LabVIEW Hands-On Campus Workshop Seminar Manual

April 2011 Edition
Part Number 351286C-01

Worldwide Technical Support and Product Information
ni.com

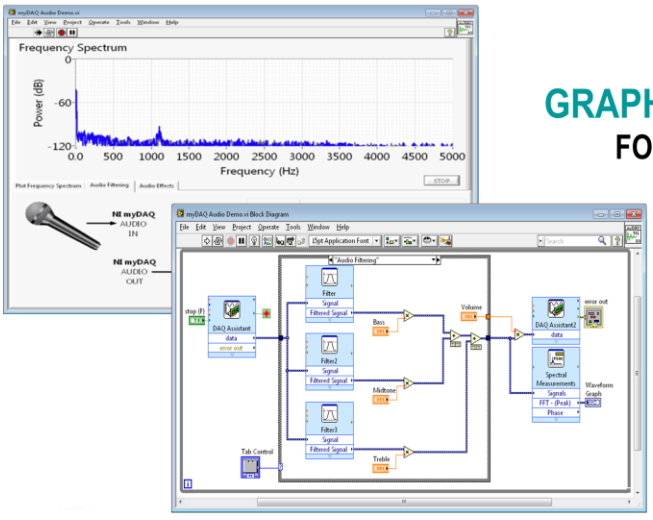
Worldwide Offices
Visit ni.com/global to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

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

Contents

Slides with Presenter Notes

Hands-On with LabVIEW

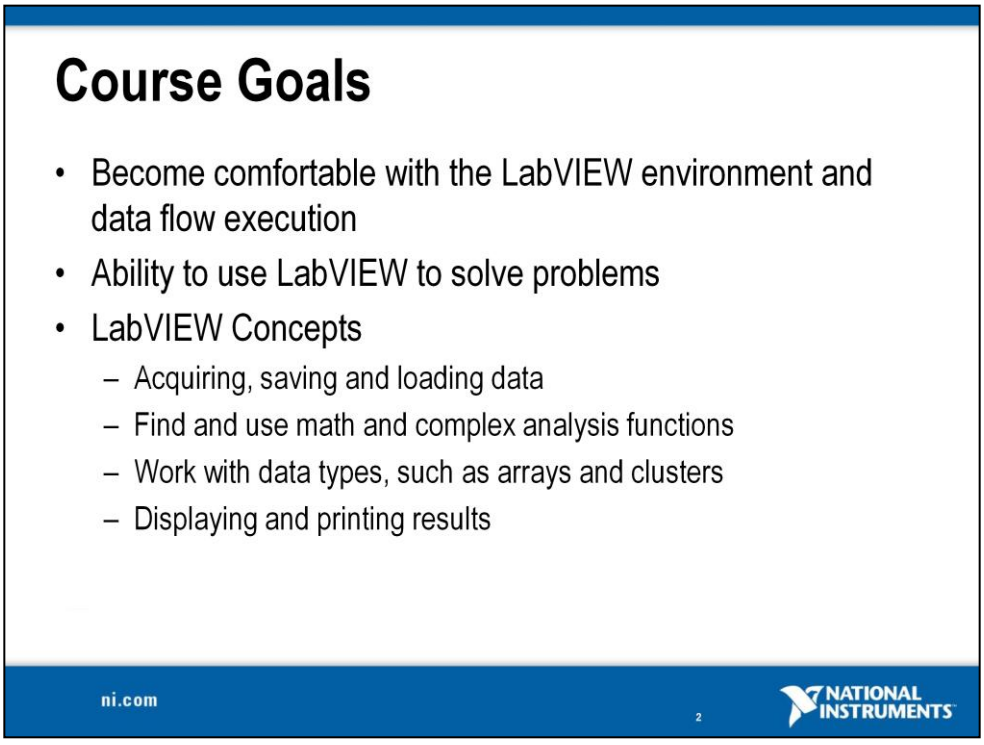


GRAPHICAL PROGRAMMING
FOR ENGINEERS AND SCIENTISTS

LabVIEW™

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, 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

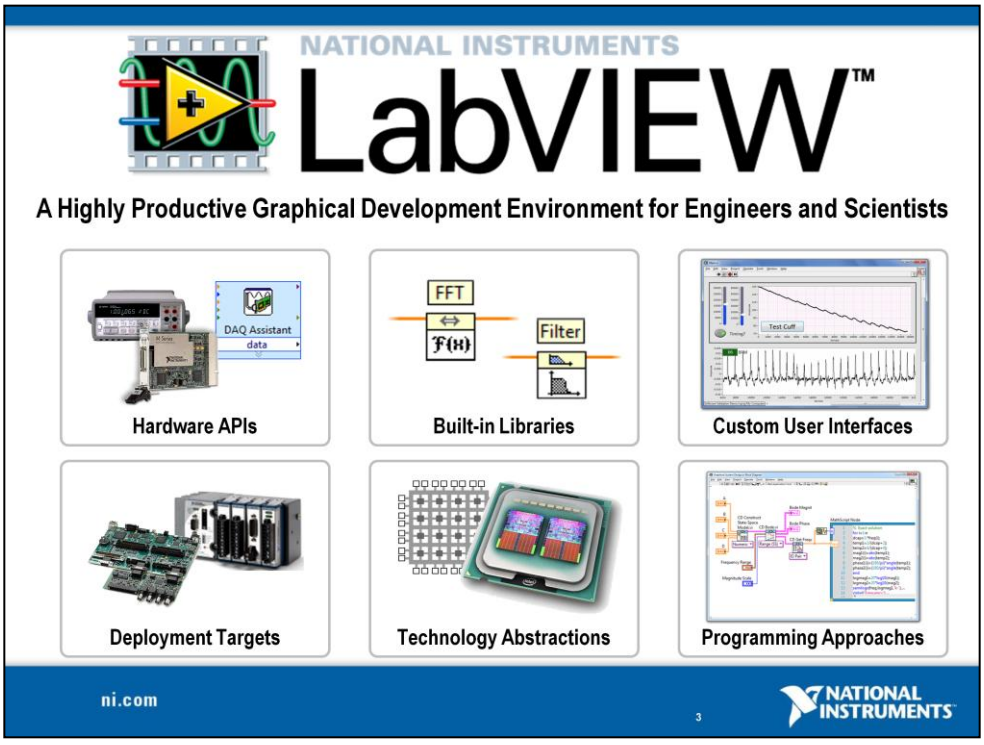
This course does *not* describe any of the following:

- 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.

For additional questions, refer to the *LabVIEW Help*:

You can access it by using the LabVIEW toolbar **Help.**



Definition

LabVIEW is a development environment that has been built specifically for engineers and scientists with the intent of making them more productive and ensuring that they have all the tools they need to prototype, design and build their applications.

How can we claim that LabVIEW makes you more productive?

LabVIEW makes users more productive because it provides all the tools engineers need in a single environment and ensures that they all work and can be used together. The key is guaranteed compatibility between engineering tools.

Technology Abstractions?

LabVIEW's compiler abstracts complex technological problems like multicore, virtualization, memory allocation and network communication.

Hardware APIs?

Over 8,000 drivers for instruments and a wide-range of USB, PCI, and PXI instruments make it easy to integrate real-world signals into software.

G Programming Language

G is a complete programming language, capable of solving the most complex and advanced problems today. There are a variety of other programming approaches in LabVIEW, but G is the language that ties them together.

Section I – LabVIEW Environment

A. Set up Your Hardware

B. Take Your First Measurement

- Open and Explore Final Project: Audio Equalizer

C. Demonstration: Creating our First VI

D. LabVIEW Environment

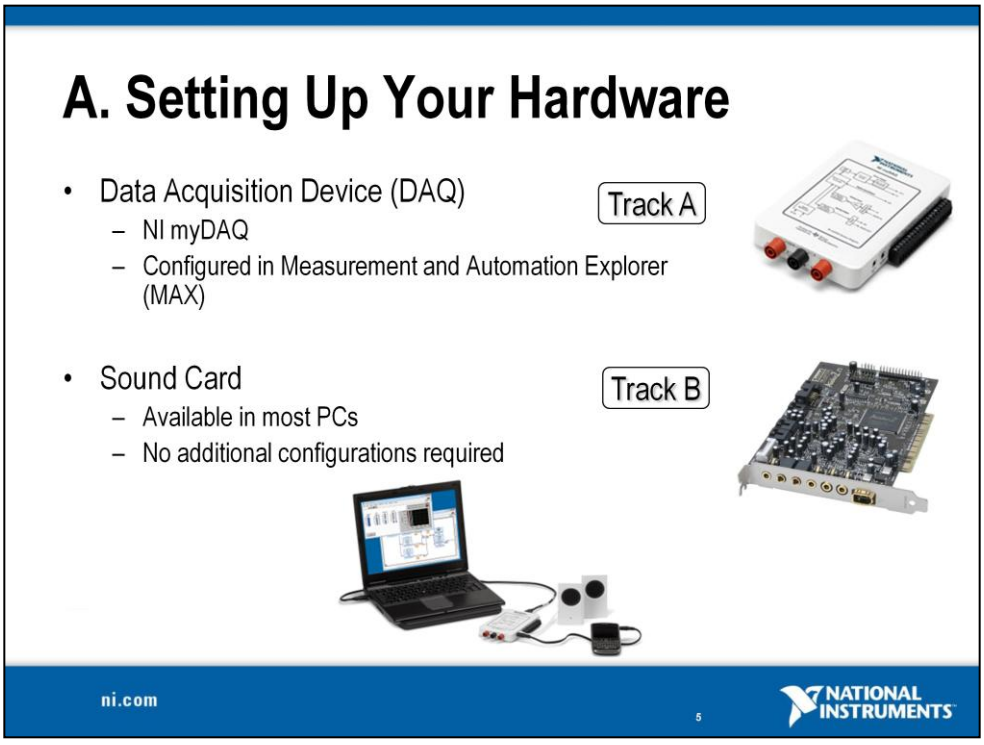
E. Hands-On Exercise: Acquiring Data

- Write a program that reads in a signal from a microphone

ni.com

4

NATIONAL INSTRUMENTS

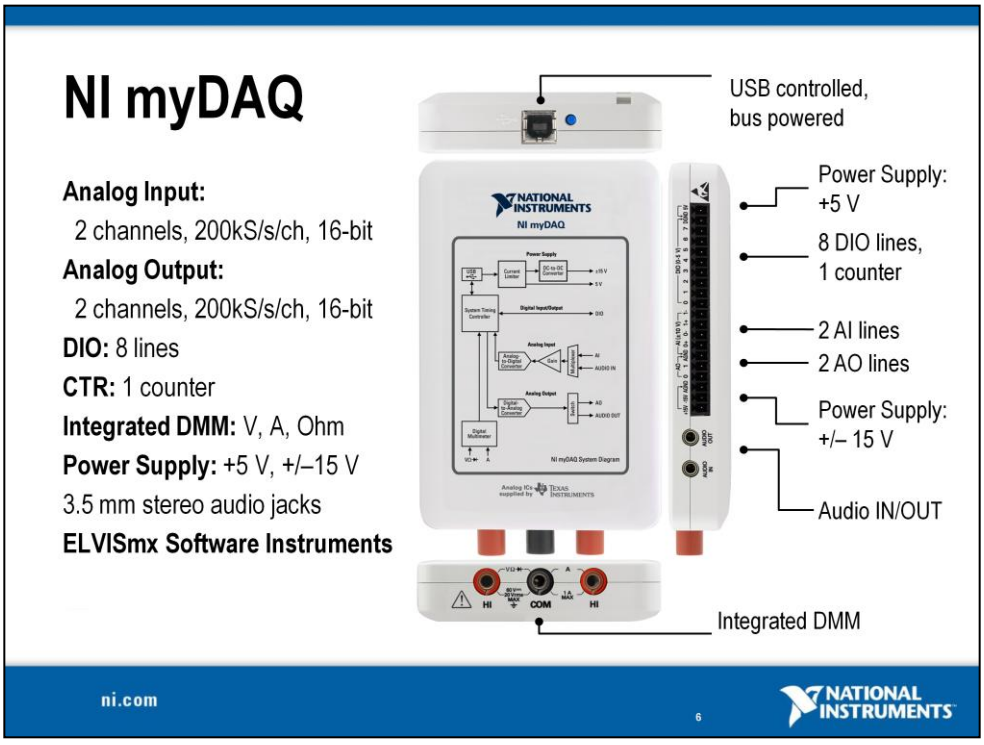


This LabVIEW course is designed for audiences with or without access to National Instruments hardware.

Each exercise is divided into two tracks, A and B:

Track A is designed to be used with the NI myDAQ, which is supported by the NI-DAQmx driver.

Track B is designed to be used with your sound card in your PC or laptop.



NI myDAQ is a low-cost portable data acquisition (DAQ) device that uses NI LabVIEW-based software instruments, allowing students to measure and analyze real-world signals. NI myDAQ is ideal for exploring electronics and taking sensor measurements. Combined with NI LabVIEW on the PC, students can analyze and process acquired signals and control simple processes anytime, anywhere.

NI myDAQ provides analog input (AI), analog output (AO), digital input and output (DIO), audio, power supplies, and digital multimeter (DMM) functions in a compact USB device.

Specifications include:

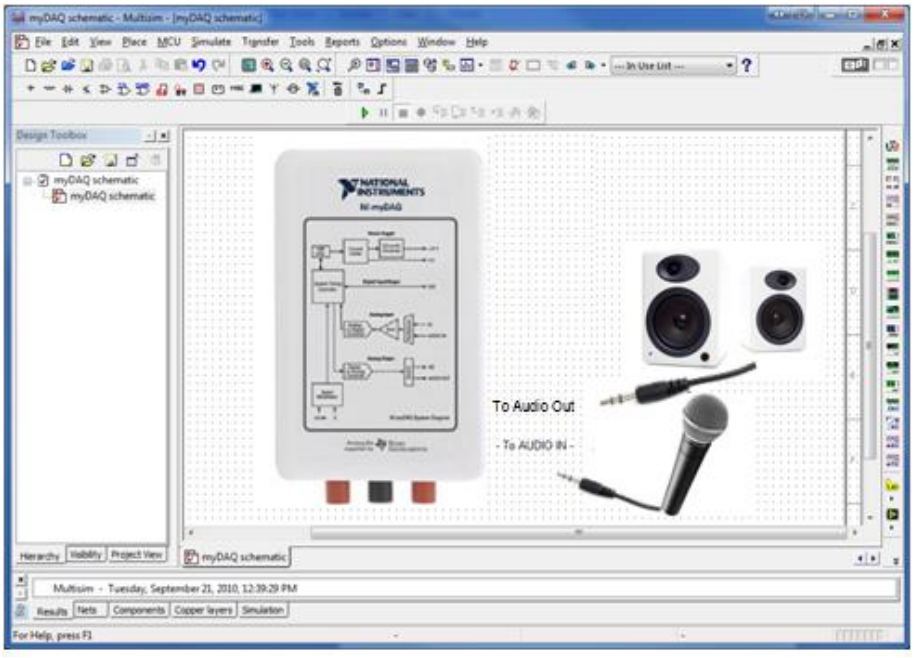
- Two differential analog input and analog output channels (200 ks/s,16 bit, +/– 10 V)
- Eight digital input and digital output lines (3.3 V TTL-compatible)
- +5 , +15, and –15 V power supply outputs (up to 500 mW of power)
- 60 Volt digital multimeter (DMM) for measuring voltage, current, and resistance

Setting Up Your Hardware for Your Selected Track

Track A – NI Data Acquisition with Microphone NI myDAQ, Microphone and speakers
Suggested Hardware

Qty	Description	Supplier
1	NI myDAQ	National Instruments
1	3.5mm Microphone	
1	Set of speakers or headphones	
1	MP3 Player (optional)	

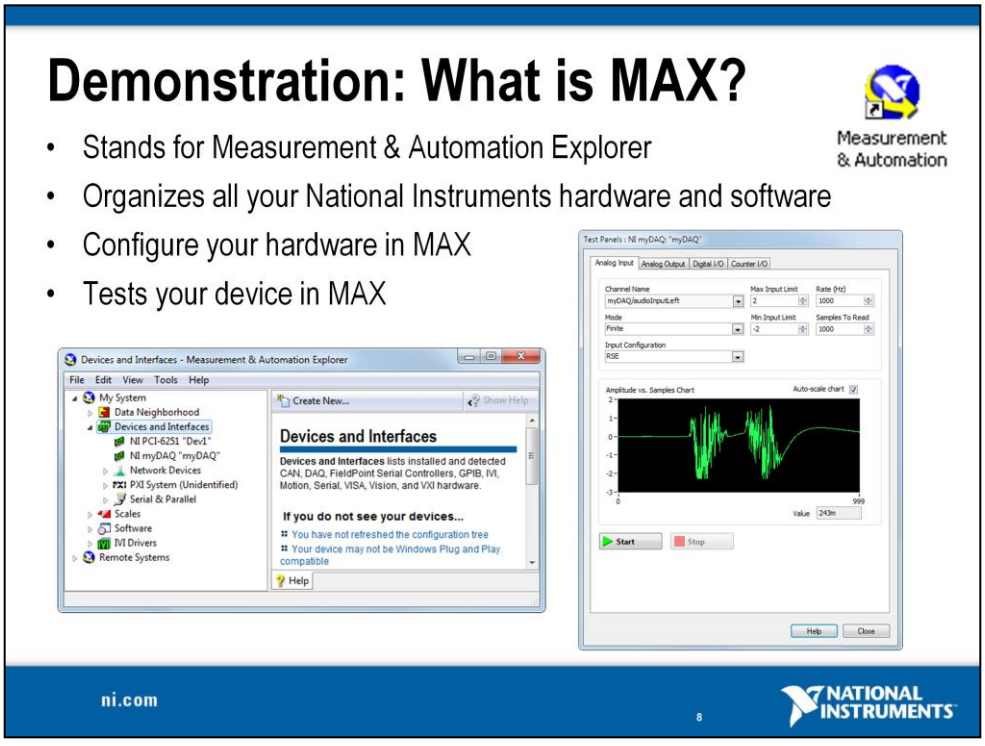
The following schematic was drawn with NI **Multisim**, a widely used SPICE schematic capture and simulation tool. Visit ni.com/Multisim for more info.



Track B – Using Your Soundcard A soundcard, microphone, and speakers
Suggested Hardware

Qty	Description	Supplier
1	3.5 mm Microphone OR	Fry's
1	3.5 mm male to male audio cable AND	Fry's
1	MP3 Player	
1	Set of speakers or headphones	Fry's

Connect the microphone to the audio input jack on your computer. OR
Connect the audio cable from an mp3 player to the audio input jack.
Connect the set of speakers to the audio output jack of your computer.



The next level of software 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 appears 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-DAQmx 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 falls into six categories:

- Data Neighborhood
- Devices and Interfaces
- IVI Instruments
- Scales
- Historical Data
- Software

This course will focus on Data Neighborhood, Devices and Interfaces, Software, and learning about the functionality each one offers.

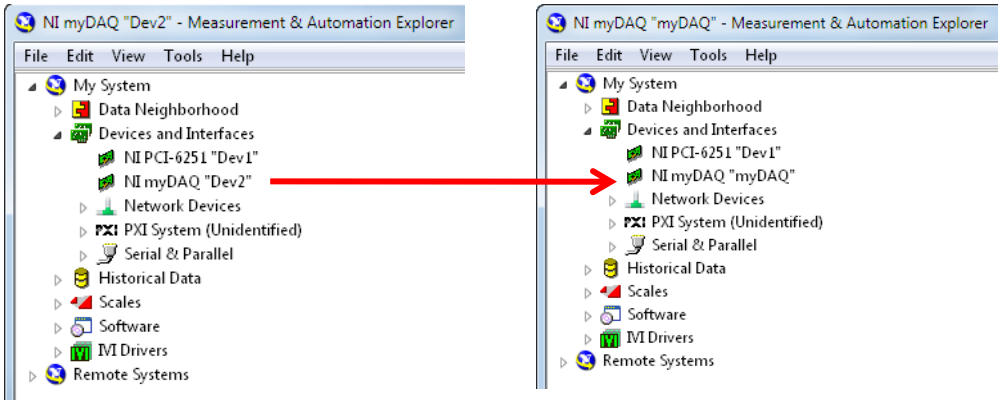
Exercise 1 – Configuring the myDAQ in MAX (Track A)

Goal Familiarize yourself with Measurement & Automation Explorer, and ensure that myDAQ is installed and functioning properly.

Description Before we can use the myDAQ in a LabVIEW program, we need to verify that it has been installed correctly and is working normally. This is done using the Measurement & Automation Explorer (MAX). In MAX, we will verify that the myDAQ appears with the correct Device Name, and open a Test Panel to ensure that signals can be read successfully.

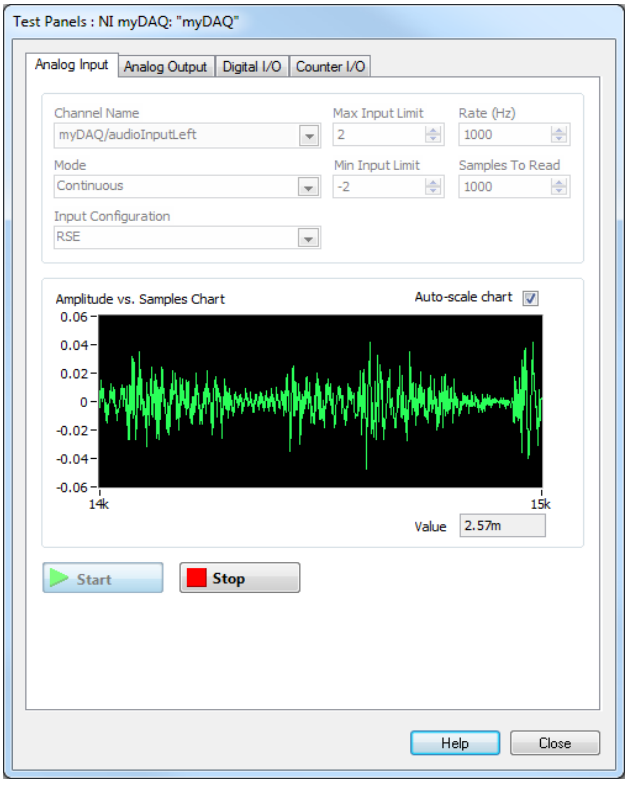
Procedure

1. Set up your myDAQ.
   a. Connect your headphones/speakers to the Audio Out terminal
   b. Connect your microphone to Audio In terminal OR
   c. If you have an mp3 player, connect the audio cable from your mp3 or phone to the Audio In terminal.
2. Launch Measurement & Automation Explorer (MAX) from **Start» All Programs»National Instruments»Measurement & Automation**.
3. Under **My System**, expand the entry for **Devices and Interfaces**
4. If the myDAQ has been installed, you should see an entry for **NI myDAQ** on the list, as shown below in the left image.



5. Change the name of your myDAQ. Your myDAQ may appear with a device name (which appears in quotes) like "Dev1" by default. Change the device name to **"myDAQ1"** by right-clicking on NI myDAQ and selecting **Rename**. This is shown in the right image above.
6. With the myDAQ selected, click the button labeled **Test Panels…** to open up the MAX Test Panel.

7. Under the **Analog Input** tab, choose myDAQ/audioInputLeft from the **Channel Name** list. Change the **Mode** to Continuous, and set the **Max Input Limit** and **Min Input Limit** to 2 and –2 respectively. Clicking the Start button should show data in the **Amplitude vs. Samples Chart**.
8. Try running the test panel again with the **Channel Name** set to myDAQ/audioInputRight. You should see similar results.



Why is it important to conduct this exercise before starting our LabVIEW program? Try running the test panel with the Max and Min limits set back to their default values (10 and –10). You should see an error; does this error message make sense?

End of Exercise 1 (Track A)

Exercise 1 – Setting Up Your Device (Track B)

In this exercise, use the Control Panel to test your soundcard to see if it works.

Set up your computer.

Connect your microphone to the Audio Input on your computer. If you have an audio cable and an mp3 player, connect the cable from the mp3 player to the Audio Input on your computer. Last, connect your speakers or headphones to the Audio Output on your computer.

Test your Sound Card

Option 1:

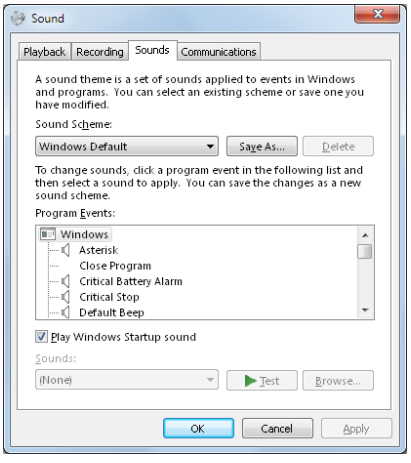
If you have music stored on your computer, play back a file.

Option 2:

Check the default sounds on your computer to see if they playback.

For Windows 7 Operating System do the following:

1. Navigate to the Control Panel.
2. Click on the **Hardware and Sound** category and the **Sound** icon and open it.
3. On the third tab, "Sounds", select a Program Event sound and click **Test**.

