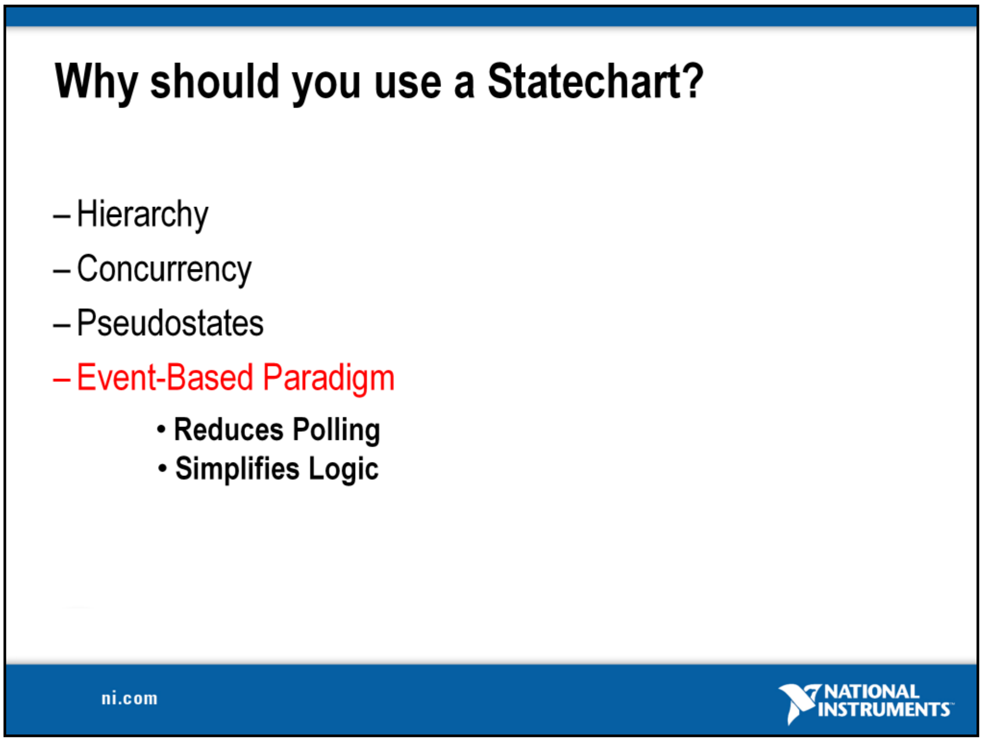


ni.com

NATIONAL INSTRUMENTS

Pseudostates –

Although we are not going to go in depth here in discussing all of the pseudostates that are defined by the UML specification. One in particular is very useful. The history pseudostate – will all users to go to the last state that a hierarchical state was in when it exited. This notation simplifies the representation by eliminating a good deal of transitions.



Why should you use a Statechart?

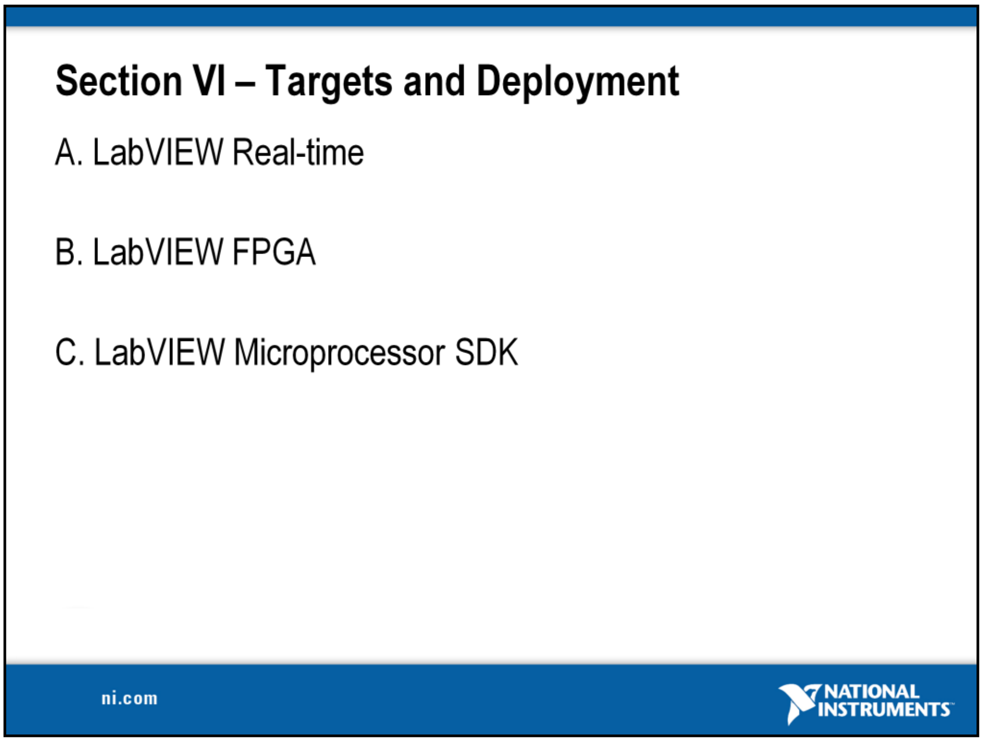
– Hierarchy
– Concurrency
– Pseudostates
– Event-Based Paradigm
  - Reduces Polling
  - Simplifies Logic

ni.com

NATIONAL INSTRUMENTS

Event Based –

Although it is hard to represent this with a graphic – one of the key benefits is that you can reduce the amount of polling that you do in your code. Particularly if you are dealing with a UI that receives events through the event structure.



Section VI – Targets and Deployment

A. LabVIEW Real-time

B. LabVIEW FPGA

C. LabVIEW Microprocessor SDK

ni.com

NATIONAL INSTRUMENTS