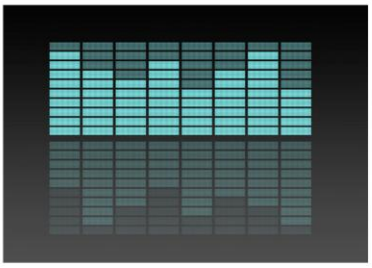


4. You should be able to hear a sound.

End of Exercise 1 (Track B)

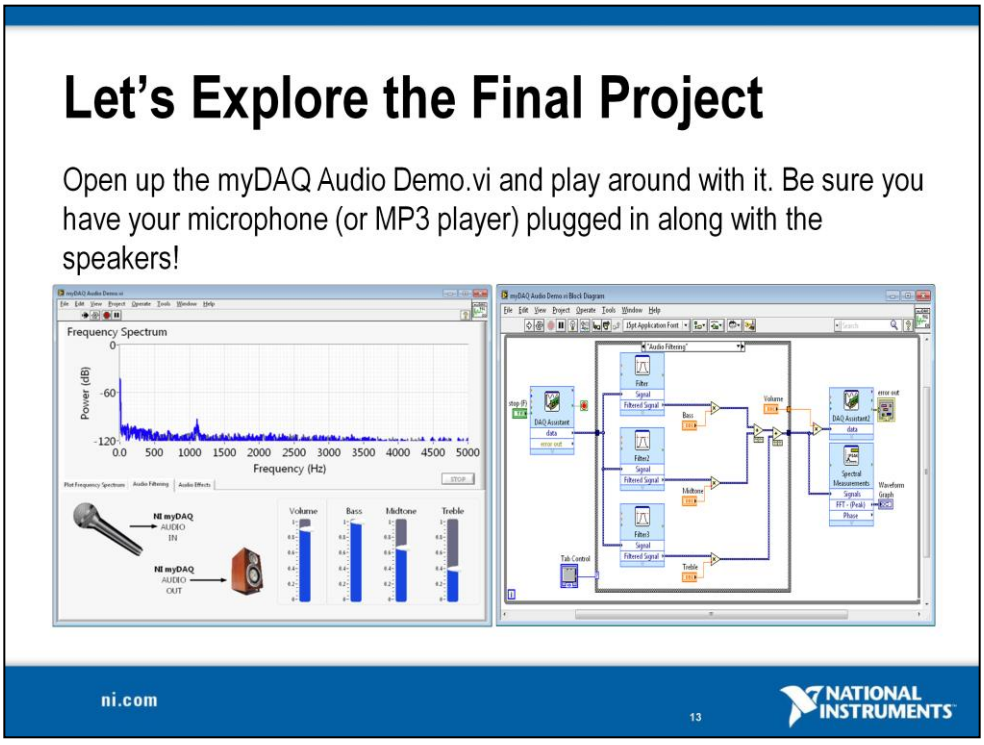
What is an Audio Equalizer?

- It's a car stereo!
- Audio Equalizers adjust the volume of a certain frequency within a signal.
- We are going to build a 3-band Audio Equalizer using filters that affects these frequency ranges
  - Bass
  - Midtones
  - Treble



ni.com 12 NATIONAL INSTRUMENTS

When was the last time you were in your car listening to the radio and then all of a sudden you started hearing a thumping sound because someone had the bass of their music maxed out? Music is composed of different pitches, or frequencies, and by increasing or decreasing the strength of certain pitches, you are manipulating the volume of the pitches. This is what we call an audio equalizer. In our application we will be building an audio equalizer that boosts or cuts bass, midtone and treble frequencies.

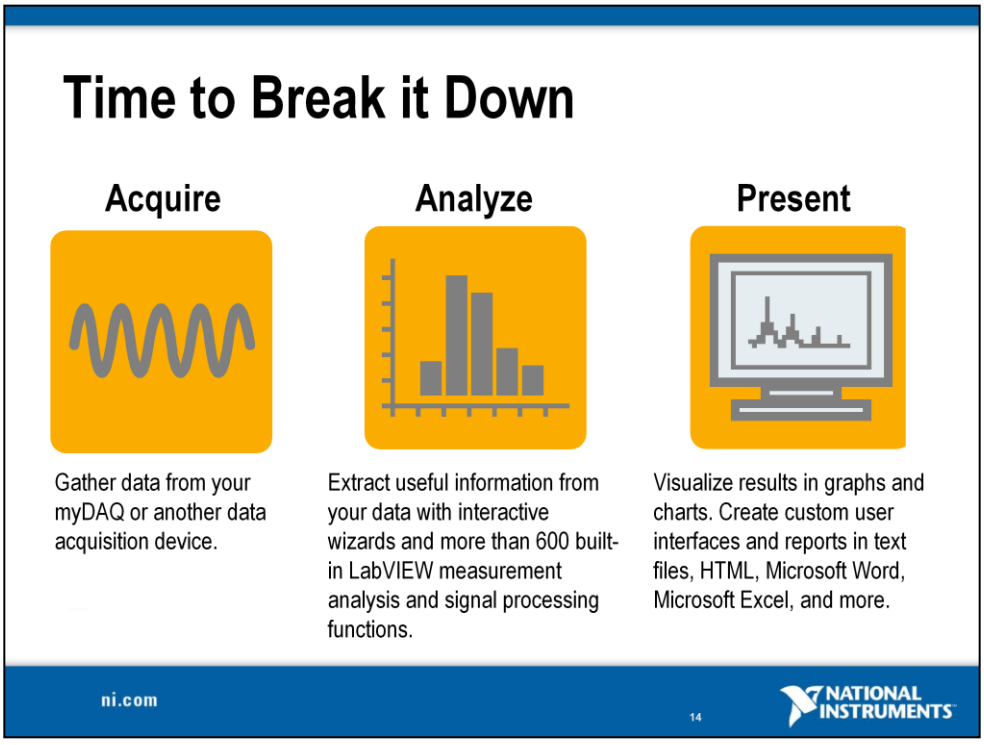


This is the application we will be building throughout this workshop. We will be breaking down each of the components of our audio equalizer and understanding what each aspect of this program is contributing to the final project. We will then rebuild the project taking time to explain the LabVIEW environment and helpful programming tips and tricks throughout this session.

You can find this file by navigating to the <Workshop VIs>\Audio Equalizer with xxx.vi. If you are using hardware, select **Audio Equalizer with Hardware.vi** and if you are using your sound card select **Audio Equalizer with Sound Card.vi**.

To run our first VI, double-click on the VI file and LabVIEW will open up. Take some time to explore the two windows that pop up. Navigate to the window with the gray back ground. In the upper left hand corner, click on the white arrow. This will start the program so you can explore the VI. After starting your VI, send audio to the program by either making sounds into the microphone or starting music on your mp3 player. You can modify the sound of the audio by clicking on the tabs and sliders to produce different effects and listen to the changes in your headphones/speakers!

Note: Make sure you completed Exercise 1 before attempting to run this VI.



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. In every LabVIEW application, there is a general pattern of first acquiring data, analyzing and processing the data, and then presenting the data in a report or on a graph.

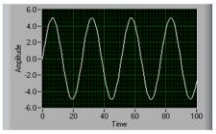
Acquire – With more than 50 million I/O channels sold in the last 10 years, NI is a global leader in PC-based data acquisition. LabVIEW provides a single programming interface resulting in seamless hardware and software integration.

Analyze – LabVIEW software has more than 600 built-in functions for signal synthesis, frequency analysis, probability, statistics, math, curve fitting, interpolation, digital signal processing, and more. You can also extend NI LabVIEW with application-specific processing for sound and vibration, machine vision, RF/communications, and more.

Present – After you acquire and perform analysis on your data, you likely need to present your data. Data presentation encompasses data visualization, report generation, data storage, Web publishing, database connectivity, data management, and more. The LabVIEW graphical development environment includes hundreds of built-in functions and tools for data presentation.

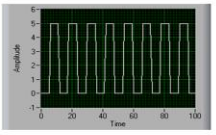
Types of Signals to Acquire

Analog



- Signal that varies continuously
- Most commonly a voltage or current
- In our case sound going into the microphone is converted to a voltage

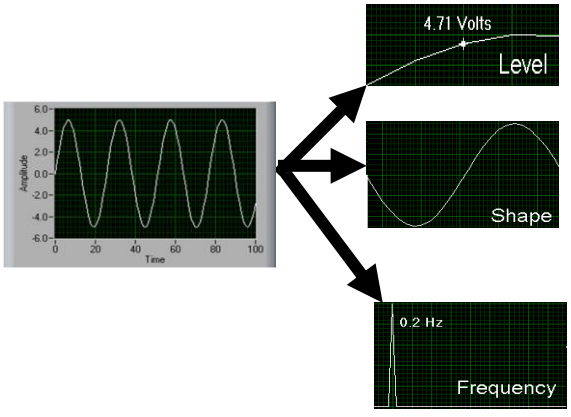
Digital



- Electrical signals that transfer digital data
- Usually represented by on/off, high/low, 0/1

ni.com 15 NATIONAL INSTRUMENTS

An analog signal can be at any voltage level with respect to time. You can measure the level, shape, and frequency of an analog signal.

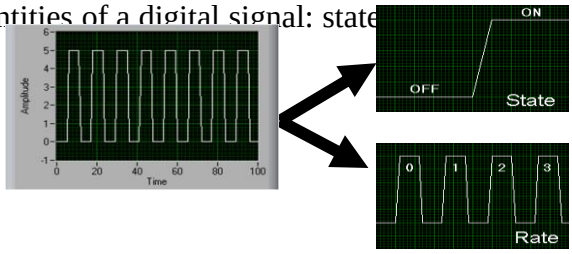


Level – Measuring the level of an analog signal can be at any voltage state.

Shape – Because analog signals can be at any state with respect to time, the shape of the signal is often important. For instance, a sine wave has a different shape than a sawtooth wave. Measuring the shape of a signal opens the door to further analysis on the signal itself such as peak values, slope, integration, etc.

Frequency – You cannot directly measure the frequency of an analog signal. Software analysis of the signal is required to extract the frequency information. The analysis is usually done by an algorithm called a Fourier Transform.

A digital signal has only two possible states: ON (high logic) or OFF (low logic).
An example of a digital signal is a TTL (Transistor-to-Transistor Logic) signal. The specifications for a TTL signal state that a voltage level between 0–0.8 Volts it is considered low logic, and a voltage level between 2–5 Volts is considered high logic. Most digital devices in industry accept a TTL compatible signal. Since a digital signal only has two states, we can only measure two quantities of a digital signal: state

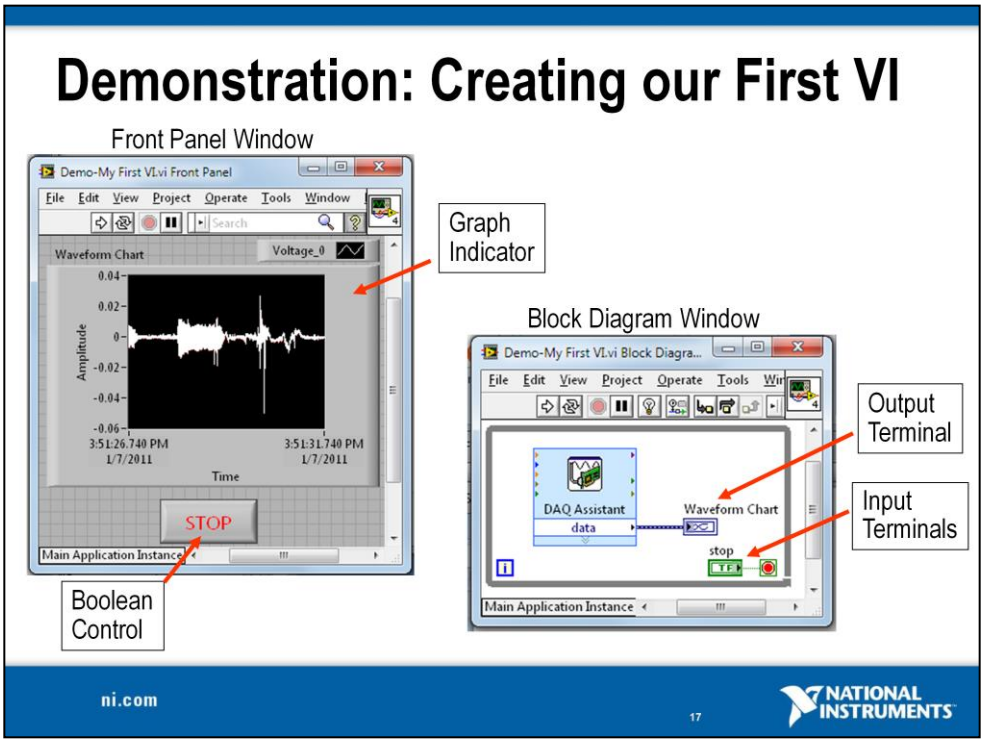


State

A digital signal only has two possible states: ON or OFF. Thus one of the quantities of a digital signal we can measure is whether the state is ON or OFF.

Rate

A digital signal also changes state with respect to time. Therefore, the other quantity of a digital signal we can measure is the rate, or in other words how the digital signal changes states with respect to time.

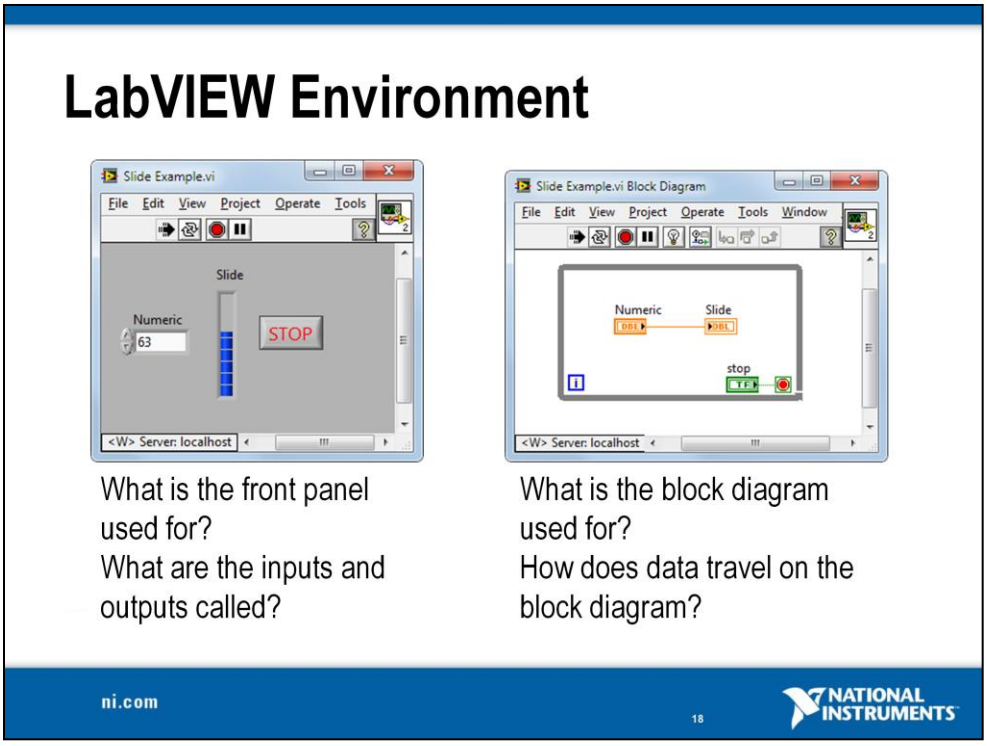


When you create an object on the front panel, a terminal is created on the block diagram. These terminals give you access to the front panel objects from the block diagram code.

Each terminal contains useful information about the front panel object it corresponds to. Such as, the color and symbols providing 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 T/F lettering.

In general, blue terminals should wire to blue terminals, green to green, and so on. This is not a hard-and-fast rule; you can use LabVIEW 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 a thick border and an arrow on the right side. Indicators have a thin border and an arrow on the left side. 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).



LabVIEW programs are called virtual instruments (VIs).

Controls are inputs and indicators are outputs.

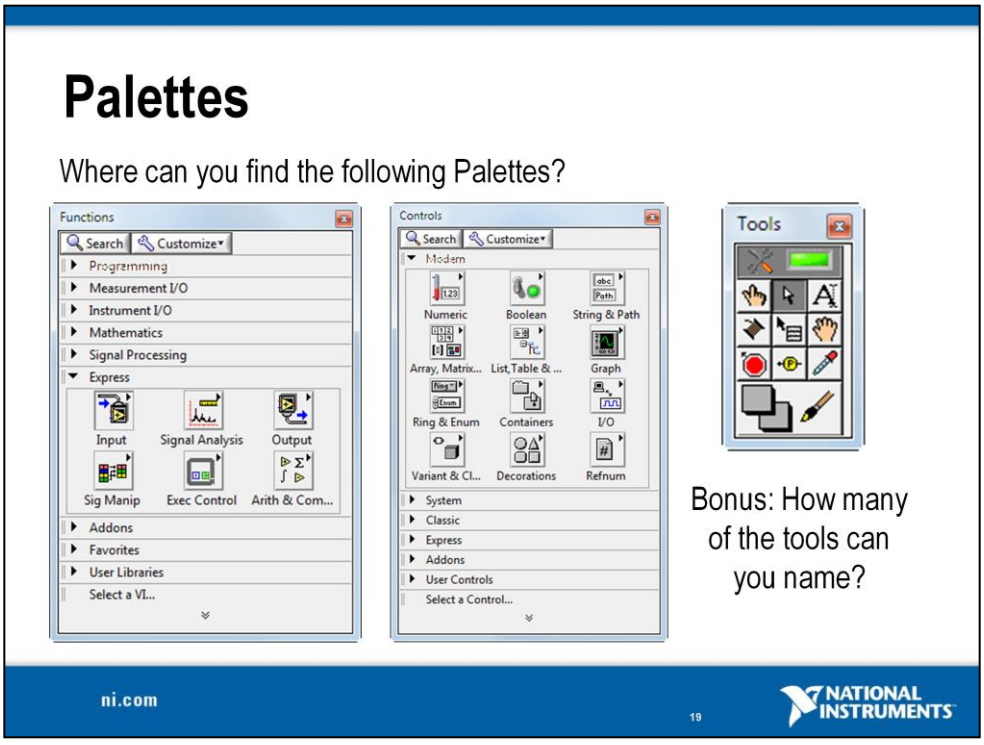
Each VI contains three main parts:

- Front panel – How the user interacts with the VI
- Block diagram – The code that controls the program
- Icon/connector – The 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.

You interact with the front panel when the program is running. You 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 stopping 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 you run a VI, 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 **View»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.

Use the Functions palette to build the block diagram. The Functions palette is available only on the block diagram. To view the palette, select **View»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.

You can view the Tools palette on both the front panel and block diagram. To view the palette, select **View»Tools Palette**. You also can display the Tools palette by holding shift+ right-clicking an open area on the front panel or block diagram.

If you enable the automatic selection tool 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 selection tool by clicking the **Automatic Selection Tool** 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.

Use the **Object Shortcut Menu tool** to access an object shortcut menu with the left mouse button.

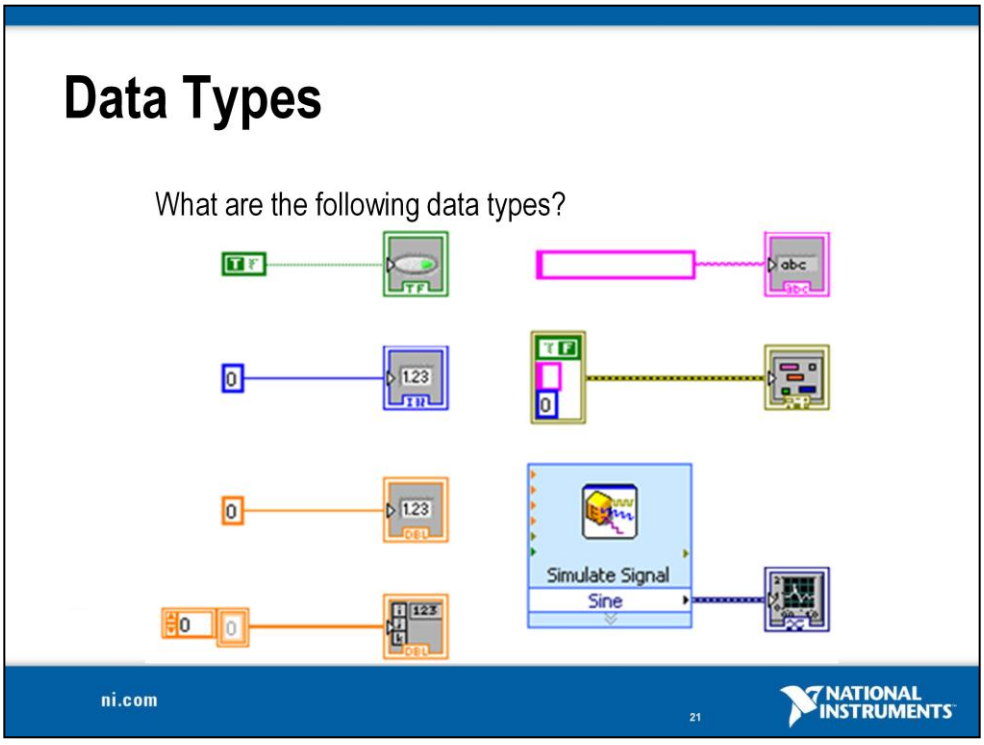
Use the **Scrolling tool** to scroll through windows without using scrollbars.

Use the **Breakpoint tool** to set breakpoints on VIs, functions, nodes, wires, and structures to pause execution at that location.

Use the **Probe tool** to create probes on wires on the block diagram. Use the Probe tool to check intermediate values in a VI that produces questionable or unexpected results.

Use the **Color Copy tool** to copy colors for pasting with the Coloring tool.

Use the **Coloring tool** to color an object. The Coloring tool also displays the current foreground and background color settings.



Controls and indicators are front panel items that you can use to interact with your program to provide input and display results. You can access controls and indicators by right-clicking the front panel.

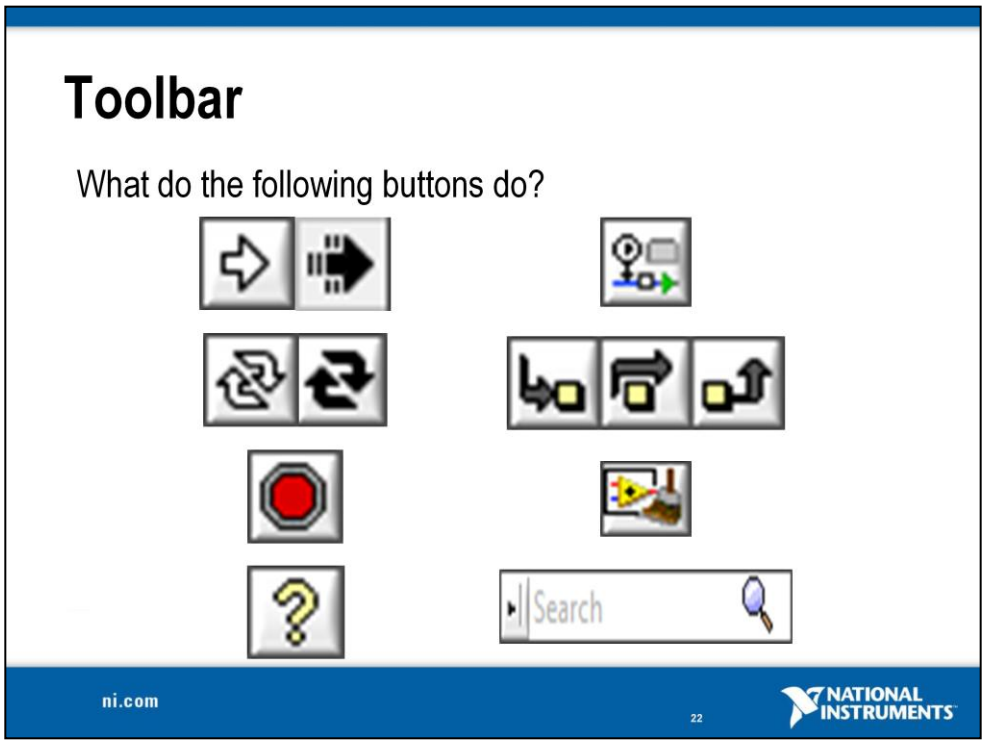
LabVIEW uses many common data types such as Boolean, numeric, integers, strings, clusters, arrays, and so on.

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 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.

The *LabVIEW User Manual* on ni.com provides additional references for data types found in LabVIEW.



Run: 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.

Continuous Run: 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. The Abort Execution button stops the VI immediately, before the VI finishes the current iteration. Aborting a VI that uses external resources, such as external hardware, might leave the resources in an unknown state by not resetting or releasing them properly. Design VIs with a stop button to avoid this problem.

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.

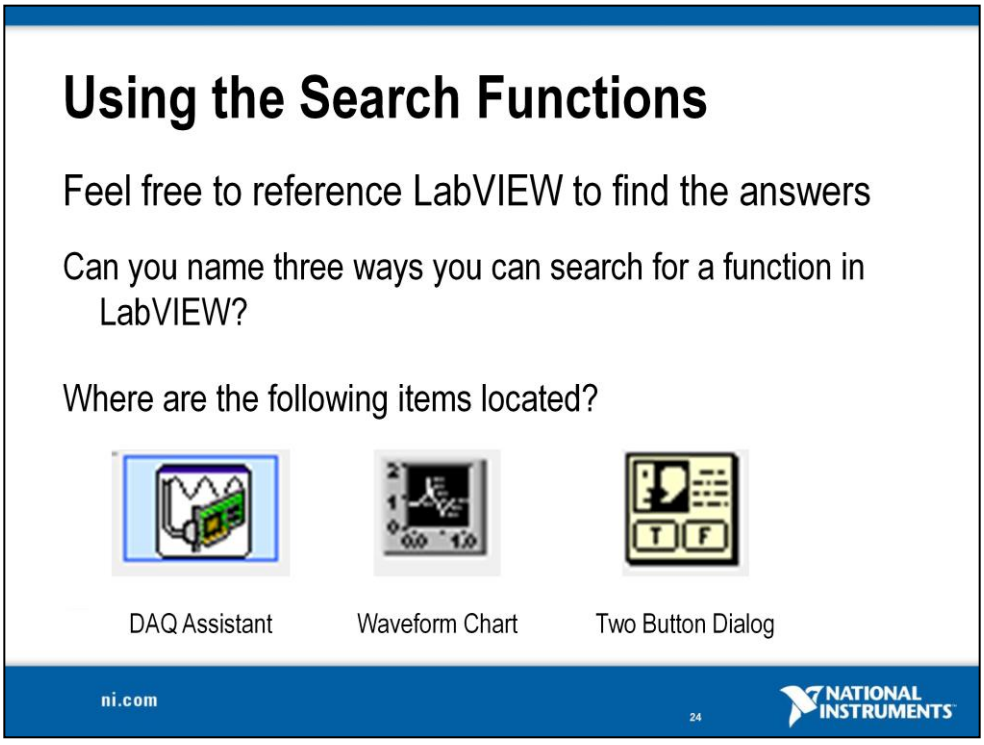
Select the **Align, Distribute and Resize Objects** pull-down menu to align objects along axes, space objects evenly, and change the width and height of objects, respectively.

Select the **Show Context Help** Window button to toggle the display of the **Context Help** window.

Click the **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 the wire you can immediately retain the most recent value of the data that passed through the wire.

Select **Step Into, Step Over, and Step Out** to debug your program step by step. Each node will blink to denote when it is ready to execute.

Click the **Clean Up Diagram** button to automatically reroute all existing wires and rearrange objects on the block diagram to generate a cleaner layout.

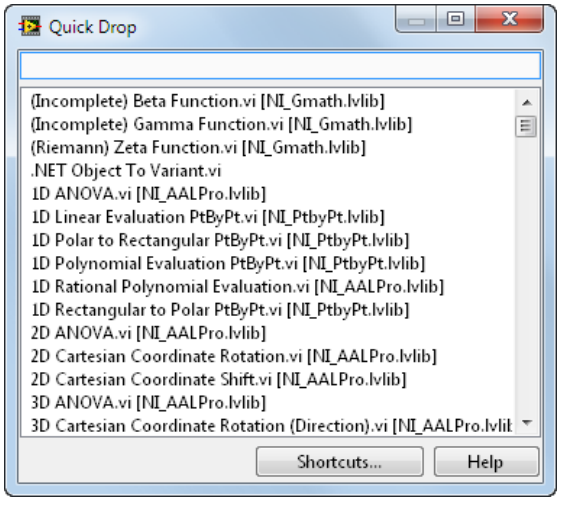


Use the buttons on top of the palette windows to navigate, search, and edit the palettes.

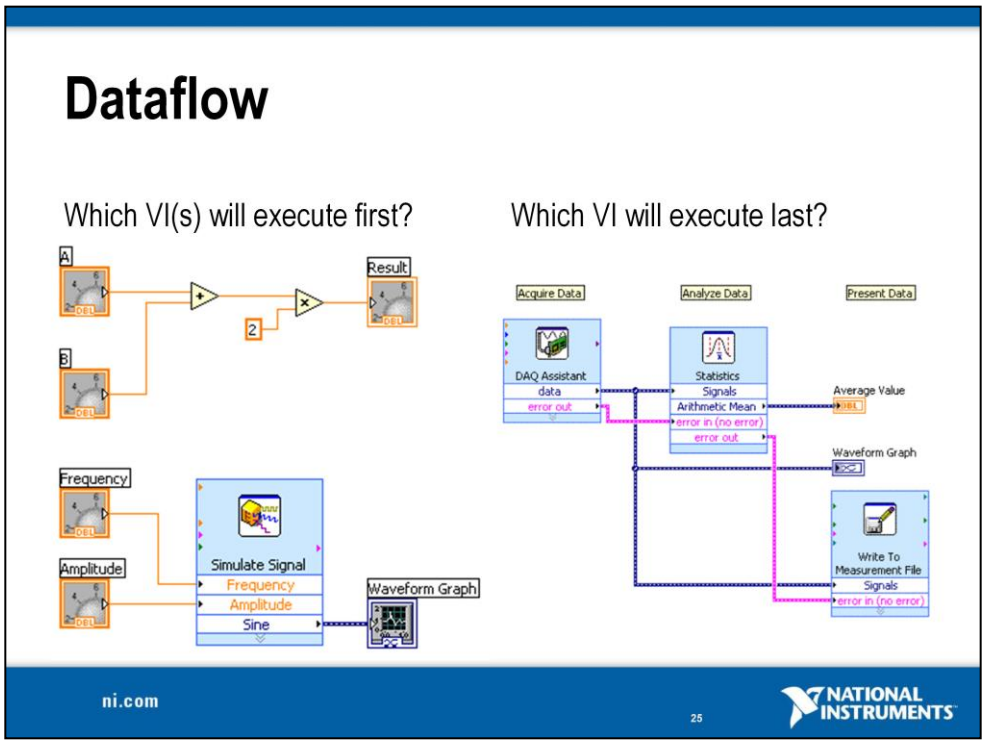
You can search for controls, VIs, and functions that either contain certain words or start with certain words. Double-clicking a search result opens the palette that contains the search result. You also can click and drag the name of the control, VI, or function directly to the front panel or block diagram. You can use the Search Bar in the toolbar on the front panel and block diagram, the search in the function and control palettes and quick drop to find functions.

Quick Drop

Select **View»Quick Drop** to display the Quick Drop dialog box. You also can press the <Ctrl-Space> keys to display this dialog box.



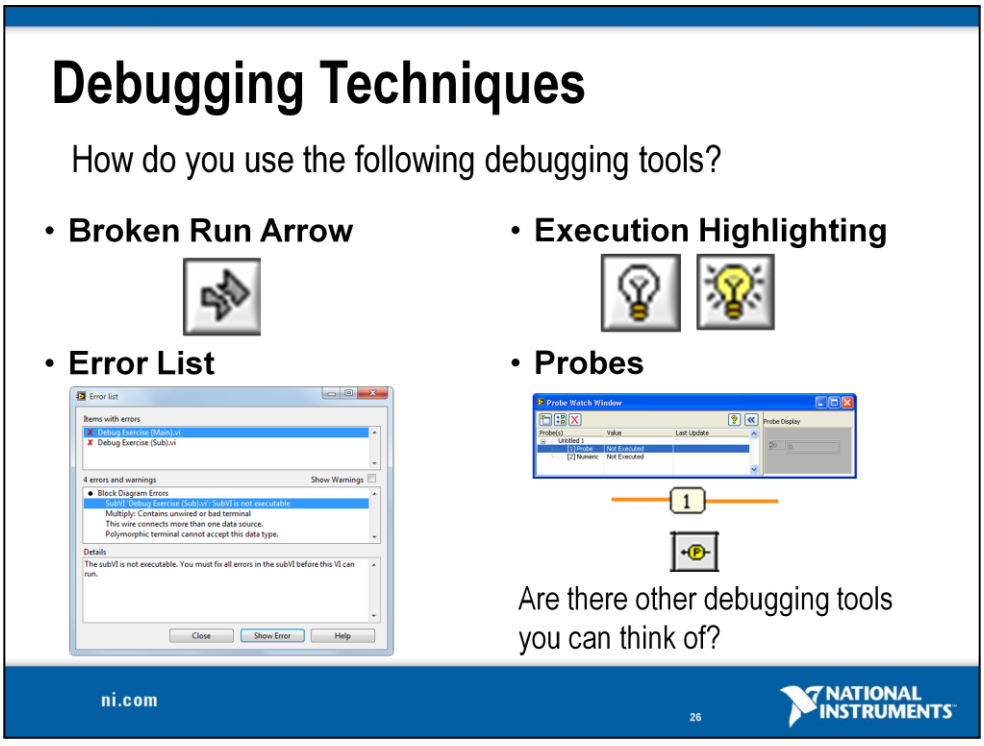
Use this dialog box to specify a palette object or project item by and then place the object on the block diagram or front panel. Press the <Enter> key, double-click the name of the object in the search results text box, or click the block diagram or front panel to attach the object to the cursor. Then click the location on the block diagram or front panel where you want to add the object.



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.

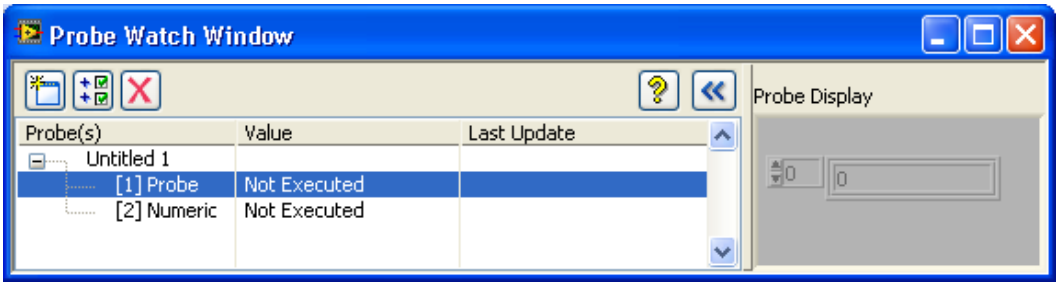
Consider the first block diagram above. It adds two numbers and then multiplies by 2 from the result of the addition. In this case, the block diagram executes from left to right, not because the objects are placed in that order but because one of the inputs of the Multiply function is not valid until 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 coexist on the same block diagram in parallel. This means that they begin executing at the same time and run independently of one another. If the computer running this code had multiple processors, these two pieces of code could run independently of one another (each on its own processor) without any additional coding.

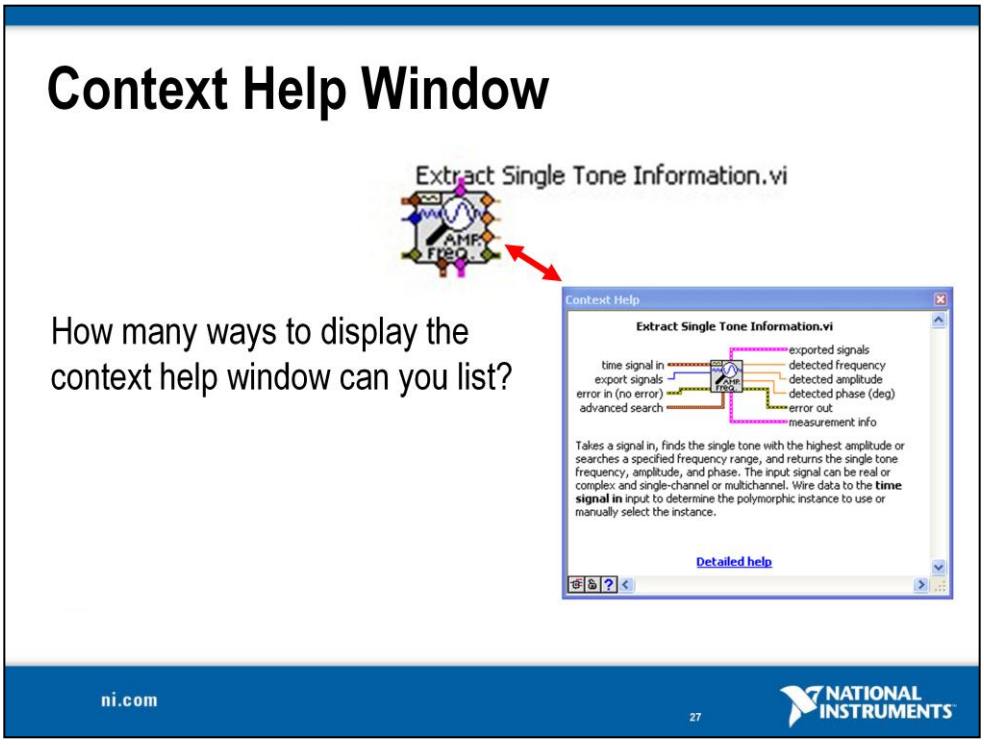


When your VI is not executable, a broken arrow is displayed in the **Run** button in the palette. Use our debugging tools to find your problem and correct it.

- **Finding Errors**: To list errors, click on the broken arrow. To locate the source of the error, 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**: The Probe Watch Window shows all of your probes, even those from subVIs. From this central location you can monitor values and see when data was last updated. If you see a suspect value, double clicking on the value in the Probe Watch Window will highlight the wire on your block diagram.



- **Retain Wire Values**: Used with probes to view the values from the last iteration of the program.
- **Breakpoint**: Sets pauses at different locations on the diagram. Click on wires or objects with the **Breakpoint** tool to set breakpoints.



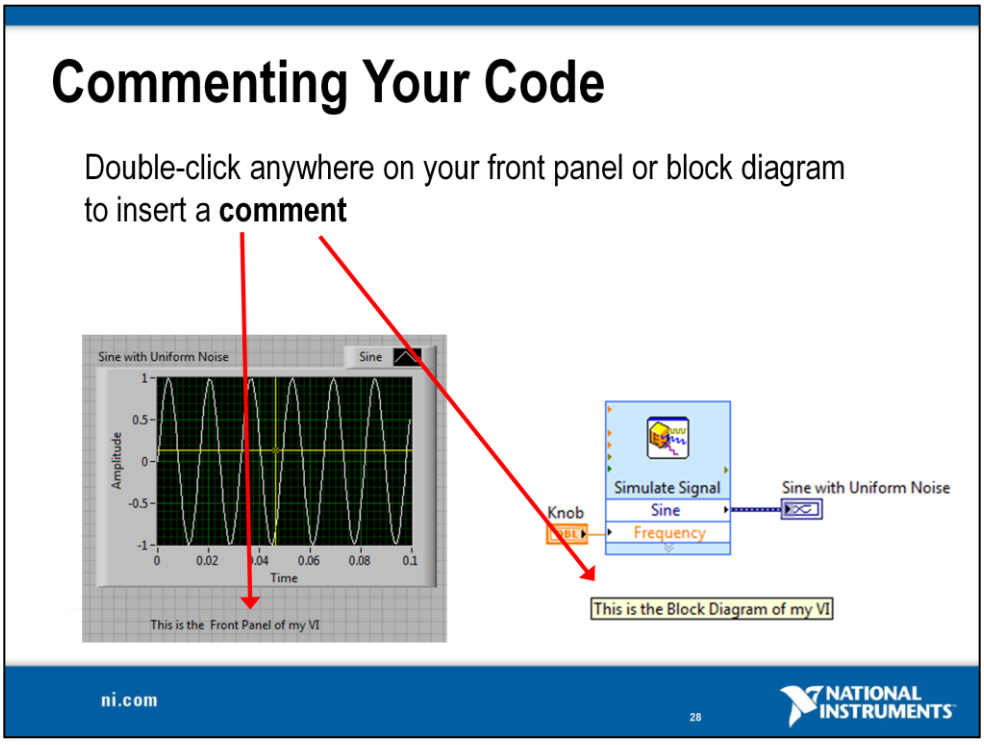
The **Context help** window displays basic information about LabVIEW objects when you move the cursor over each object. Objects with context help information include VIs, functions, constants, structures, palettes, properties, methods, and events. To display the **Context help** window, select **Help»Show Context Help**, press the <Ctrl-H> keys, or press the **Show Context Help Window** button in the toolbar. For any item on the Front Panel or Block Diagram you can display the LabVIEW help by right-clicking and selecting **Help**.

The connections for the VI are displayed in Context Help. There are different indicators that represent what inputs needs to be wired for the VI to work. You cannot set the connection property of the outputs of a VI.

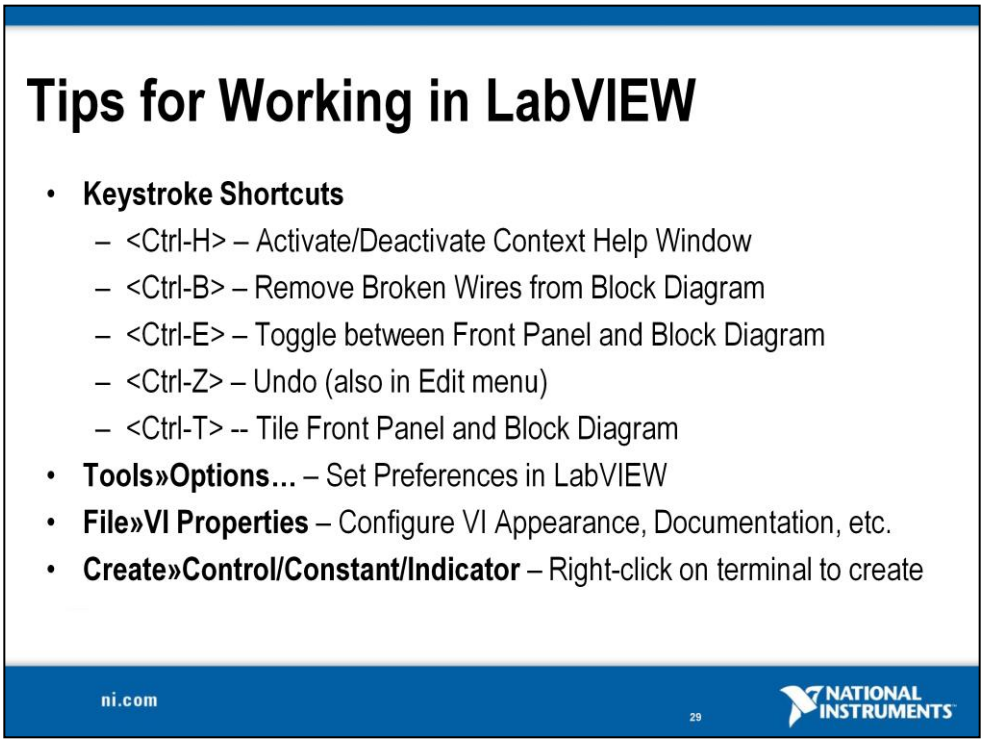
Required – bold	VI will be broken if you do not wire the required inputs.
Recommended – normal	VI can execute if you do not wire the recommended or optional terminals .
Optional – dimmed	If these inputs are not wired, the VI does not generate any warnings.

LabVIEW Example Finder

LabVIEW features hundreds of example VIs you can use and incorporate into VIs that you create. In addition to the example VIs that are shipped with LabVIEW, you 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.



Even within graphical programming environments, comments make code more understandable and easier to debug. Within the LabVIEW environment, inserting comments is as simple as double-clicking on empty space and typing. Comments can be created for both front panels and block diagrams.

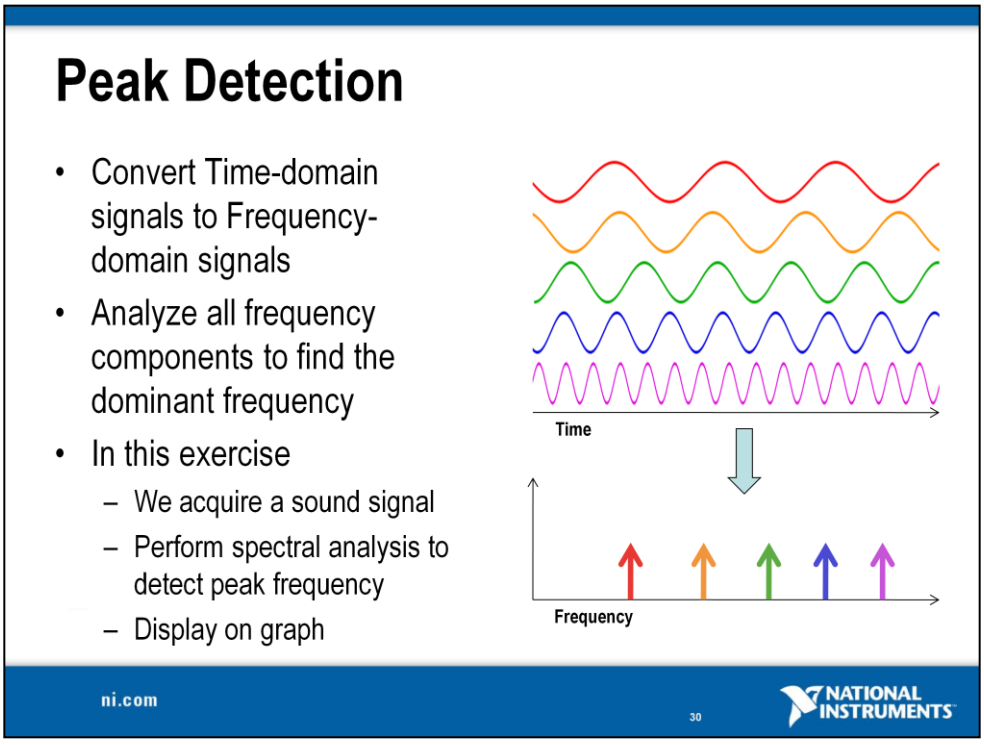


LabVIEW has many keystroke shortcuts that make working easier. The most common shortcuts are listed above.

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 chose, 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 so on.

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. In the block diagram, right-click on terminals and select C**reate» Control/Constant/Indicator** to make a control, constant, or indicator for the block diagram.



A time-domain graph shows how a signal changes over time. The data we acquire from our myDAQ is plotted versus time. However, since sound is a series of frequencies between the range of 20 to 20,000 Hz, displaying the data on an axis relating to frequencies will help us to visualize and analyze the information.

In our exercise we will use a Spectral Measurements Express VI to convert the Time-domain signal to the Frequency-Domain. To measure the peak, the VI will analyze all of the frequencies our microphone is picking up and display them on our graph. As the frequency changes its range, the peak will shift accordingly on the frequency access.

Exercise 2 – Capturing Sound with myDAQ (Track A)

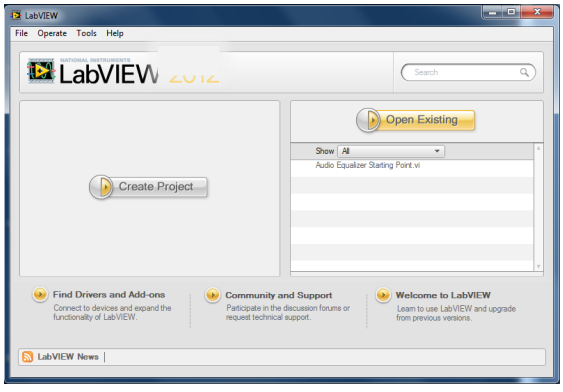
Goal Acquire data in LabVIEW with myDAQ and present it on the user interface.

Part 1

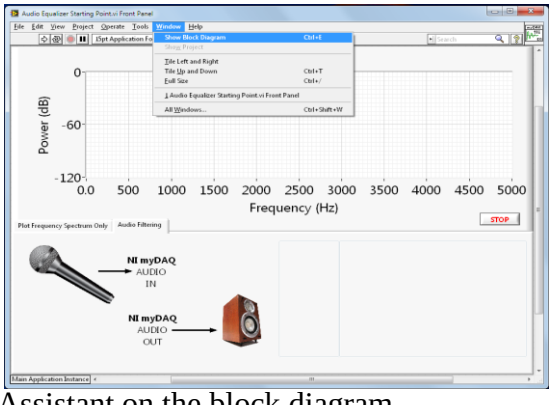
Description We have a microphone connected to the audio input of the myDAQ. In this exercise, we're going to write a LabVIEW VI that reads the audio signal from the myDAQ and displays the Frequency Band of that signal.

Procedure

1. Launch LabVIEW
2. In the Getting Started window, select **Open Existing**, and navigate to **Audio Equalizer Starting Point.vi**.



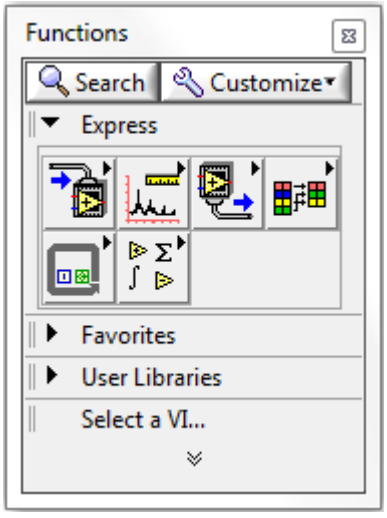
3. You will see that the major elements of the front panel have already been created for you. There is a Waveform Graph, a Stop button, and a Tab Control; we will add more items to the front panel in the next exercise.
4. Open the block diagram by selecting **Window»Show Block Diagram** or by pressing <Ctrl-E>. You can see the terminals that correspond to the three existing front panel objects.



5. Place a DAQ Assistant on the block diagram.
   a. Right-click on the block diagram to open the Functions Palette.

b. Select the DAQ Assistant by navigating to **Express»Input»DAQ Assistant**

c. Click on the block diagram to place the DAQ Assistant.

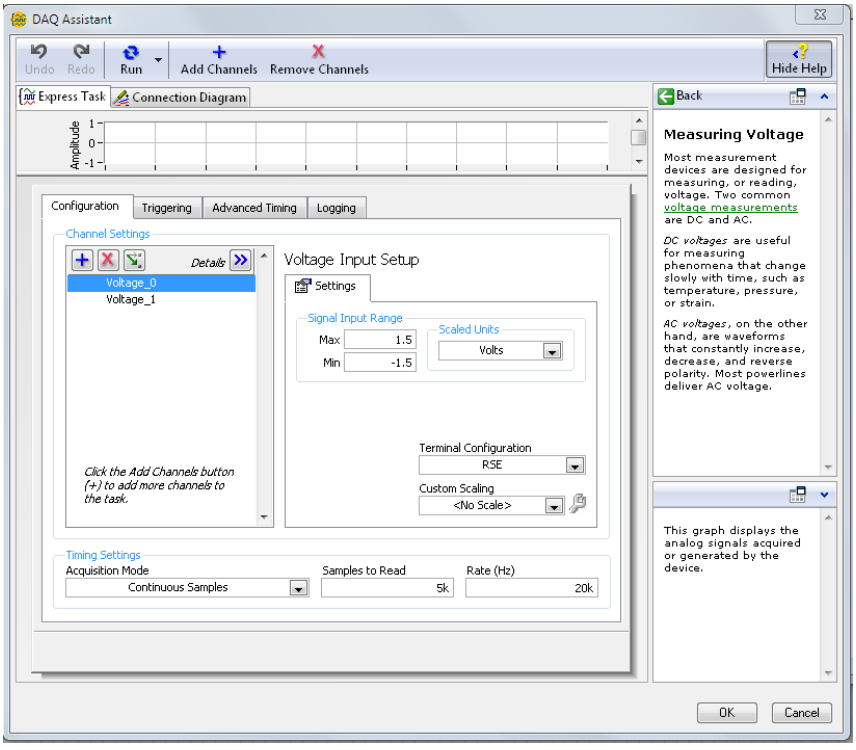


6. Configure the DAQ Assistant.

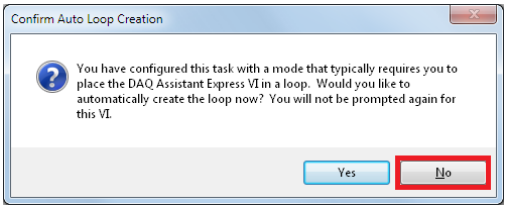
   a. The Create New Express Task window will appear. If it does not, double-click on the DAQ Assistant icon. Navigate to **Acquire Signals»Analog Input»Voltage.**

   b. In the next window, expand the myDAQ entry and select audioInputLeft and audioInputRight (hold down the Shift key to select multiple channels). Click **Finish**.

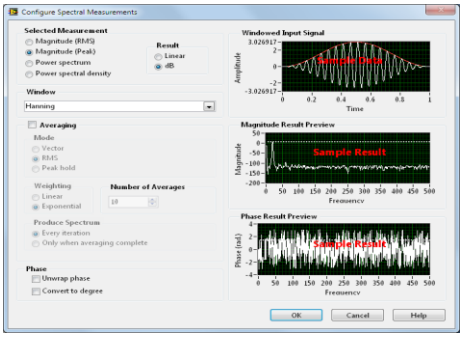
   c. The DAQ Assistant window appears. In the Signal Input Range area, change the **Max** and **Min** to 1.5 and –1.5 Volts. ***Make sure to do this for both channels (Voltage_0 and Voltage_1)***. Under **Timing Settings**, change the Acquisition Mode to **Continuous Samples**, set **Samples to Read** to 5k, and **Rate (Hz)** to 20k. Your window should match the one shown on the next page.



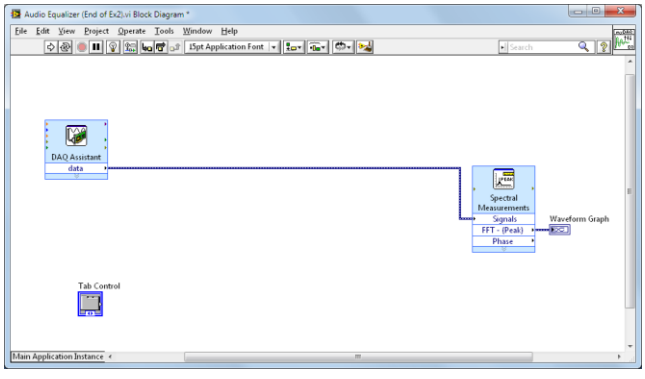
d. When finished, click **OK** and wait for the function to build. When the build is complete, you may see the **Confirm Auto Loop Creation** prompt. Should this occur, select **No**.



7. Place a Spectral Measurements function on the block diagram.
   a. Create a Spectral Measurements function by navigating to **Express» Signal Analysis»Spectral** in the Functions palette. Place the function on the block diagram.
   b. In the **Configure Spectral Measurements** window, change the **Selected Measurement** to Magnitude (Peak). Leave all other default settings unchanged. Click **OK**.



8. Connect the functions and test the VI.
   a. Wire the **data** terminal of the DAQ Assistant to the **Signals** input of the Spectral Measurements.
   b. Wire the **FFT – (Peak)** output of the Spectral Measurements to the input terminal of the Waveform Graph.
   c. Return to the front panel and click the **Run** button (the white arrow in the upper left corner).



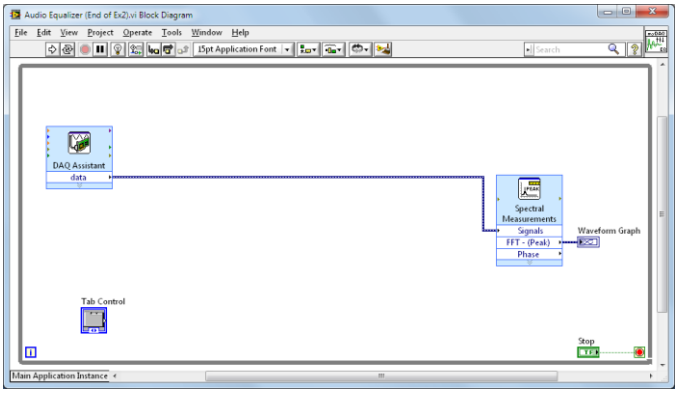
What did you see when you clicked the Run button? Try running the VI multiple times while making different sounds into the microphone. Do the results on the graph change? Why are there two different lines (blue and gray) on the graph?

Part 2

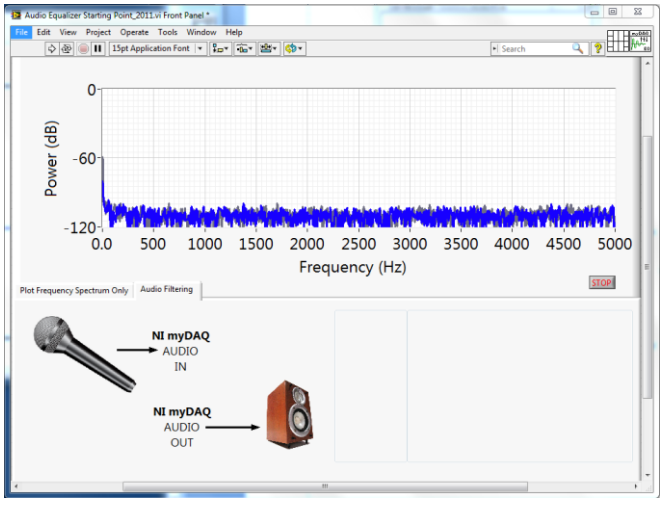
Description Oftentimes a user will want to acquire data continuously. In order to repeat an operation indefinitely in LabVIEW, we can use a While Loop. This will repeat the code contained inside until a user-defined stop condition is achieved. In this case, we will use the Stop button on the front panel.

Procedure

1. Place a While Loop on the block diagram.
   a. Navigate to **Programming»Structures»While Loop** on the Functions palette.
   b. Click in the upper left corner of the block diagram, and drag your mouse to the lower right corner to place the While Loop. The loop should surround all the objects on the diagram.
   c. The stop sign icon in the lower right corner of the loop is the conditional terminal, which is used to stop the loop. Wire the **Stop** button terminal into the conditional terminal. Your VI should look like the image on the next page.



2. Return to the front panel.
3. Move your Stop button (created with the While Loop) to a location that makes it easy to use. To do this, simply click and drag the Stop button. An example is shown below.



4. Run the VI.

Did the VI perform how you expected it to? What methods could we use to adjust the rate at which data is acquired? How could we change how often the Waveform Graph is updated? If we wanted to read in data from a different channel, what would we do?

Notes

- 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 pushpin in the upper left-hand corner of the palette. This tacks down the palette so that it doesn't disappear. This step is omitted in the following exercises, but you should repeat it.

End of Exercise 2 (Track A)

Exercise 2 – Capturing Sound with your Sound Card (Track B)

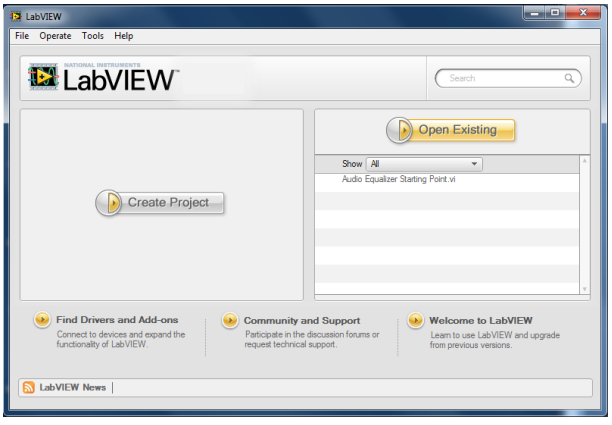
Goal Acquire sound in LabVIEW and present it on the user interface.

Part 1

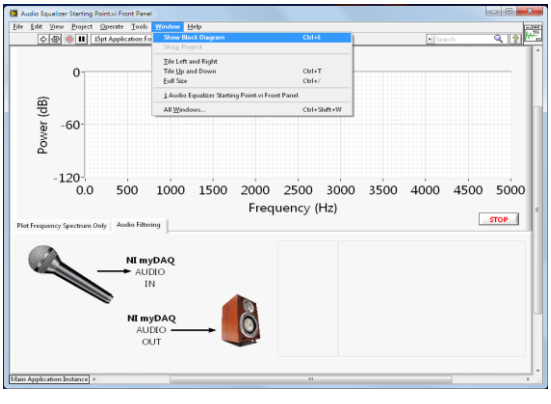
Description We have a microphone connected to the audio input of the computer. In this exercise, we're going to write a LabVIEW VI that reads the audio signal from the computer and displays the Frequency Band.

Procedure

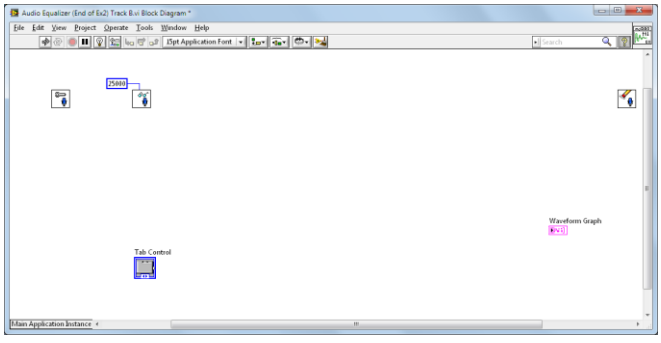
1. Launch LabVIEW.
2. In the Getting Started window, select **File»Open**, and navigate to **Audio Equalizer Starting Point.vi**.



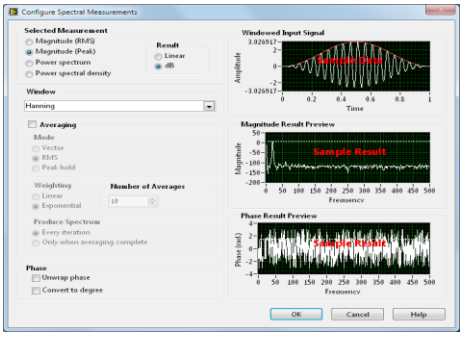
3. You will see that the major elements of the front panel have already been created for you. There is a Waveform Graph, a Stop button, and a Tab Control; we will add more items to the front panel in the next exercise.
4. Open the block diagram by selecting **Window»Show Block Diagram** or by pressing <Ctrl-E>. You can see the terminals that correspond to the three existing front panel objects.



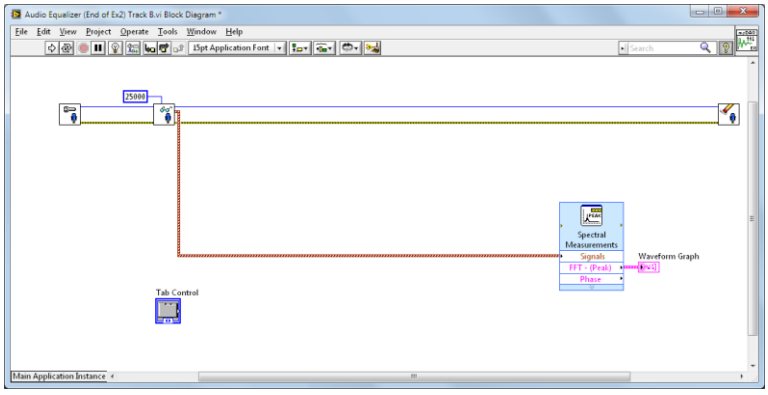
5. Place a Sound Input Configure VI on the block diagram.
   a. Right-click on the block diagram to open the Functions palette.
   b. Select the Sound Input Configure by navigating to **Programming»Graphics and Sound»Sound»Input»Sound Input Configure.vi**.
   c. Click on the block diagram to place the Sound Input Configure VI.
6. Place a Sound Input Read VI on the block diagram.
   a. Right-click on the block diagram to open the Functions palette.
   b. Select the Sound Input Read by navigating to **Programming»Graphics and Sound»Sound»Input»Sound Input Read.vi**.
   c. Click on the block diagram to place the Sound Input Read VI.
   d. Create a constant for the **number of samples/ch input**. Right-click on the input on the top of the Sound Input Read VI and select **Create»Constant**.
   e. Type 25000 in the constant by double-clicking on the value.
7. Place a Sound Input Clear VI on the block diagram following the steps outlined in step 5 but select **Sound Input Clear.vi** instead of Sound Input Read.vi.



8. Place a Spectral Measurements function on the block diagram.
   a. Create a Spectral Measurements function by navigating to **Express»Signal Analysis»Spectral** in the Functions palette. Place the function on the block diagram.
   b. In the **Configure Spectral Measurements** window, change the **Selected Measurement** to Magnitude (Peak). Leave all other default settings unchanged. Click **OK**.



9. Connect the functions and test the VI.
    a. Wire the **task ID** and **error out** terminals of the Sound Input Configure to the **task ID** and **error in** input of the Sound Input Read, respectively.
    b. Wire the **task ID** and **error out** terminals of the Sound Input Read to the **task ID** and **error in** input of the Sound Input Clear, respectively.
    c. Wire the **data** output of the Sound Input Read to the **Signals** input of the Spectral Measurements.
    d. Wire the **FFT – (Peak)** output of the Spectral Measurements to the input terminal of the Waveform Graph.
    e. Return to the front panel, start your music and click the **Run** button (the white arrow in the upper left corner).



What did you see when you clicked the Run button? Try running the VI multiple times while making different sounds into the microphone. Do the results on the graph change?

Part 2

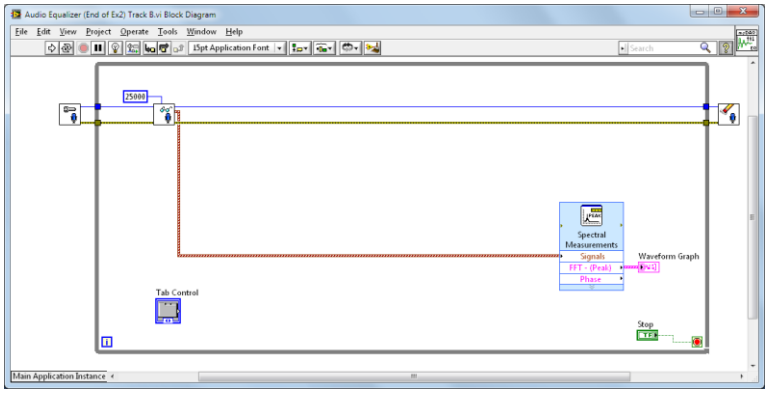
Description Oftentimes a user will want to acquire data continuously. In order to repeat an operation indefinitely in LabVIEW, we can use a While Loop. This will repeat the code contained inside until a user-defined stop condition is achieved. In this case, we will use the Stop button on the front panel.

1. Place a While Loop on the block diagram.
    a. Navigate to **Programming»Structures»While Loop** on the Functions Palette.

We want to enclose only some items in the loop. We want everything but the Sound Input Configure and Sound Input Clear VIs in our While Loop. Can you think of why? We only need to configure and clear our system once, so we do not need for these VIs to execute every iteration.

b. Click in the upper left corner of the block diagram, and drag your mouse to the lower right corner to place the While Loop. Ensure that all components except for the Sound Input Configure and Sound Input Clear VIs are in the dotted outline.

c. The stop sign icon in the lower right corner of the loop is the conditional terminal, which is used to stop the loop. Wire the **Stop** button terminal into the conditional terminal.



2. Return to the front panel and run the VI.

Did the VI perform how you expected it to? What methods could we use to adjust the rate at which data is acquired? How could we change how often the Waveform Graph is updated? If we wanted to read in data from a different channel, what would we do?

End of Exercise 2 (Track B)

Show us how you're incorporating LabVIEW software into your design project for a chance to win cash prizes and trips to NIWeek in Austin, Texas. From inexpensive medical devices to complex underwater autonomous vehicles, we want to see how students like you are engineering a better world using the LabVIEW graphical development environment.

Two members from each of the top four selected projects will be flown to Austin, Texas, to attend NIWeek. One of those projects will win $2,000 USD and be recognized at the Graphical System Design Achievement Awards dinner. $1,500 in cash prizes awarded to first, second and third place winners of the popular vote.

You can submit your projects at

NI.COM/STUDENTDESIGN

Video: Blind Driver Challenge

YouTube Link Part 1
YouTube Link Part 2

Don't forget to submit your project to **ni.com/studentdesign** for a chance to win prizes and a trip to Austin, TX

Section II – Elements of Typical Programs

A. Demonstration: While Loops and SubVI

B. Loops

- While Loop
- For Loop

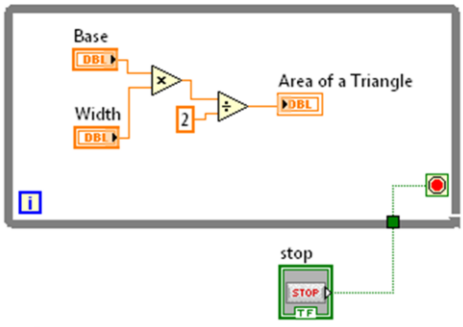
B. Functions and SubVIs

- Types of Functions
- Creating Custom Functions (subVI)

C. Hands-On Exercise: Filtering and Outputting Sound

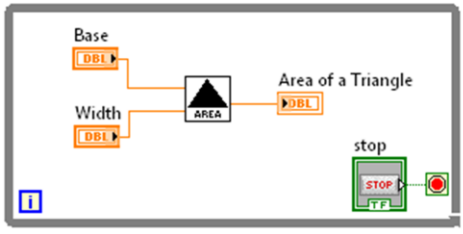
Demonstration: While Loop and SubVIs

Dataflow and Loops



Bad Example

SubVIs

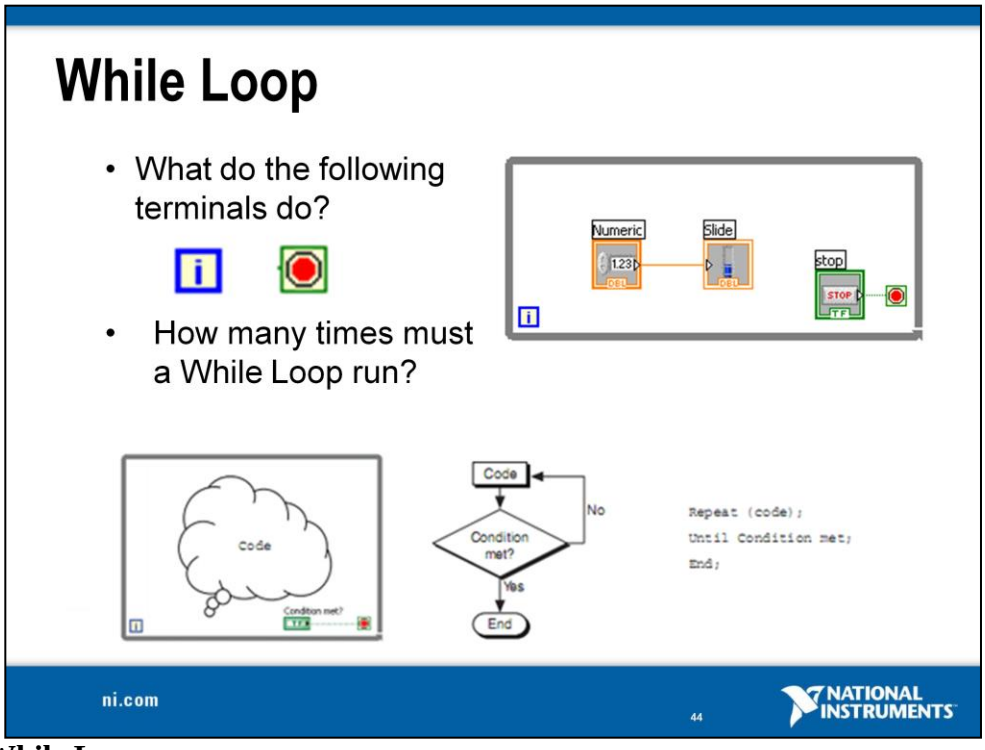


Good Example

ni.com

43

NATIONAL INSTRUMENTS



While Loops

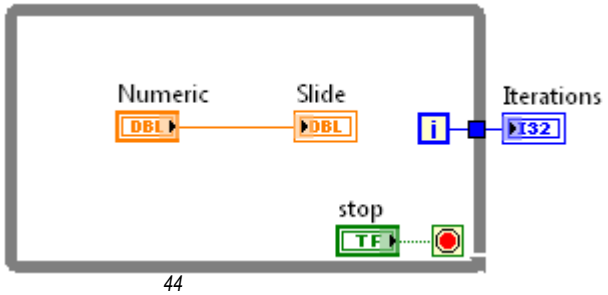
Similar to a do loop or a repeat-until loop in text-based programming languages, a While Loop executes a subdiagram until a condition is met. The While Loop executes the subdiagram until the **Conditional Termina** (an input terminal) receives a specific Boolean value.

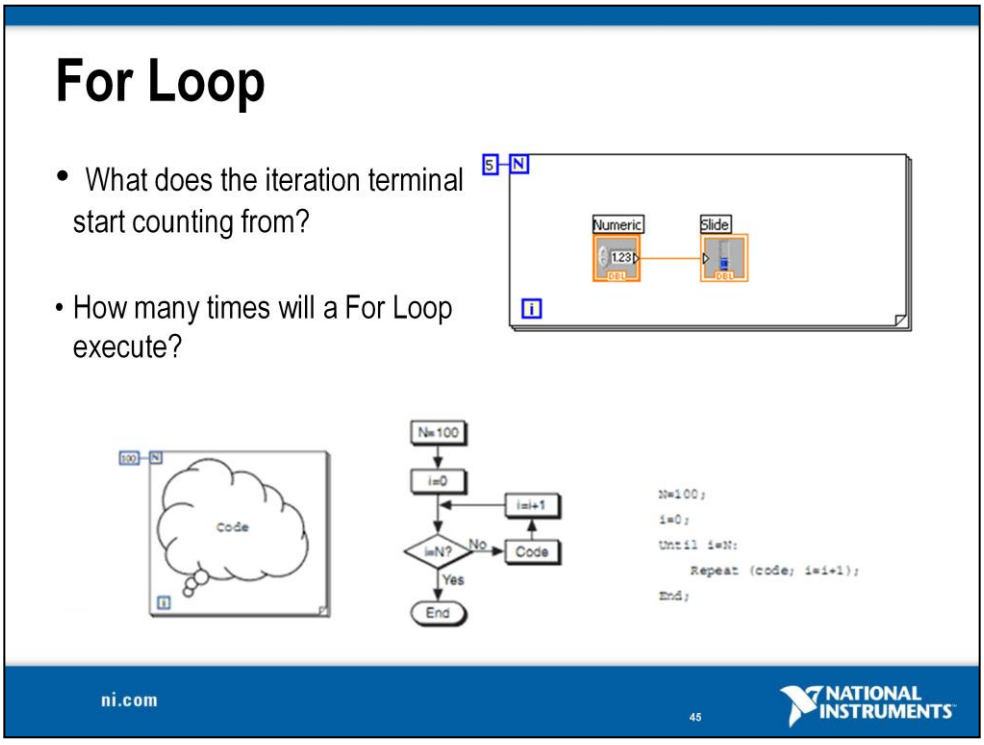
The **Iteration Termina** is an output terminal that contains the number of completed iterations. The iteration count for the While Loop always starts at zero.

Note: The While Loop always executes at least once.

Tunnels feed data into and out of structures. The tunnel appears as a solid block on the border of the While Loop. The block is the color of the data type wired to the tunnel. Data passes out of a loop after the loop terminates. When a tunnel passes data into a loop, the loop executes only after data arrives at the tunnel.

In the image below, the Iteration Terminal is connected to a tunnel. The value in the tunnel does not get passed to the Iterations indicator until the While Loop finishes executing. Only the last value of the Iteration Terminal displays in the Iterations indicator.

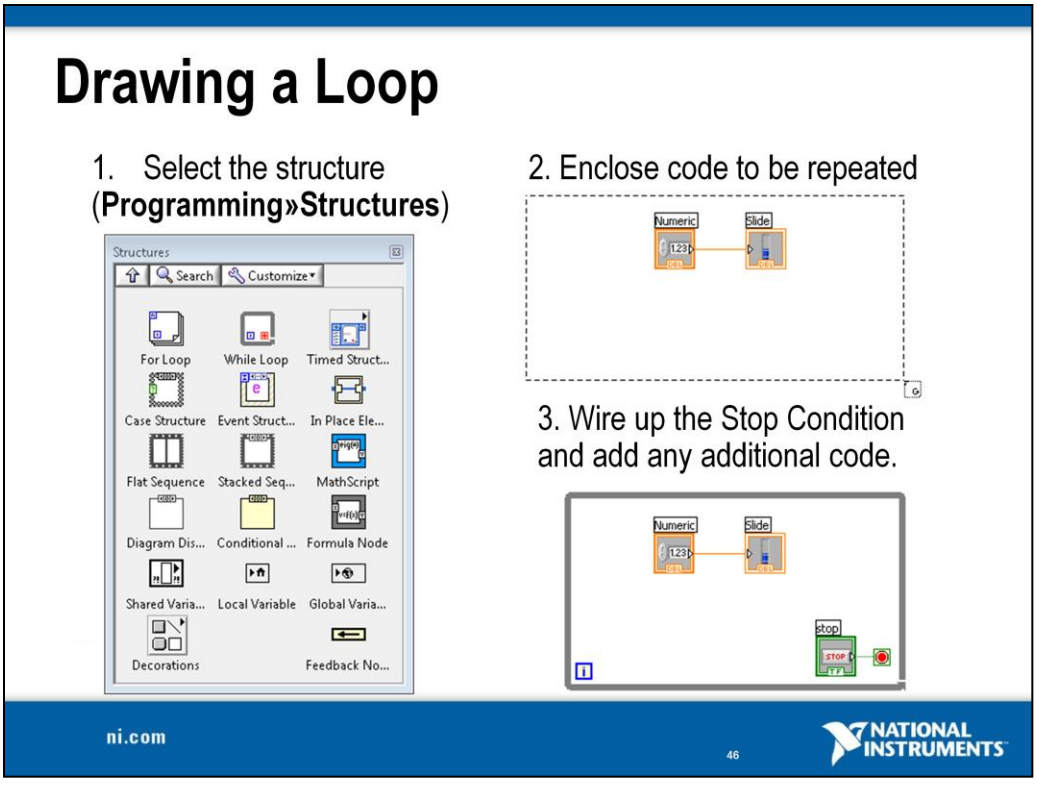




For Loops

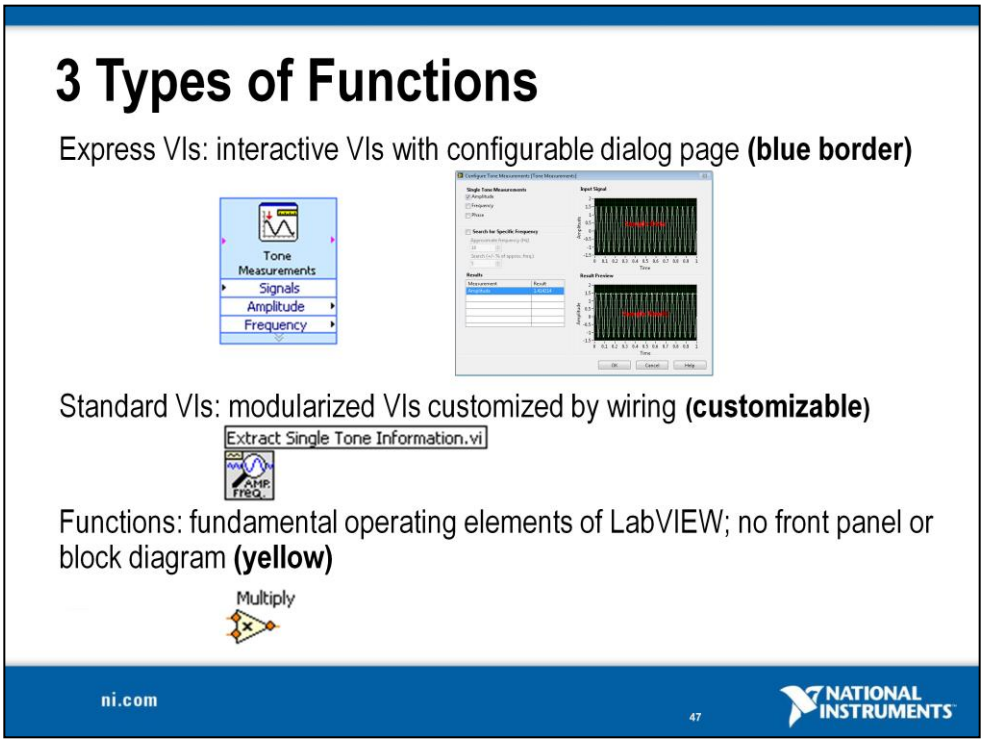
A For Loop, shown above, executes a subdiagram a set number of times. The value in the count terminal N (an input terminal) indicates how many times to repeat the subdiagram.

The For Loop also has an **Iteration Termina**i . It is also an output terminal that contains the number of completed iterations. The iteration count for the For Loop always starts at zero.



The While and For Loops can be found by navigating to **Functions Palette»Programming» Structures**. Select the While or For Loop from the palette then use the cursor to drag a selection rectangle around the section of the block diagram you want to repeat. When you release the mouse button, a loop boundary encloses the section you selected. Add block diagram objects to the loop by dragging and dropping them inside the loop.

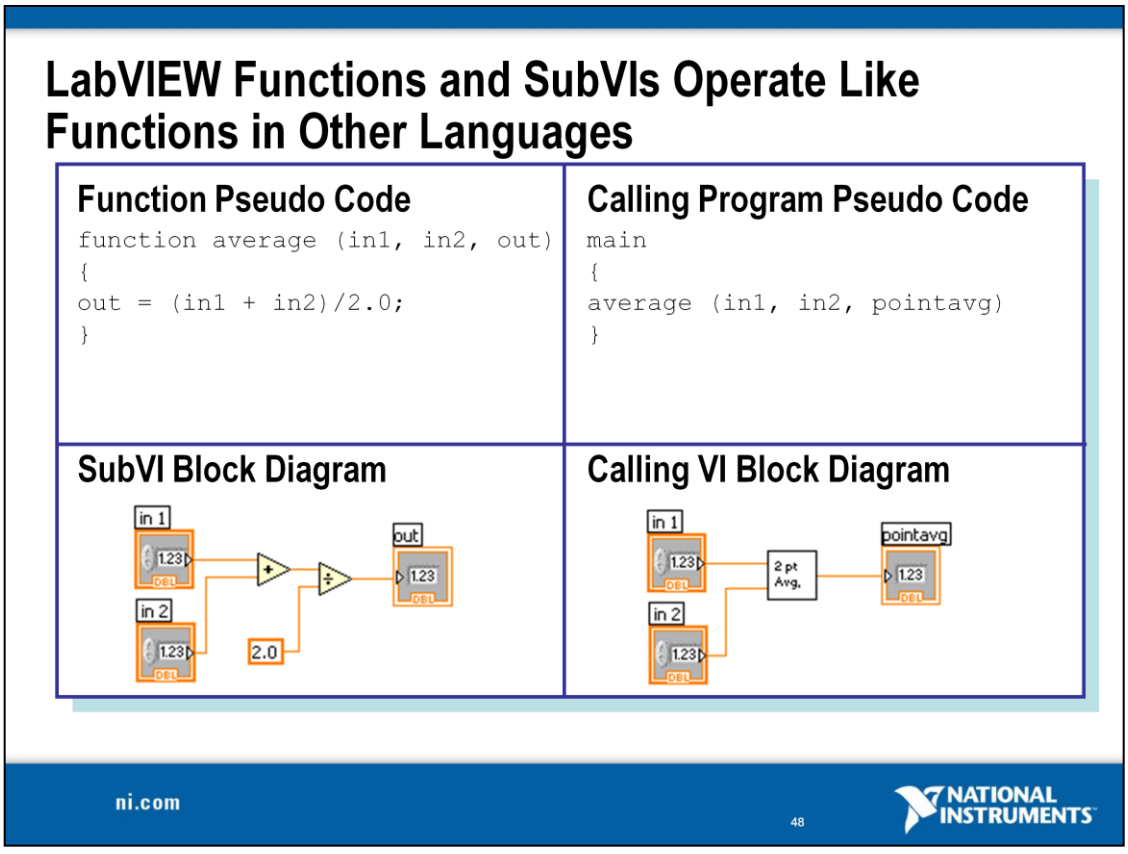
- When selected, the mouse cursor becomes a special pointer that you use to enclose the section of code you want to include in the While Loop.
- Click the mouse button to define the top-left corner and then click the mouse button again at the bottom-right corner. The loop boundary appears around the selected code.
- Drag or drop additional nodes in the loop if needed.
- If you draw a While Loop, be sure to wire up the Conditional Terminal.



LabVIEW includes configuration-based Express VIs for measurement analysis and signal processing. With Express VIs, you can interactively explore the various analysis algorithms, while immediately seeing the results on the configuration dialog. LabVIEW then generates a subVI based on these settings. The complexity associated with adding analysis and signal processing algorithms into your measurement and automation applications is significantly reduced by using Express VIs.

SubVIs are VIs (consisting of a front panel and a block diagram) that are used within another VI. When you double-click a subVI on the block diagram, its front panel window appears.

Functions are the fundamental operating elements of LabVIEW. Functions do not have front panel windows or block diagrams windows but do have connector panes. Double-clicking a function only selects the function. A function has a pale yellow background on its icon.

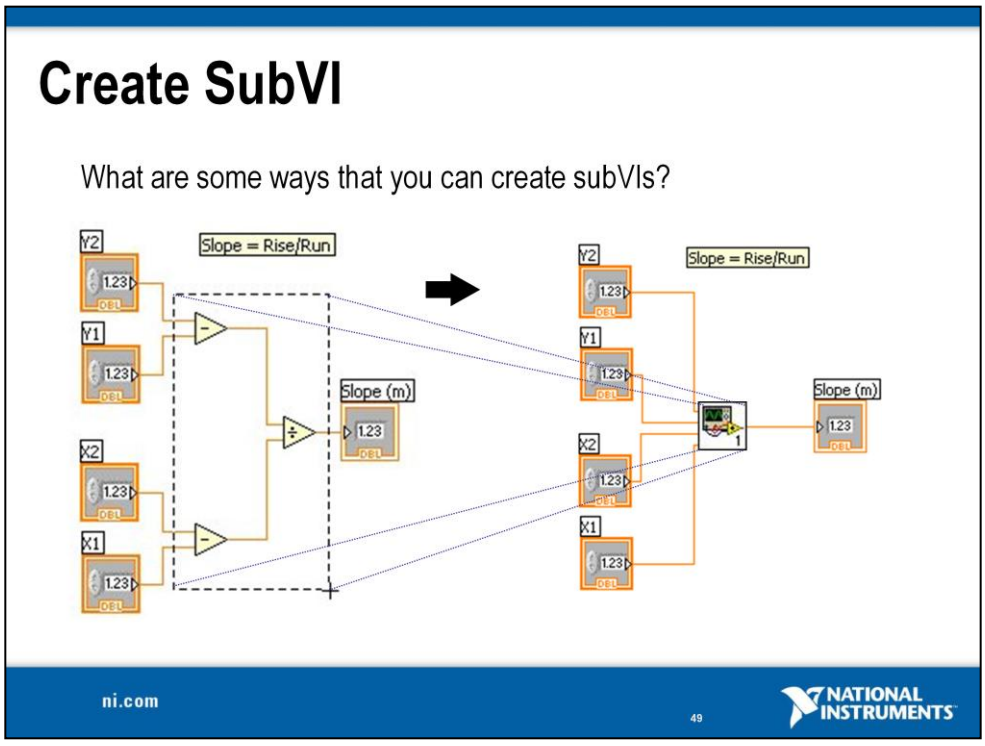


Modularity defines the degree to which a program is composed of discrete modules such that a change to one module has minimum impact on other modules. Modules in LabVIEW are called subVIs. The modular approach makes applications easier to debug and maintain.

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

As you create VIs, you might find that you perform a certain operation frequently. Consider using subVIs or loops to perform that operation repetitively.



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.

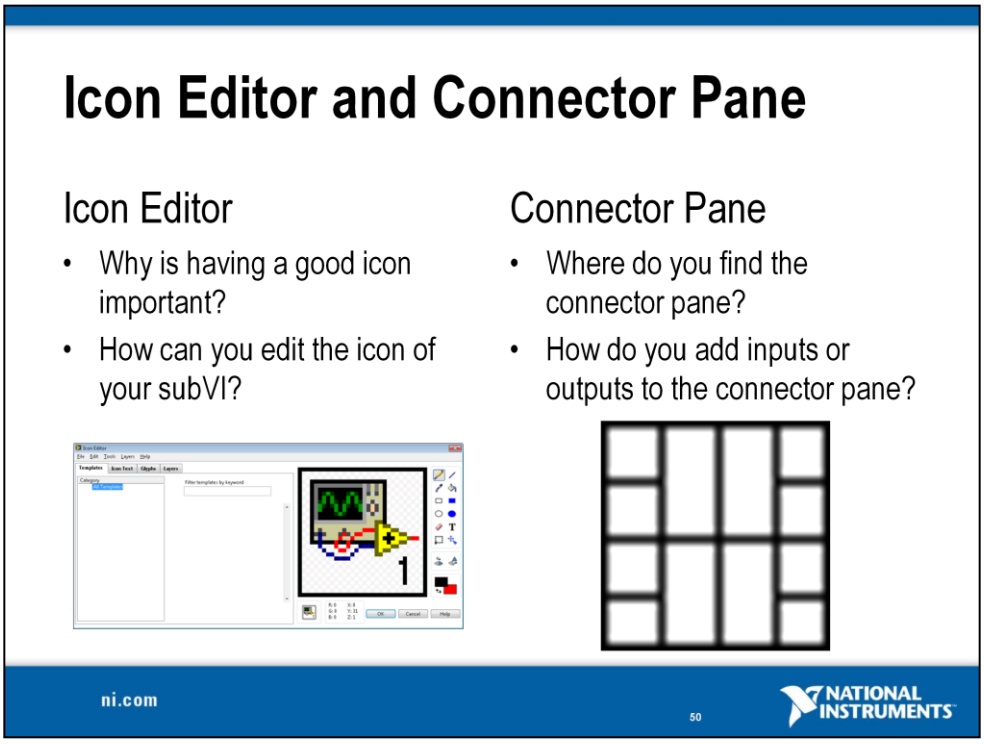
A subVI node corresponds to a subroutine call in text-based programming languages. A block diagram that contains several identical subVI nodes calls the same subVI several times.

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.

You can easily customize subVI input and output terminals as well as the icon. 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.

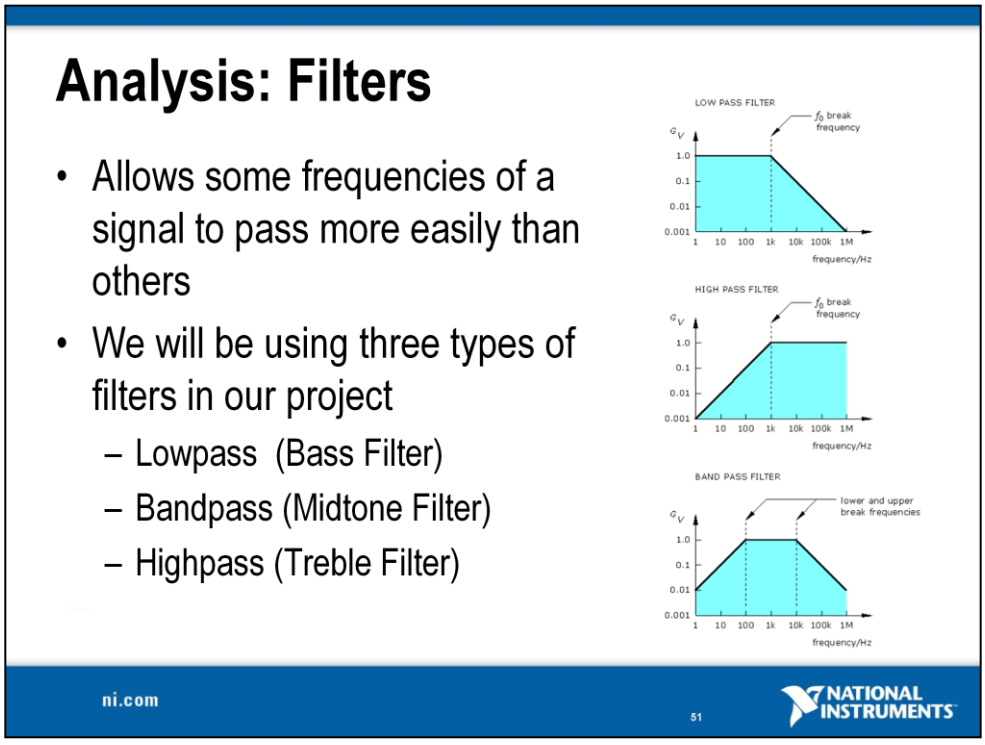


After you build a VI front panel and block diagram, modify the icon and the connector pane so you can use the VI as a subVI. Every VI displays an icon in the upper right corner of the front panel and block diagram windows.

A VI icon is a graphical representation of a VI that can contain text, images, or a combination of both. If you use a VI as a subVI, the icon identifies the subVI on the block diagram of the VI. You can double-click the icon in the front panel window or block diagram window to customize or edit it. You can also right-click on the icon and select **Edit Icon** to launch the Icon Editor dialog box.

To use a VI as a subVI you also need to build a connector pane. The connector pane defines the inputs and outputs you can wire to the VI so you can use it as a subVI. Define connections by assigning a front panel control or indicator to each of the connector pane terminals. To define a connector pane, right-click the icon in the upper right corner of the front panel and select **Show Connector** from the shortcut menu to display the connector pane. When you view the connector pane for the first time, you see a connector pattern. You can select a different pattern by right-clicking the connector pane and selecting **Patterns** from the shortcut menu.

Each rectangle on the connector pane represents a terminal. Use the rectangles to assign inputs and outputs. To assign a terminal to a front panel control or indicator, click a terminal of the connector pane, then click the front panel control or indicator you want to assign to that terminal. The terminal changes to the data type color of the control to indicate that you connected the terminal. You can also select the control or indicator first and then select the terminal.



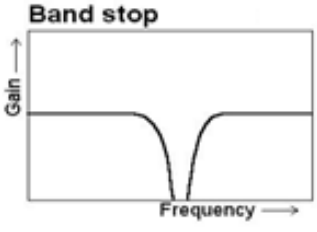
A filter removes an unwanted component or feature. Often times you will use a filter to remove certain frequencies (such as 60 Hz noise) while not altering the original frequencies. There are many different filters available for us to use.

Lowpass Filter – Passes all frequencies below a specified value while attenuating higher frequency values. The top image in the slide shows a lowpass filter.

Highpass Filter – Passes all frequencies above a specified value while attenuating lower frequency values. The middle image in the slide shows a highpass filter.

Bandpass Filter – Passes frequencies within a specified range and rejects frequencies outside of that range. The last image in the slide shows a bandpass filter.

Bandstop Filter – Passes all frequencies except those specified by a specified range as seen in the figure to the right.



In our project, we are building an audio equalizer. In order to isolate the frequencies to manipulate the way our input sounds, we will need to build a filter for each of the three frequency ranges we have defined.

Outputting signal to DAQ

- Send a signal from the computer to your data acquisition device or sound card
- In our case we are modifying the audio input and sending the signal to our speakers.

ni.com 52 NATIONAL INSTRUMENTS

Analog output allows us to send a continuous signal that varies within a specified range from our data acquisition card to a sensor or another application if applicable. In our case we are sending our modified audio signal to the speakers or headphones so we can hear the changes that our filter is implementing.

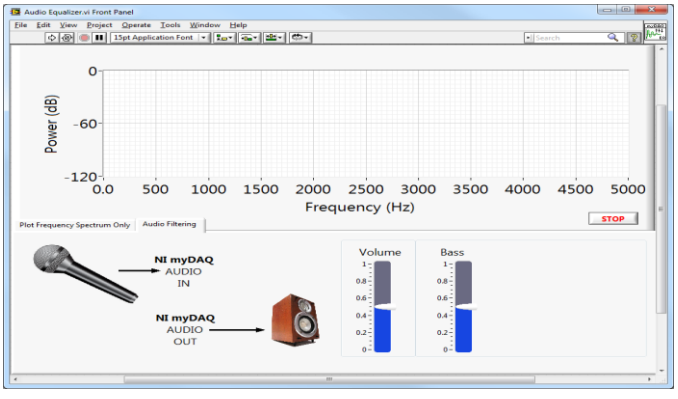
Exercise 3 – Outputting Sound (Track A)

Goal Apply a variable filter to alter the incoming data, and output the modified signal from myDAQ.

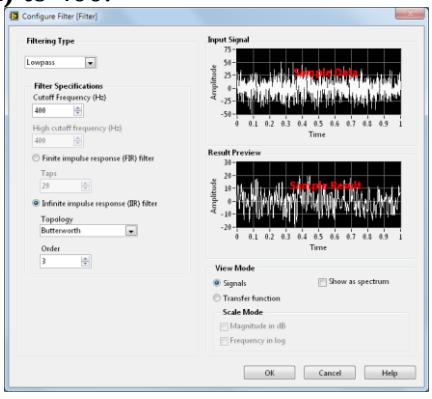
Description We can do more than just acquire and analyze a signal in LabVIEW. In this exercise, we will use the audio output of the myDAQ to play the signal on speakers or headphones. There's no fun in outputting the exact same signal that we read in, so we'll apply a variable filter to isolate part of the signal. In this case, we'll extract the bass (low-frequency) elements and play them back.

Procedure

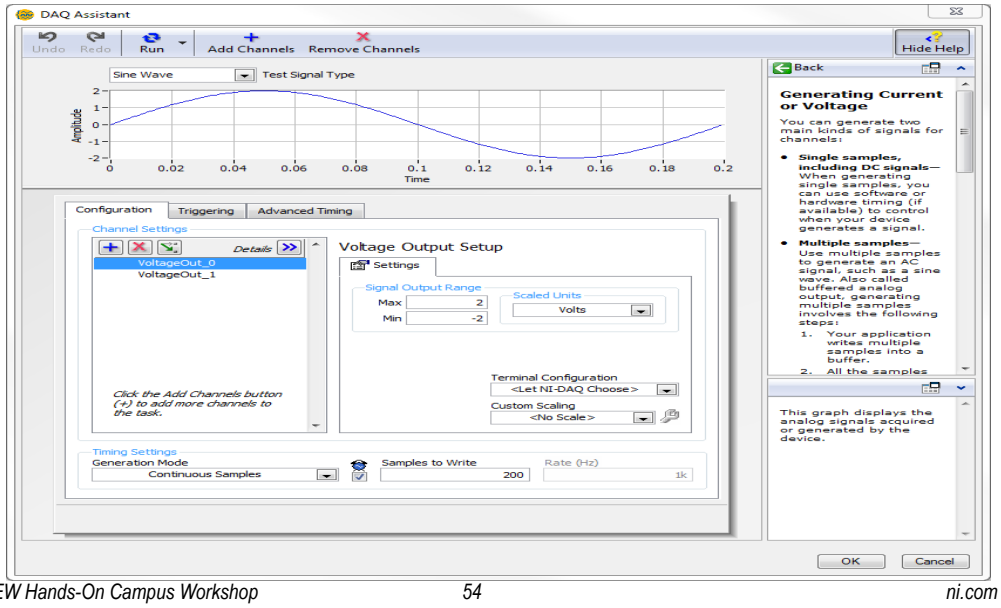
1. Add volume slides to the Audio Filtering tab of the Front Panel.
    a. On the Front Panel, select the Audio Filtering tab.
    b. Right-click on the Front Panel to open the Controls Palette.
    c. Navigate to **Modern»Numeric»Vertical Pointer Slide**.
    d. Place the slide in the small box in the Audio Filtering tab and name it "Volume."
    e. The default scale is 0 to 10, but we want ours to be 0–1. Double-click the 10 at the top of the slide and type a 1. The rest of the scale will update accordingly.
    f. Create a copy of the slide by pressing <Ctrl > and dragging the slide to the right. Name the copy Bass. Your front panel should resemble the one shown below.



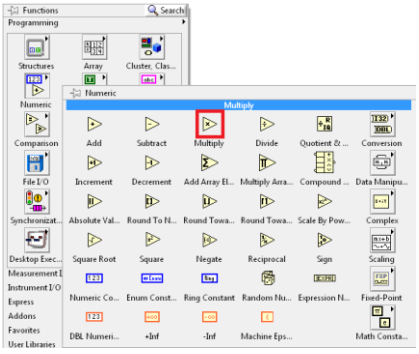
2. Place a filter on the block diagram.
   a. Return to the block diagram and add a filter from **Express»Signal Analysis»**
      Filter.
   b. Leave the **Filtering Type** as the default Lowpass, but change **Cutoff Frequency (Hz)** to 400.



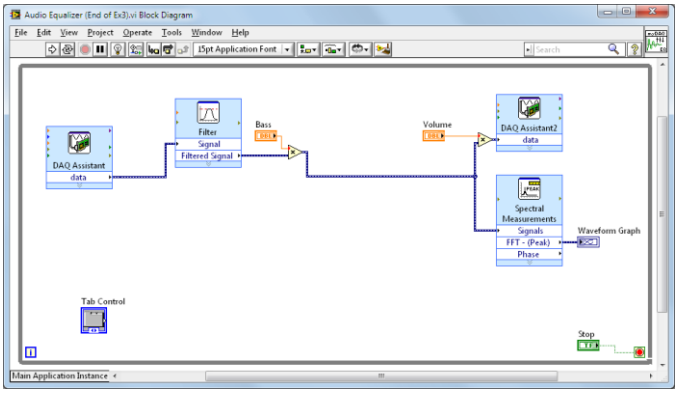
3. Create a second DAQ Assistant to output the filtered signal.
   a. Create a DAQ Assistant from **Express»Output»DAQ Assistant** and place it above the Spectral Measurements.
   b. Navigate to **Generate Signals»Analog Output»Voltage**, and select the audioOutputLeft and audioOutputRight physical channels under the myDAQ listing.
   c. Change the **Signal Output Range Max** and **Min** to 2 and -2 Volts for both channels. ***Make sure to do this for both channels.***
   d. Under **Timing** Settings, change the **Generation Mode** to Continuous Samples and set the **Samples to Write** to 200. The configuration window should match the one below.



4. Place two Multiply functions on the block diagram. These can be found on the Functions palette at **Programming»Numeric»Multiply**.



5. Re-wire the block diagram to include the new functions.
    a. Wire the **data** terminal of the first DAQ Assistant (analog input) to the **Signal** input of the filter.
    b. Wire the **Filtered Signal** output and the Bass control into a Multiply function.
    c. The output of the first Multiply function goes two places. One is the **Signals** input of the Spectral Measurements. The second Multiply function receives the Volume control and the output of the first Multiply function.
    d. The output of the second Multiply is wired into the **data** terminal of the second DAQ Assistant (analog output).



6. Return to the front panel and run the VI. If you have an mp3 player or external audio source, try connecting it to myDAQ instead of the microphone.

How does the behavior of the finished VI differ from the last exercise? Try adjusting the values of the Volume and Bass slides; how is the graph affected? What sort of sound comes out of your speakers/headphones? Can you think of a practical application for this type of signal manipulation?

End of Exercise 3 (Track A)

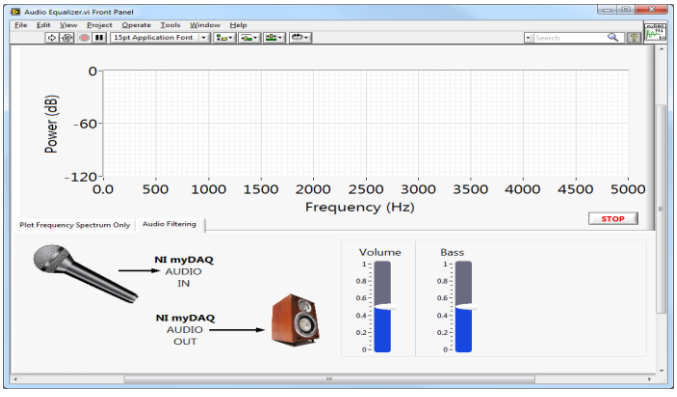
Exercise 3 – Outputting Sound (Track B)

Goal Apply a variable filter to alter the incoming data, and output the modified signal.

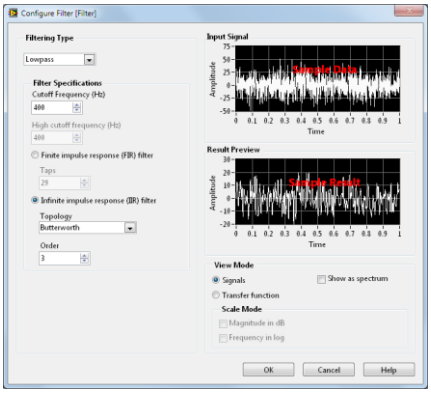
Description We can do more than just acquire and analyze a signal in LabVIEW. In this exercise, we will use the audio output of your computer to play the signal on speakers or headphones. There's no fun in outputting the exact same signal that we read in, so we'll apply a variable filter to isolate part of the signal. In this case, we'll extract the bass (low-frequency) elements and play them back.

Procedure

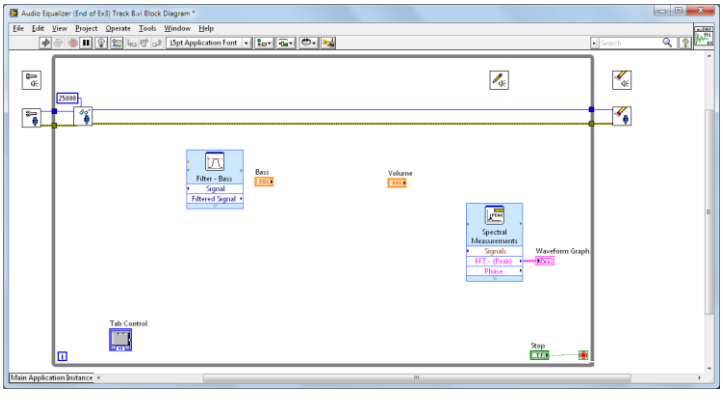
1. Add volume slides to the Audio Filtering tab of the front panel.
   a. On the front panel, select the Audio Filtering tab.
   b. Right-click on the front panel to open the Controls Palette.
   c. Navigate to **Modern»Numeric»Vertical Pointer Slide**.
   d. Place the slide in the small box in the Audio Filtering tab and name it Volume.
   e. The default scale is 0 to 10, but we want ours to be 0–1. Double-click the 10 at the top of the slide and type a 1. The rest of the scale will update accordingly.
   f. Create a copy of the slide by pressing <Ctrl >and dragging the slide to the right. Name the copy Bass. Your front panel should resemble the one shown below.



2. Place a filter on the block diagram.
   a. Return to the block diagram and add a filter from **Express»Signal Analysis»**
      Filter.
   b. Leave the **Filtering Type** as the default Lowpass, but change **Cutoff Frequency (Hz)** to 400.

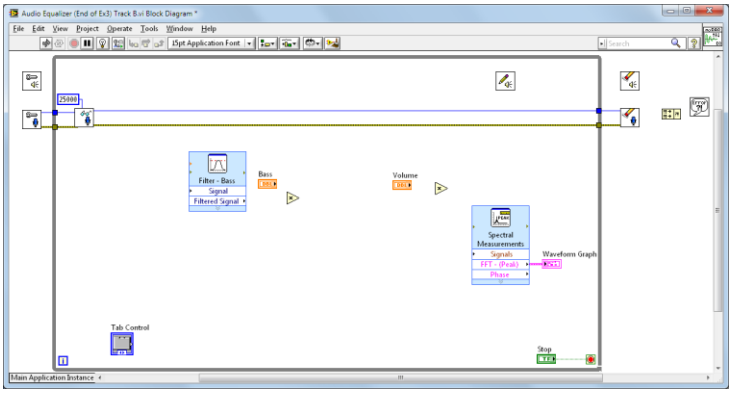


3. Place a Sound Output Configure VI on the block diagram.
   a. Right-click on the block diagram to open the Functions palette.
   b. Select the Sound Input Configure by navigating to **Programming»Graphics and Sound»Sound»Output**. Then select **Sound Input Configure.vi.**
   c. Click on the block diagram to place the Sound Output Configure VI.
4. Place a Sound Output Write.vi on the block diagram following the steps outlined above.
5. Place a Sound Output Clear.vi on the block diagram following the steps outlined in step 3.

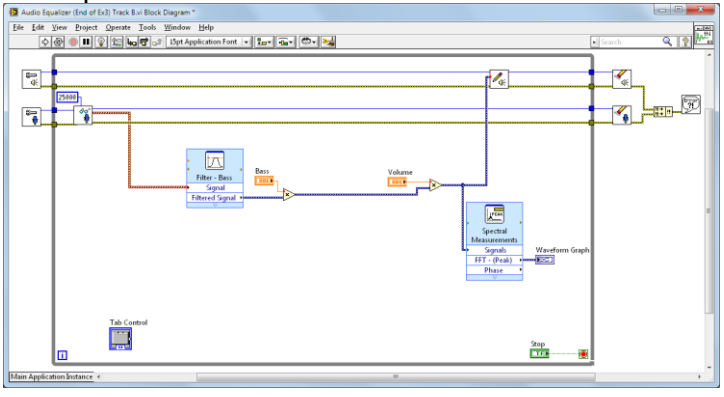


6. Place two Multiply functions on the block diagram. These can be found on the Functions palette at **Programming»Numeric»Multiply**.
7. Place a Merge Errors VI on the block diagram outside of the While Loop. This can be found on the Functions palette at **Programming»Dialog & User Interface»Merge Error**.

8. Place a Simple Error Handler on the block diagram outside of the While Loop. This can be found on the **Dialog and User Interface** palette as well.



9. Re-wire the block diagram to include the new functions.
   a. Wire the **task ID** and **error out** terminals of the Sound Output Configure to the **task ID** and **error in** input of the Sound Output Write, respectively.
   b. Wire the **task ID** and **error out** terminals of the Sound Output Write to the **task ID** and **error in** input of the Sound Output Clear, respectively.
   c. Wire the error out wires of the Sound Input Clear and Sound Output Clear to the Merge Error VI.
   d. Wire the output of the Merge Errors VI to the Simple Error Handler.
   e. Wire the **data** terminal of the Sound Input Read to the **Signal** input of the Filter.
   f. Wire the **Filtered Signal** output and the Bass control into a Multiply function.
   g. The output of the first Multiply function goes two places. One is the **Signals** input of the Spectral Measurements. The second Multiply function receives the Volume control and the output of the first Multiply function.
   h. The output of the second Multiply is wired into the **data** terminal of the Sound Output Write.



10. Return to the front panel and run the VI. If you have an mp3 player or external audio source, try connecting it to your microphone input.

How does the behavior of the finished VI differ from the last exercise? Try adjusting the values of the Volume and Bass slides; how is the graph affected? What sort of sound comes out of your speakers/headphones? Can you think of a practical application for this type of signal manipulation? What is the purpose of the error handling that we did?

End of Exercise 3 (Track B)

Video: myDAQ Piano Staircase

YouTube Video Link

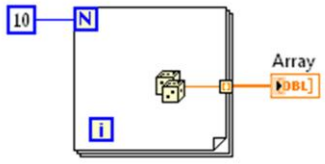
Don't forget to submit your project to **ni.com/studentdesign** for a chance to win prizes and a trip to Austin, TX

Section III – Analyzing and Presenting Your Results

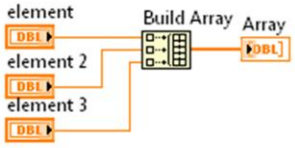
A. Demonstration: Arrays and Auto-Indexing
B. Arrays
   - Creating Arrays
   - Auto-Indexing
C. Displaying Data on the Front Panel
   - Graphs and Charts
D. Demonstration: Case Structures
   - Decision Making
E. Hands-On Exercise: Audio Equalizer

ni.com 61 NATIONAL INSTRUMENTS

Demonstration: Arrays



Creating Arrays and Auto-Indexing



Programatically Creating Arrays

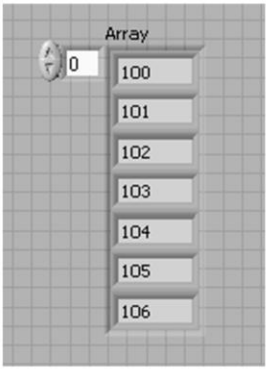
ni.com

62

NATIONAL INSTRUMENTS

Arrays

- An array consists of elements and dimension. What do those terms mean?
- When would you use an array?
- In LabVIEW, what is the index of the first element in an array?



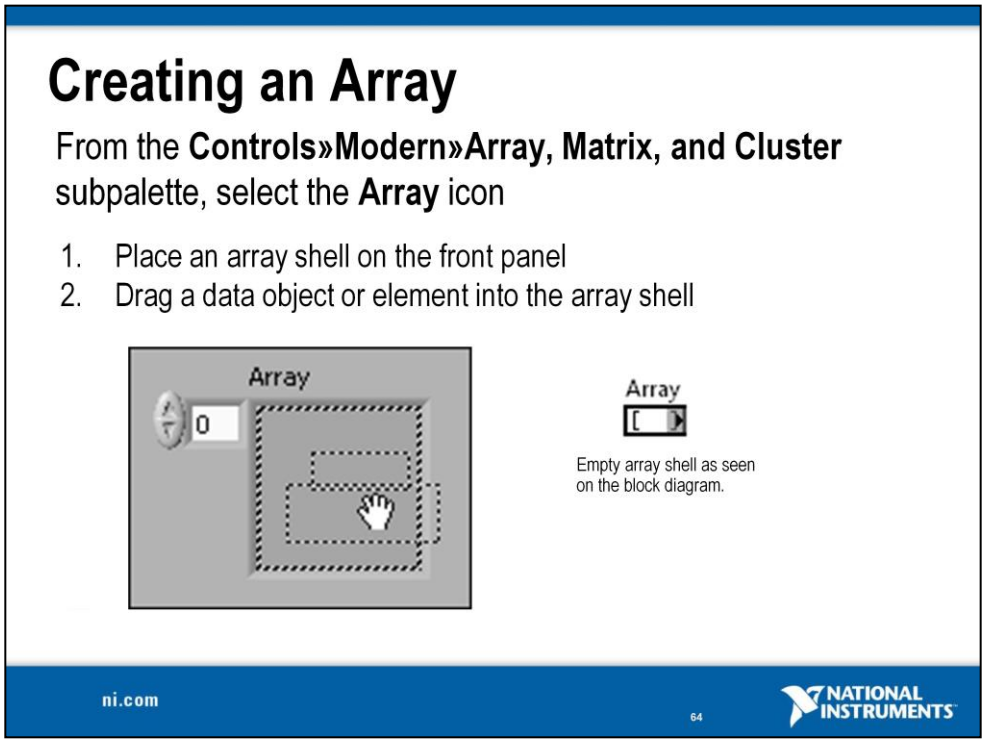
ni.com 63 NATIONAL INSTRUMENTS

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.

You can build arrays of numeric, Boolean, path, string, waveform and cluster data types. You cannot create an array of arrays, but you can use a multidimensional array in LabVIEW. Consider using arrays when you work with a collection of similar data and when you perform repetitive computations. Arrays are ideal for storing data you collect from waveforms or data generated in loops, where each iteration of a loop produces one element of the array.

Note: Array indexes in LabVIEW are zero-based. The index of the first element in the array, regardless of its dimension, is zero.

Array elements are ordered. An array uses an index so you can readily access any particular element. The index is zero-based, which means it is in the range 0 to n–1, where n is the number of elements in the array.



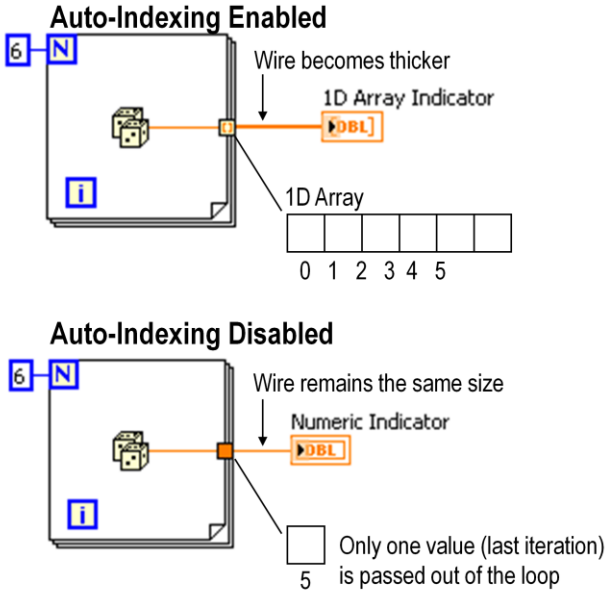
To create an array control or indicator as shown, select an array on the **Controls»Modern»Array, Matrix, and Cluster** palette, place it on the front panel, and drag a control or indicator into the array shell. If you attempt to drag an invalid control or indicator such as an XY graph into the array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

To add dimensions to an array, 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.

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
- How can you enable/disable auto-indexing?



ni.com 65 NATIONAL INSTRUMENTS

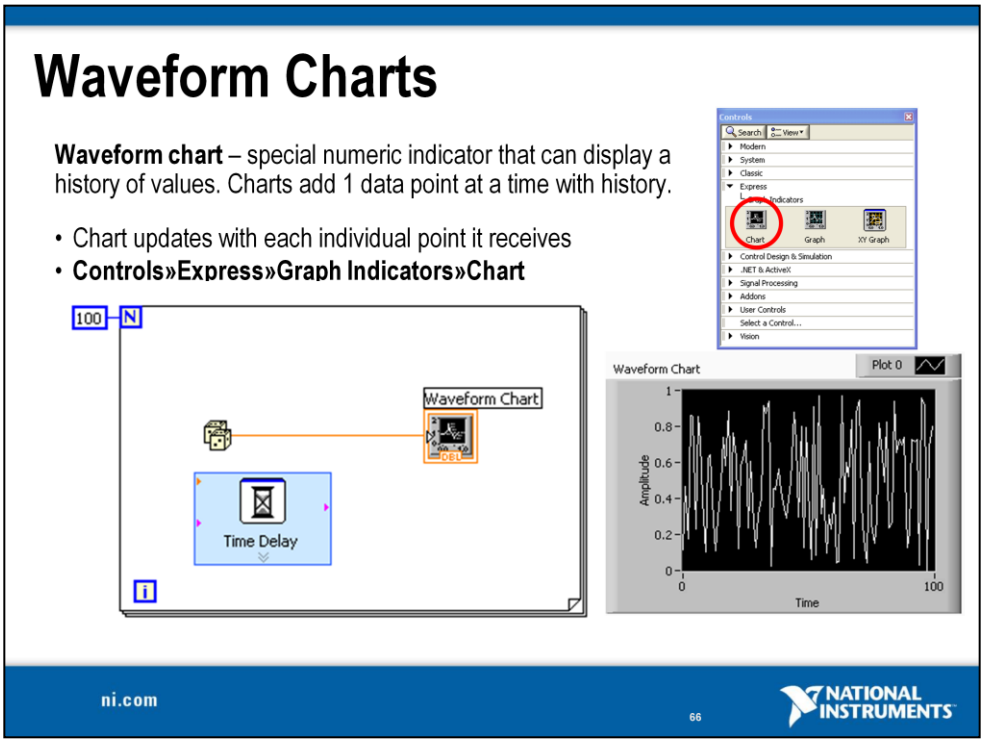
For Loops and While Loops can index and accumulate arrays at their boundaries. This is known as auto-indexing.

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

You can enable or disable auto-indexing by right-clicking on a tunnel and selecting **Enable Indexing or Disable Indexing.**

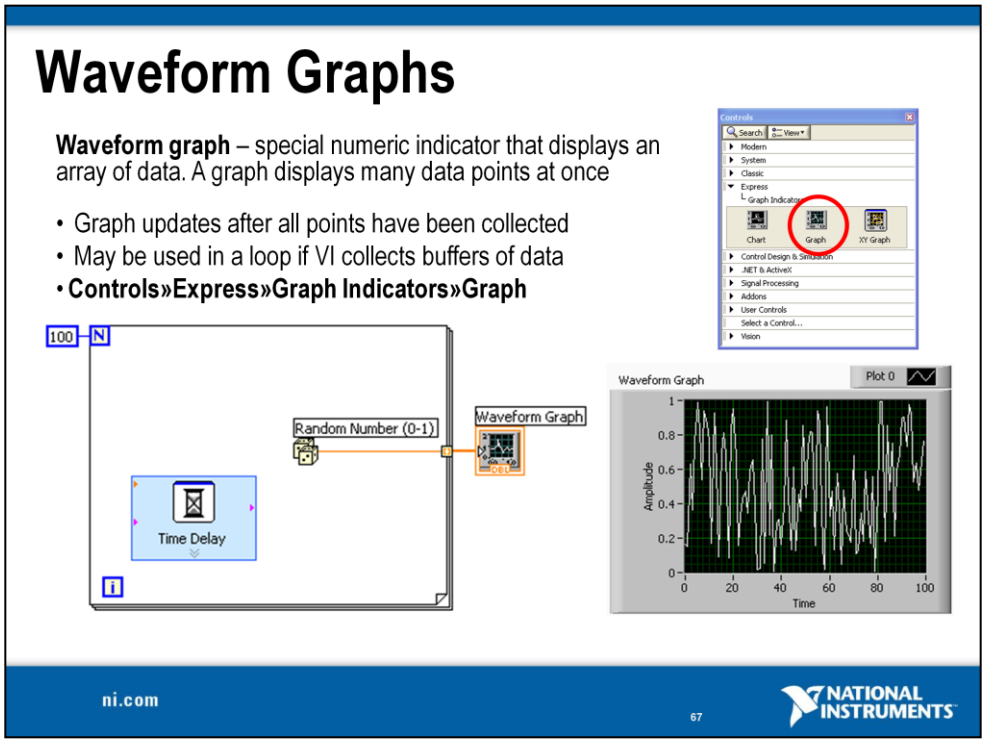
Examples:

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



The waveform chart is a special numeric indicator that displays one or more plots. It is located on the **Controls»Modern»Graph** palette. Waveform charts can display single or multiple plots. Waveform Charts have the ability to update its display with every iteration of a loop. This allows you to see you data as you acquire it.

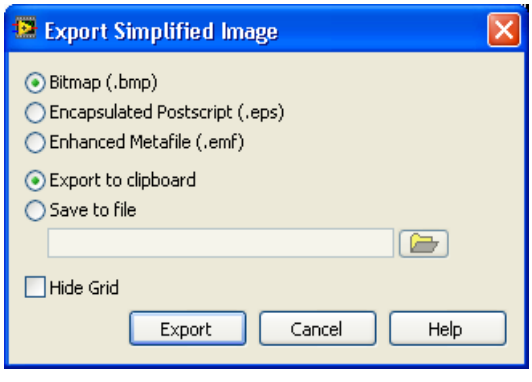
You can change the minimum and maximum values of either the x-axis or y-axis axis by double-clicking on the value with the labeling tool and typing the new value. Similarly, you can change the label of the axis. You can also right-click the plot legend and change the style, shape, and color of the trace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

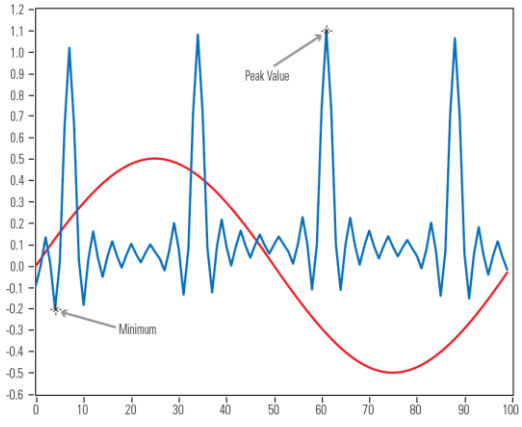


Graphs are very powerful indicators in LabVIEW. You can use these highly customizable tools to concisely display a great deal of information. VIs with a graph usually collect the data in an array and then plot the data to the graph. A waveform graph does not update with every iteration of a loop like a waveform chart does.

With the properties page of the graph, you can 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**.

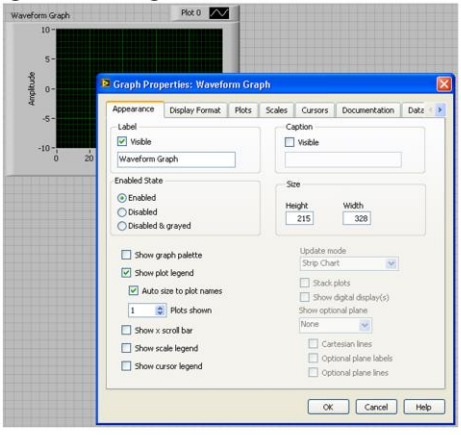
You can also create technical-paper-quality graphics with the Export Simplified Image function. Right-click the graph and select **Data Operations»Export Simplified Image…**





Control and 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

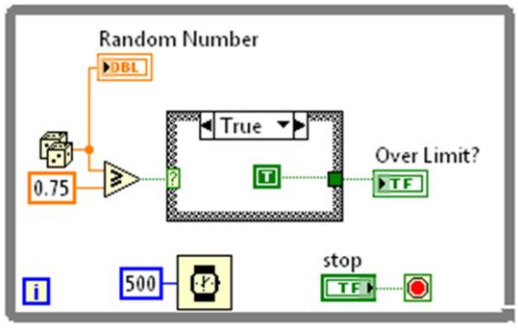


ni.com 68 NATIONAL INSTRUMENTS

Objects in the front panel window also have property dialog boxes that you can use to change the look or behavior of the objects. Right-click an object and select **Properties** from the shortcut menu to access the property dialog box. 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.

You can select multiple objects on the front panel or the block diagram and edit any properties the objects share. To select multiple objects, use the Positioning tool to drag a selection rectangle around all of the objects you want to edit or hold down the <Shift> key while clicking each object.

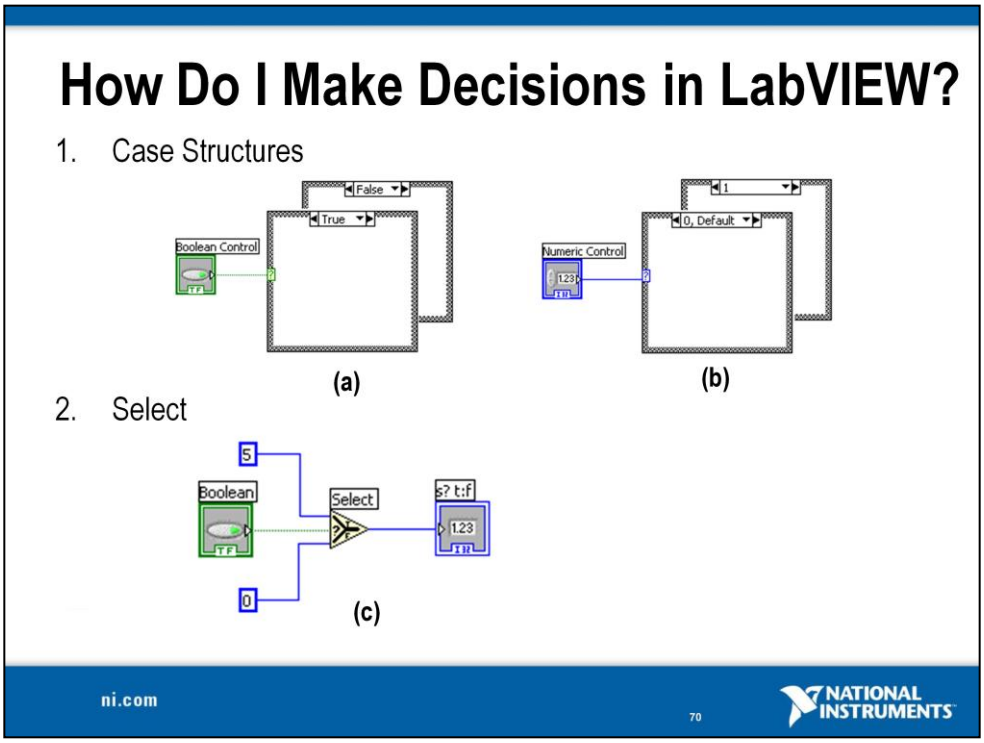
Demonstration: Case Structures



ni.com

69

NATIONAL INSTRUMENTS



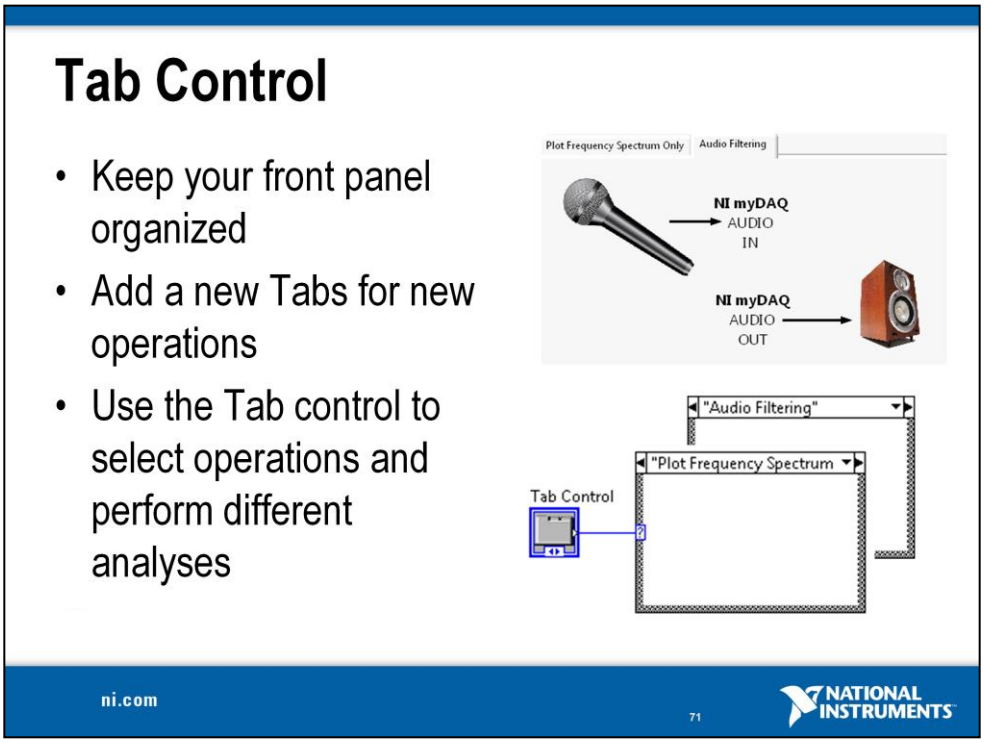
Case Structure

The case structure has one or more subdiagrams, or cases, 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 executes; otherwise the FALSE case executes.
- **Example B**: Numeric input. The input value determines which box to execute. If out of range of the cases, LabVIEW chooses 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.



You can use tab controls to overlap front panel controls and indicators in a smaller area. Place front panel objects on the pages of the tab control and use the tab as the selector to display each page. You can add any front panel objects to the active page. A page is active when the tab for that page is flush with the page and the objects on the page are visible. Terminals for controls and indicators you place on the tab control appear as any other block diagram terminal.

Note: When you add objects to a tab control page, frequently go to the block diagram window and arrange the newly added terminals. When you add several objects to a tab control, the terminals on the block diagram can become cluttered.

On the block diagram, the tab control is an enumerated control or indicator. As a control, you can pass the value of the active page to other block diagram nodes. As an indicator, you can wire nodes to control which page is displayed. You do not need to wire the tab control terminal for the tab control to operate.

You can wire the enumerated control terminal of the tab control to the selector of a Case Structure to produce cleaner block diagrams. With this method, you associate each page of the tab control with a subdiagram in the Case structure. You place the control and indicator terminals from each page of the tab control—as well as the block diagram nodes and wires associated with those terminals—into the subdiagrams of the Case structure.

Exercise 4 – Creating an Audio Equalizer (Track A)

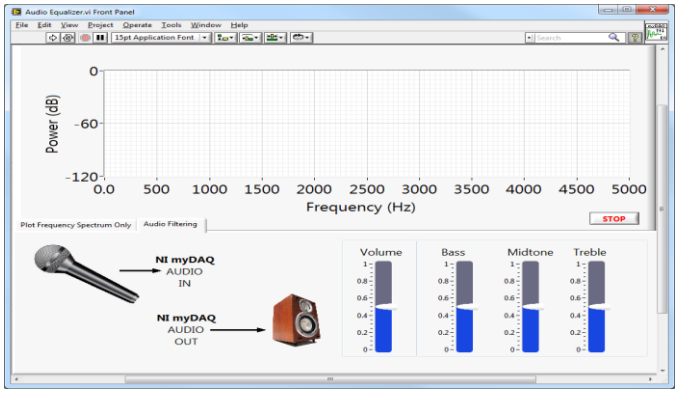
Goal Use additional filters to create a full audio equalizer. Implement a Case Structure to allow the user to turn equalizer on or off.

Part 1

Description We can add to the existing code in our program to make an audio equalizer that functions like your car stereo. In order to do this, we'll need to isolate more than just the Bass. If we create separate filters for the Midtone and Treble frequency ranges, we can control the three ranges independently. This functionality is comparable to what a sound system does when you change its settings.

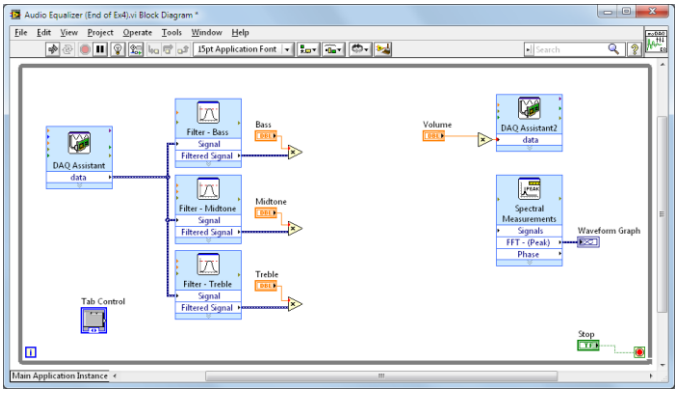
Procedure

1. On the front panel, create two more slide controls next to Bass. Name the new slides Midtone and Treble. Your front panel should look like the one shown below.

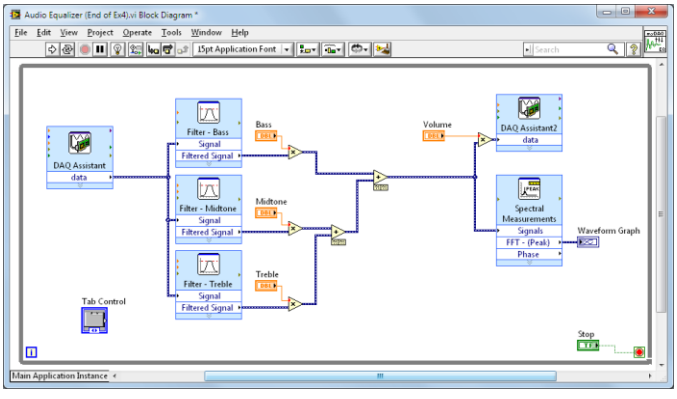


2. Add two band filters to the block diagram.
   a. On the block diagram, create two more Filter functions. You can quickly do this by holding down <Ctrl> and dragging the existing Filter down. You may find it useful to rename the Filters "Bass," "Midtone," and "Treble."
   b. You can change Filter settings by double-clicking in the light blue area, which opens the **Configure Filter** window. For the Midtone filter, change the **Filtering Type** to Bandpass, and set the **Low** and **High cutoff frequency (Hz)** to 450 and 2500, respectively.
   c. The Treble filter is also a Bandpass, with a cutoff range from 3000 to 10000 Hz.

3. Separate and filter the input signal.
   a. The DAQ Assistant's **data** terminal should be wired to the **Signal** inputs of all three Filters.
   b. As we did with the Bass control, use Multiply functions to integrate the Midtone and Treble controls with their Filters.



4. Recombine the three filtered signals.
   a. Place two Add functions on the block diagram from **Programming» Numeric»Add**.
   b. Use one Add function to combine the multiplied Midtone and Treble outputs, and use the second Add to combine that sum with the Bass output.
   c. This new sum should be wired to the Spectral Measurements and the second DAQ Assistant multiplier.



5. Return to the front panel and run the VI.

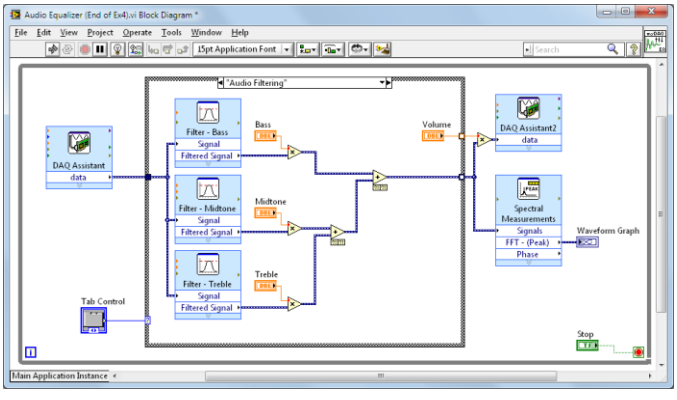
Try listening to yourself or music through the audio equalizer. Do the results match what you expected? If you turn all three equalizer slides up, does the output signal match the original input signal?

Part 2

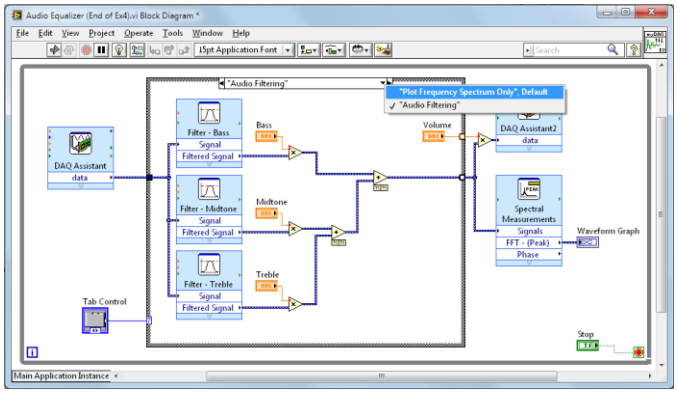
Description We now have a working audio equalizer, but a key question has been overlooked. What if the user wants to turn off the equalizing functions and analyze the original input signal? We need to modify our program to make the equalizing an optional function. In this case, we will utilize the Tab Control on the front panel. When the Audio Filtering tab is selected, the equalizing functions will work as we just observed. If the Plot Frequency Spectrum Only tab is selected, those functions will be disabled.

Procedure

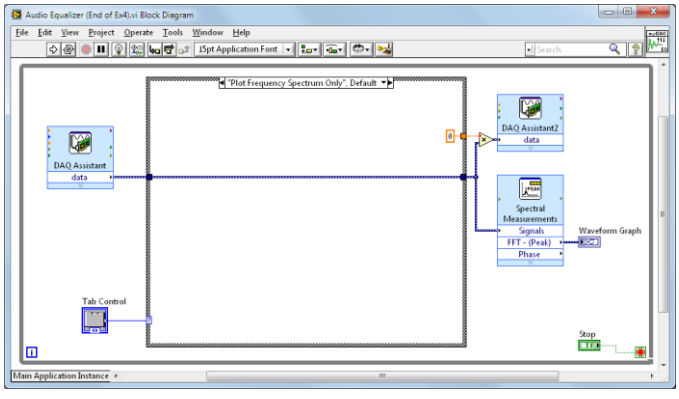
1. Place a Case Structure on the block diagram.
   a. Select a Case Structure by navigating to **Programming»Structures»Case Structure**.
   b. We want the Case Structure to contain the filters, signal addition, and Volume control. As with the While Loop, click in the upper left corner and drag to create the Structure.
   c. The small green question mark on the left side of the Case Structure is called the case selector. Wire the Tab Control terminal into the case selector. The case selector should turn blue ( ).



2. The case label should change to Audio Filtering.
3. The colored boxes on either side of the Case Structure are called data tunnels. The two tunnels on the right edge of case structure have white centers because they aren't wired in the other case. The VI will only execute if the output value of a tunnel is defined in every case.
4. Change to the Plot Frequency Spectrum Only case using the arrows at the top of the Case Structure.



5. Pass the DAQ Assistant **data** wire directly through this case to the output tunnel.
6. To resolve the empty orange tunnel, right-click on the terminal and select **Create»Constant**. Leave it as the default 0.



7. Return to the front panel and run the VI.

How has the behavior of the VI changed? Does switching tabs change the data on the graph? Why doesn't any audio come out when the "Plot Frequency Spectrum Only" signal tab is selected?

End of Exercise 4 (Track A)

Exercise 4 – Creating an Audio Equalizer (Track B)

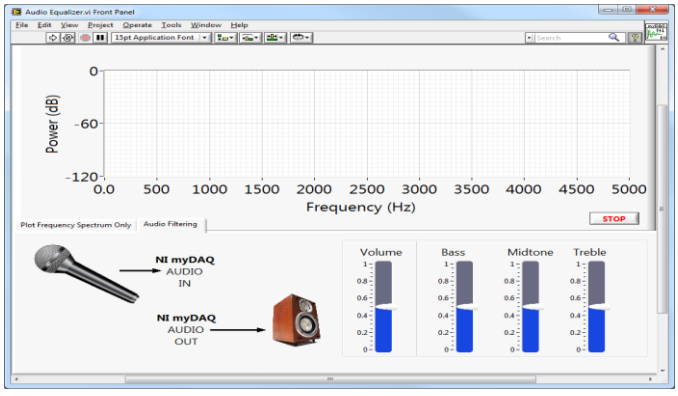
Goal Use additional filters to create a full audio equalizer. Implement a Case Structure to allow the user to turn equalizer on or off.

Part 1

Description We can add to the existing code in our program to make an audio equalizer that functions like your car stereo. In order to do this, we'll need to isolate more than just the Bass. If we create separate filters for the Midtone and Treble frequency ranges, we can control the three ranges independently. This functionality is comparable to what a sound system does when you change its settings.

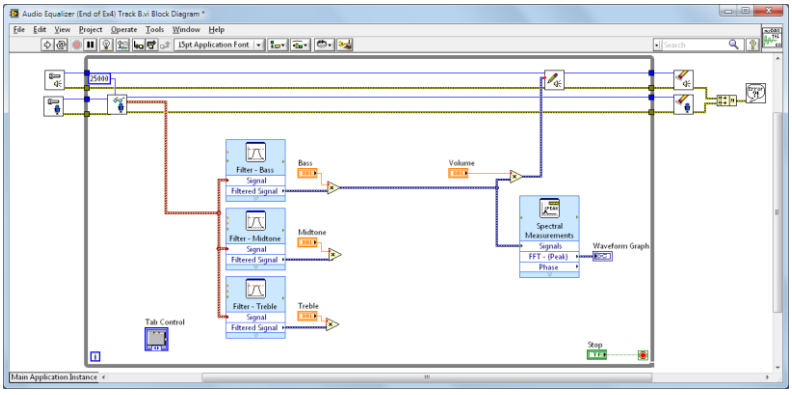
Procedure

1. On the front panel, create two more slide controls next to Bass. Name the new slides Midtone and Treble. Your front panel should look like the one shown below.

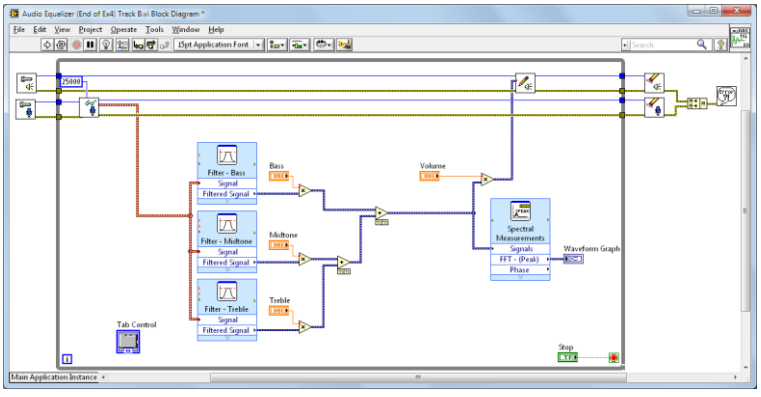


2. Add two band filters to the block diagram.
   a. On the block diagram, create two more Filter functions. You can quickly do this by holding down <Ctrl > and dragging the existing Filter down. You may find it useful to rename the Filters Bass, Midtone, and Treble.
   b. You can change Filter settings by double-clicking in the light blue area, which opens the **Configure Filter** window. For the Midtone filter, change the **Filtering Type** to Bandpass, and set the **Low** and **High cutoff frequency (Hz)** to 450 and 2500, respectively.
   c. The Treble filter is also a bandpass, with a cutoff range from 3000 to 10000 Hz.

3. Separate and filter the input signal.
   a. The Sound Input Read's **data** terminal should be wired to the **Signal** inputs of all three Filters.
   b. As we did with the Bass control, use Multiply functions to integrate the Midtone and Treble controls with their filters.



4. Recombine the three filtered signals.
   a. Place two Add functions on the block diagram from **Programming» Numeric»Add.**
   b. Use one Add function to combine the multiplied Midtone and Treble outputs, and use the second Add to combine that sum with the Bass output.
   c. This new sum should be wired to the Spectral Measurements and the second multiplier that feeds into the Sound Output Write VI.



5. Return to the front panel and run the VI.

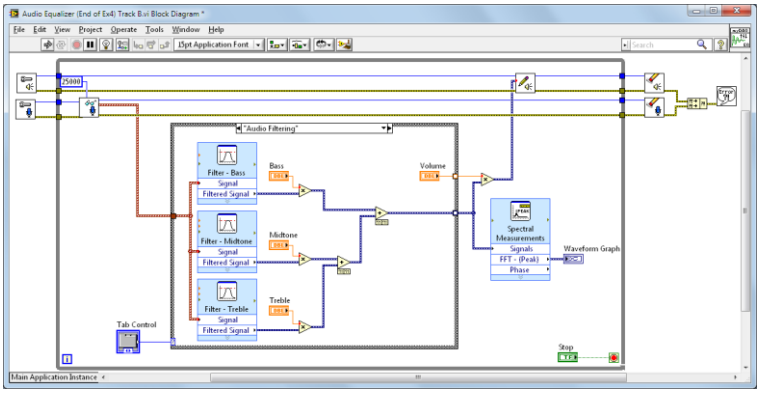
Try listening to yourself or music through the audio equalizer. Do the results match what you expected? If you turn all three equalizer slides up, does the output signal match the original input signal?

Part 2

Description We now have a working audio equalizer, but a key question has been overlooked. What if the user wants to turn off the equalizing functions and analyze the original input signal? We need to modify our program to make the equalizing an optional function. In this case, we will utilize the Tab Control on the front panel. When the Audio Filtering tab is selected, the equalizing functions will work as we just observed. If the Plot Frequency Spectrum Only tab is selected, those functions will be disabled.

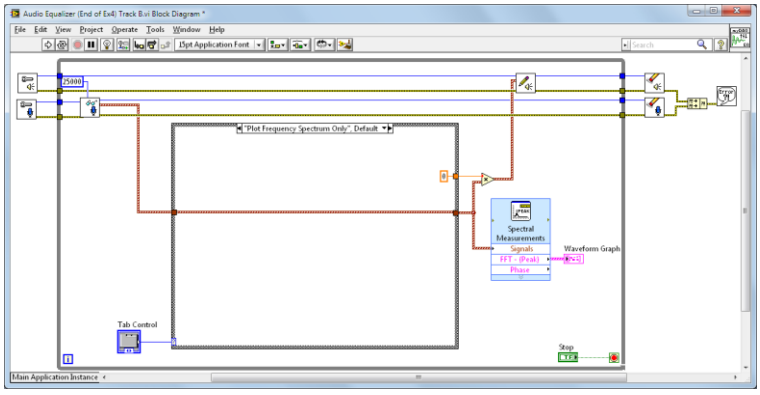
Procedure

1. Place a Case Structure on the block diagram.
   a. Select a Case Structure by navigating to **Programming»Structures»Case Structure**.
   b. We want the Case Structure to contain the filters, signal addition, and Volume control. As with the While Loop, click in the upper left corner and drag to create the Structure.
   c. The small green question mark on the left side of the Case Structure is called the case selector. Wire the Tab Control terminal into the case selector. The case selector should turn blue ( ).



2. The case label should change to Audio Filtering.
3. The colored boxes on either side of the Case Structure are called data tunnels. The two tunnels on the right edge of case structure have white centers because they aren't wired in the other case. The VI will only execute if the output value of a tunnel is defined in every case.
4. Change to the Plot Frequency Spectrum Only case using the arrows at the top of the Case Structure.
5. Pass the Sound Input Read **data** wire directly through this case to the output tunnel.

6. To resolve the empty orange tunnel, right-click on the terminal and select **Create»Constant**. Leave it as the default 0.



7. Return to the front panel and run the VI.

How has the behavior of the VI changed? Does switching tabs change the data on the graph? Why doesn't any audio come out when the "Plot Frequency Spectrum Only" signal tab is selected?

End of Exercise 4 (Track B)

Video: Mind Controlled Wheelchair

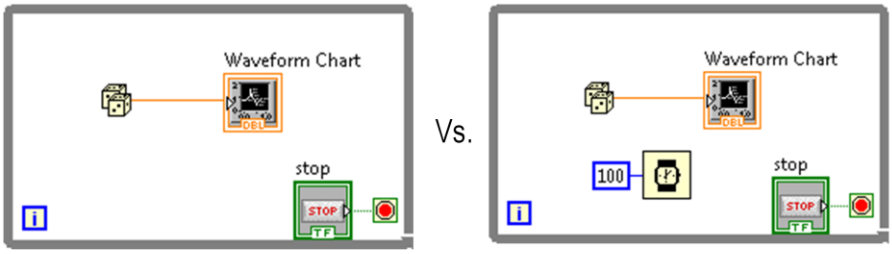
YouTube Link

Don't forget to submit your project to **ni.com/studentdesign** for a chance to win prizes and a trip to Austin, TX

Section IV – Timing and File I/O

A. Demonstration: Timing a While Loop
B. Timing Loops
C. File I/O
D. Hands-On Exercise: Adding a Karaoke Machine

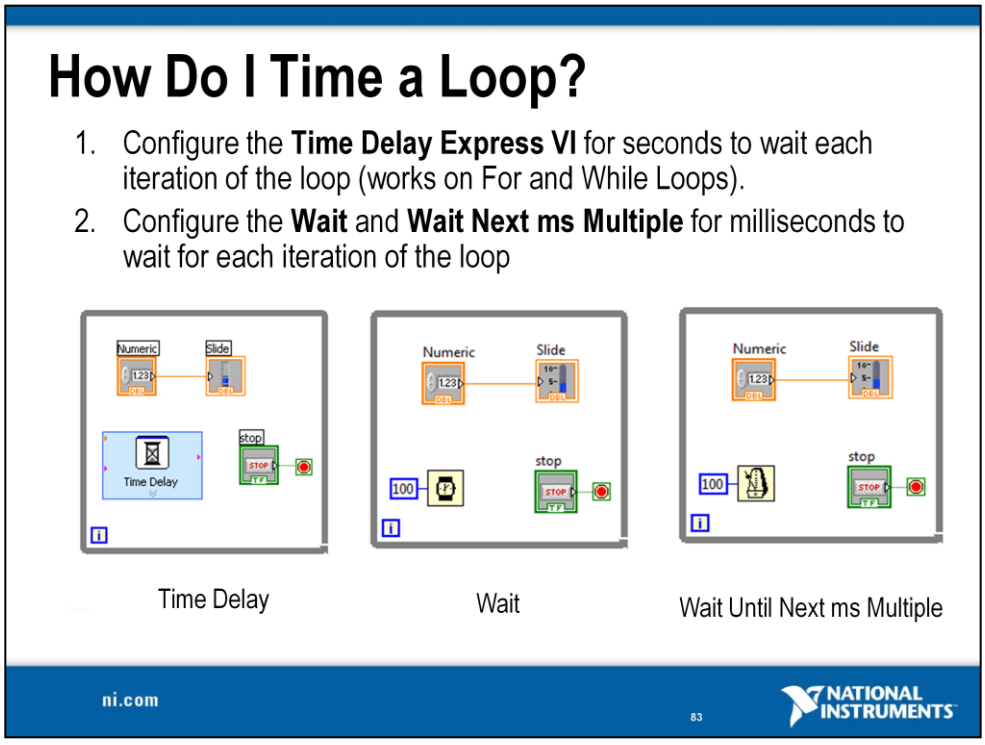
Demonstration: Timing a Loop



ni.com

82

NATIONAL INSTRUMENTS



Time Delay

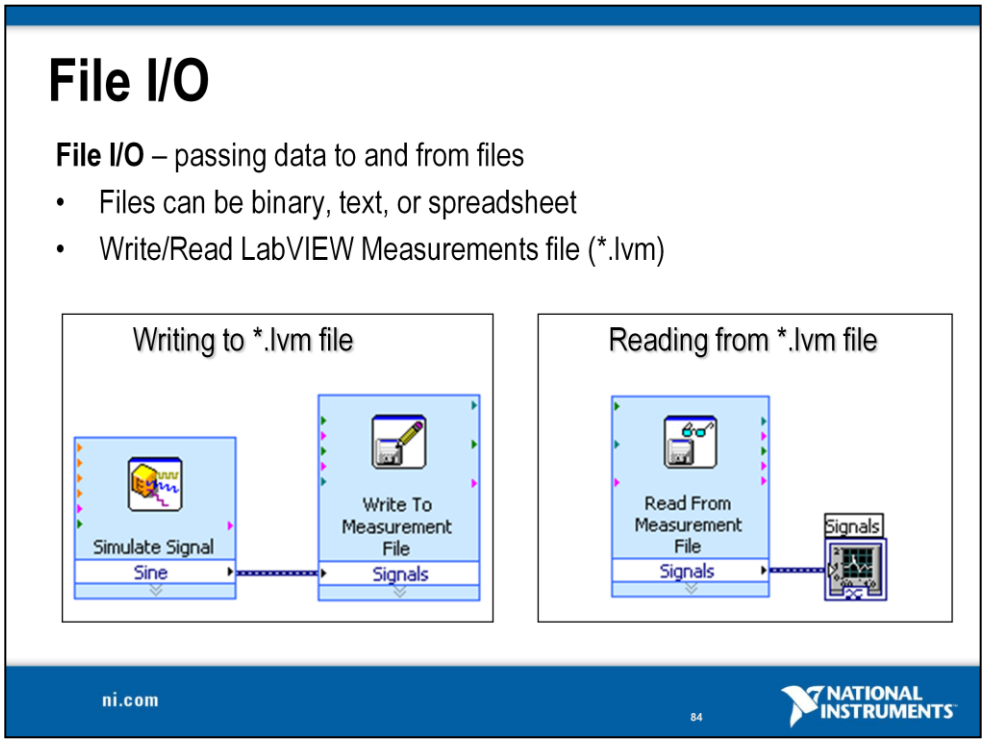
The Time Delay Express VI delays execution by a specified number of seconds. Following the rules of dataflow programming, the While Loop does not iterate until all tasks inside of it are complete, thus delaying each iteration of the loop.

Wait

The Wait VI waits until the specified number of milliseconds before allowing the While Loop to iterate.

Wait Until Next ms Multiple

This function waits until the value of the millisecond timer becomes a multiple of the specified millisecond multiple to help you 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. This function can be found at **Functions»Programming»Timing»Wait Until Next ms Multiple**.

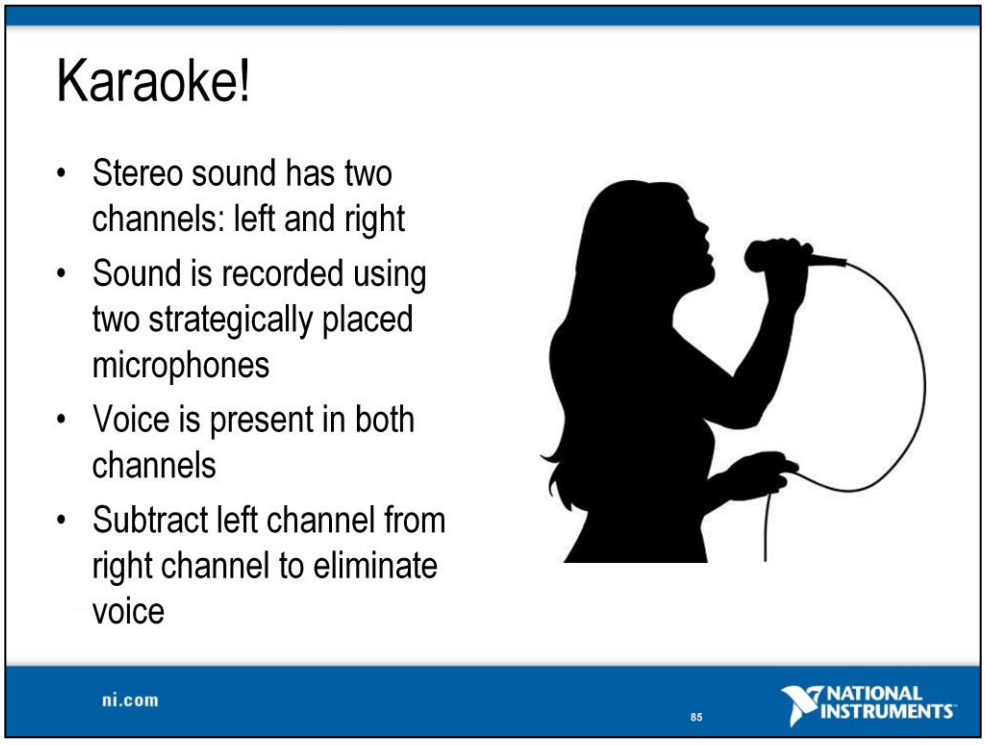


Use LabVIEW measurement data files to save data that the Write Measurement File Express VI generates. The LabVIEW data file is a tab-delimited text file you can open with a spreadsheet application or a text-editing application. In addition to the data an Express VI generates, the .lvm file includes information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

In the next example, examine how to write to or read from LabVIEW measurement files (*.lvm files).



Karaoke. Entertainment where the background music of popular songs is played and a performer sings live, usually by following the words on a video screen. Who can forget the precious memories that occur when groups go out to a Karaoke bar. You can learn so much about your friends, such as if they have a hidden talent for singing or if they are completely and utterly tone deaf.

Stereo sound is composed of two or more audio channels so that it appears as if the sound is coming in from multiple directions. To create this effect, an audio engineer may position microphones in different locations in the recording studio to put an emphasis on different sounds. When the audio is played back, you hear sounds from each channel. This is often the case for capturing a singer's voice and the accompanying instrumentation.

To create a Karaoke effect in our project, we will use our myDAQ's AudioInputLeft and AudioInputRight and subtract the Left Input from the Right Input to simulate a Karaoke effect. This process will reduce or remove some of the audio and mimic the functionality of a Karaoke machine.

Exercise 5 – Creating a Karaoke Setting(Track A)

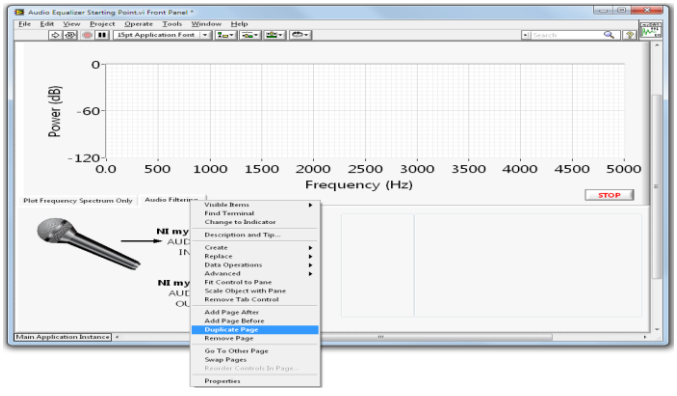
Goal Use advanced signal manipulation to remove vocal track from song.

Part 1

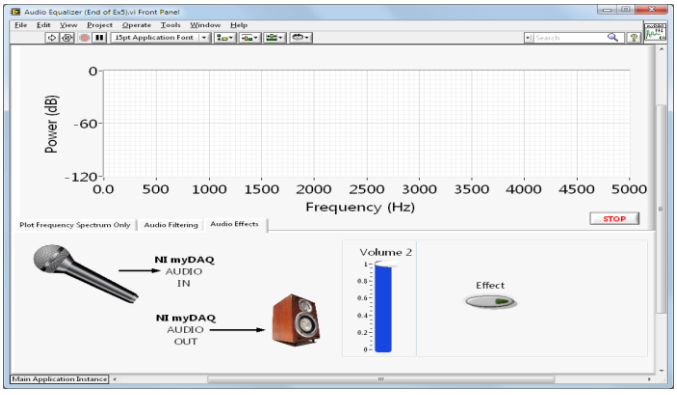
Description Now that we have a working audio equalizer, what else can we do? Instead of isolating a particular frequency band, we can try to remove the vocal track from a song to create a "karaoke" setting in our program. Before we can do this, we'll need to modify the front panel and case structure to accommodate this new functionality.

Procedure

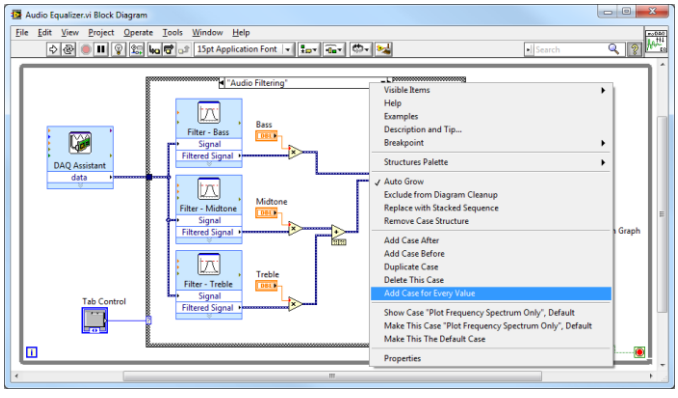
1. On the front panel, select the Audio Filtering tab.
2. Right-click on the tab heading and select **Duplicate Page**. This will create a new tab called Audio Filtering 2.



3. Change the title of the new tab to Audio Effects. You can change the title by double-clicking the title.
4. This new tab contains duplicates of the four slides we already created. Leave Volume 2 as is, but delete the new Bass 2, Midtone 2, and Treble 2 slides.
5. In place of the slides we removed, place a push button. This can be found at **Modern»Boolean»Push Button**. Name the push button Effect. Your finished front panel should look like the one shown on the next page.



6. Add a new case to the case structure
   a. On the block diagram, right-click the border of the case structure.
   b. Select **Add Case for Every Value** from the right-click menu. This is shown below.



7. Examine the case structure. A third case called Audio Effects has been added. Place the Volume 2 and Effect controls inside this case.

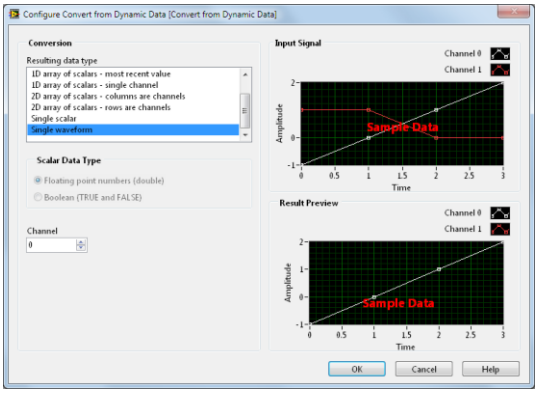
Why did selecting “add case for every value” automatically name the new case? The Run button shows a broken arrow icon, indicating that the VI cannot currently run; do you know why not? If not, try clicking the **Run** button. This will open an Error List that lists items that must be resolved for the VI to run.

Part 2

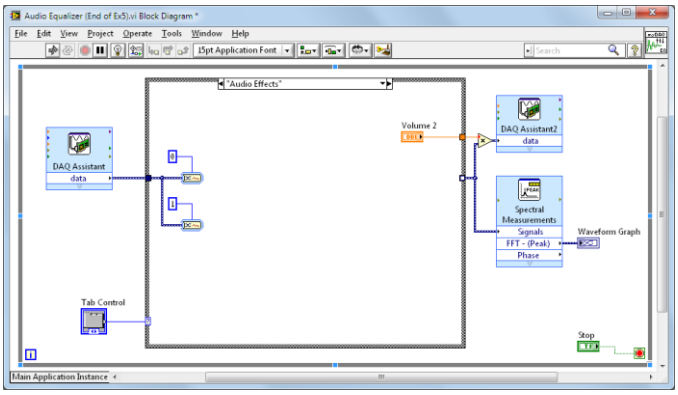
Description Now that we have modified the architecture of the VI, we can add new functionality. Many music recordings have different musical elements on the left and right tracks, but the same vocal elements. By (quite literally) subtracting one track from the other, we can remove the matched vocal track from the output signal. In order to do this, we'll need to split the signal into its two component tracks.

Procedure

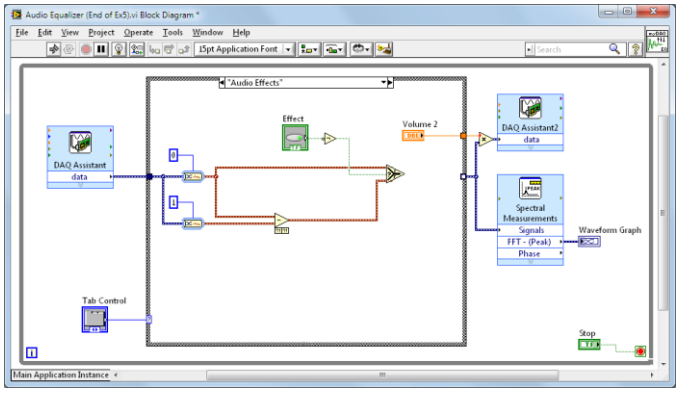
1. Wire the Volume 2 control terminal into the orange tunnel on the right side of the case structure.
2. Separate the input signal into left and right tracks.
   a. Place a Convert From Dynamic Data function inside the Audio Effects case. This function is under **Express»Signal Manipulation»From DDT**.
   b. When you place the function on the block diagram, a configuration window will appear. On the Resulting data type list, select **Single waveform**. Leave **Channel** as 0.



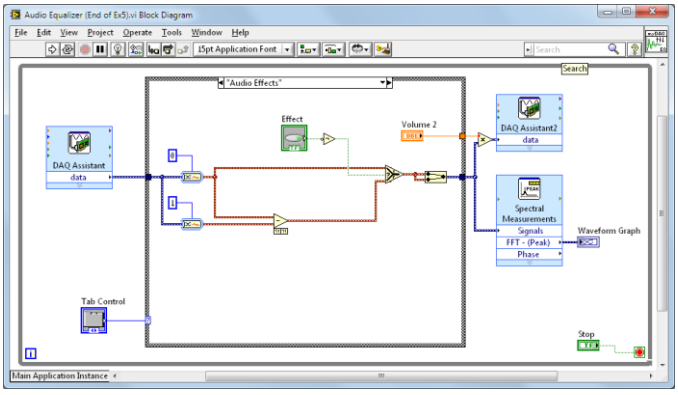
   c. Right-click on the Convert From Dynamic Data function's **Channel** terminal and select **Create»Constant**. Leave it as the default 0.
   d. Create a second Convert From Dynamic Data function. Use a constant to set its **Channel** to 1.
   e. Wire the input signal into the **Dynamic Data Type** terminal on both conversion functions. It should look like the image on the next page.



3. Place a Subtract, a Select, and a Not function on the Block Diagram. These are under **Programming** in **Numeric»Subtract**, **Comparison»Select**, and **Boolean»Not**.
4. Wire the two conversion **Waveform** terminals into the Subtract function's inputs (**x** and **y**), and the Subtract output (**x-y**) into the false (**f**) terminal of the Select function.
5. In addition, wire the **Waveform** terminal of either conversion function directly into the true (**t**) terminal of the Select function.
6. Wire the Effect control terminal into the input (**x**) terminal of the Not function, and the output (**not x**) into the selector (**s**) terminal of the Select function.



7. Place a Merge Signals function on the block diagram from **Express»Signal Manipulation**. Wire the Select function's output terminal into both Merge Signals inputs (**Signal 1** and **Signal 2**), and the **combined signal** output terminal into the signal out data tunnel. Your VI should look like the one on the next page.



8. Return to the front panel and run the VI. Try using one of the supplied audio files (*Great Wide Open* or *Don't Worry, Be Happy*) to see how turning the Effect button on and off changes the output signal.

Did the VI behave how you expected it to? What is the Not function accomplishing? If you wanted to eliminate the Not function, how could you change the VI? What is the purpose of the Merge Signals function? Would the VI run normally without it?

End of Exercise 5 (Track A)

Exercise 5 – Creating a Karaoke Setting(Track B)

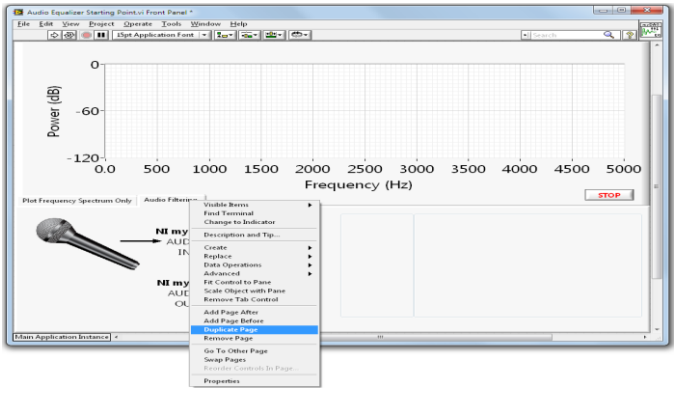
Goal Use advanced signal manipulation to remove vocal track from song.

Part 1

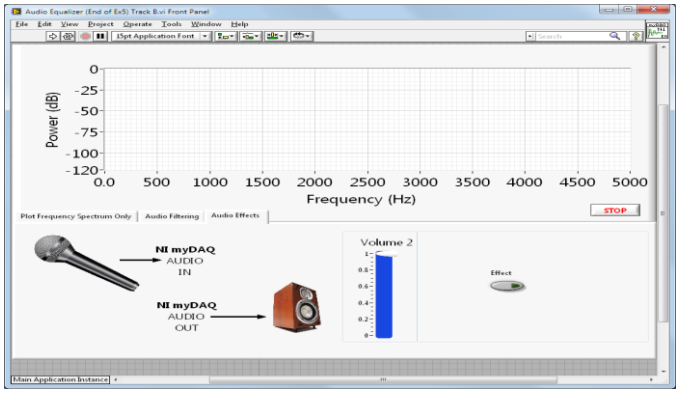
Description Now that we have a working audio equalizer, what else can we do? Instead of isolating a particular frequency band, we can try to remove the vocal track from a song to create a "karaoke" setting in our program. Before we can do this, we'll need to modify the front panel and case structure to accommodate this new functionality.

Procedure

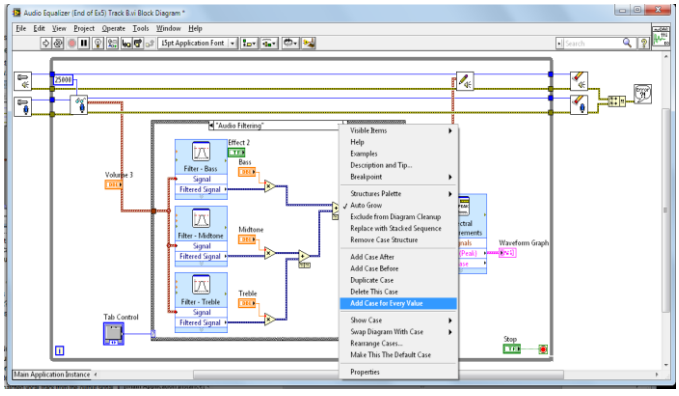
1. On the front panel, select the Audio Filtering tab.
2. Right-click on the tab heading and select **Duplicate Page**. This will create a new tab called Audio Filtering 2.



3. Change the title of the new tab to Audio Effects. You can change the title by double-clicking on the title.
4. This new tab contains duplicates of the four slides we already created. Leave Volume 2 as is, but delete the new Bass 2, Midtone 2, and Treble 2 slides.
5. In place of the slides we removed, place a push button. This can be found at **Modern»Boolean»Push Button**. Name the push button Effect. Your finished front panel should look like the one on the next page.



6. Add a new case to the case structure.
   a. On the block diagram, right-click the border of the case structure.
   b. Select **Add Case for Every Value** from the right-click menu. This is shown below.



7. Examine the case structure. A third case called Audio Effects has been added. Place the Volume 2 and Effect controls inside this case.

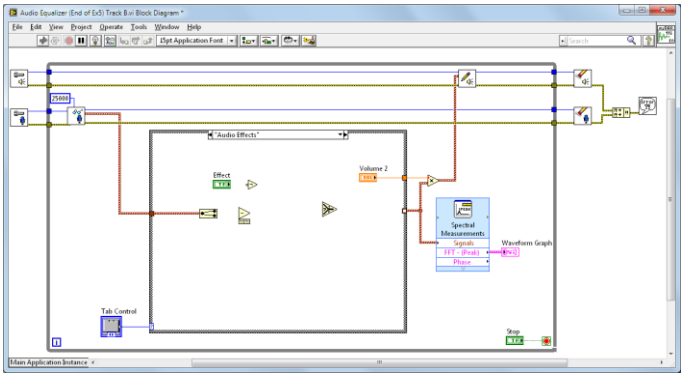
Why did selecting "add case for every value" automatically name the new case? The Run button shows a broken arrow icon, indicating that the VI cannot currently run; do you know why not? If not, try clicking the **Run** button. This will open an Error List that lists items that must be resolved for the VI to run.

Part 2

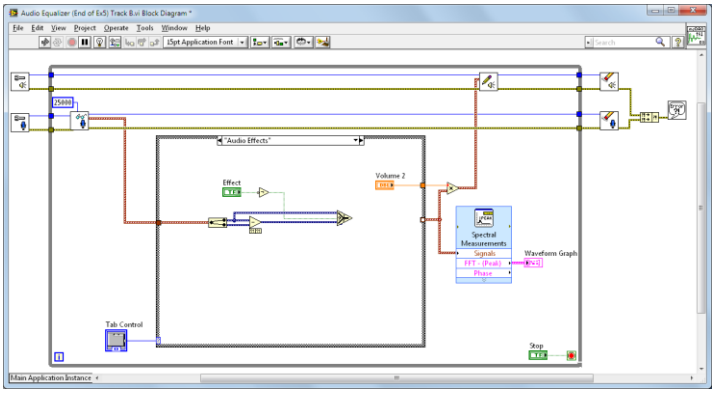
Description Now that we have modified the architecture of the VI, we can add new functionality. Many music recordings have different musical elements on the left and right tracks, but the same vocal elements. By (quite literally) subtracting one track from the other, we can remove the matched vocal track from the output signal. In order to do this, we'll need to split the signal into its two component tracks.

Procedure

1. Wire the Volume 2 control terminal into the orange tunnel on the right side of the case structure.
2. Separate the input signal into left and right tracks.
   a. Place a Split Signals VI on the block diagram. This can be found on the Functions palette on the **Express»Signal Manipulation.**
   b. Wire the data output from the case structure to the input of the split signal.
3. Place a Subtract, a Select, and a Not function on the block diagram. These are under **Programming** in **Numeric»Subtract**, **Comparison»Select**, and **Boolean»Not**.

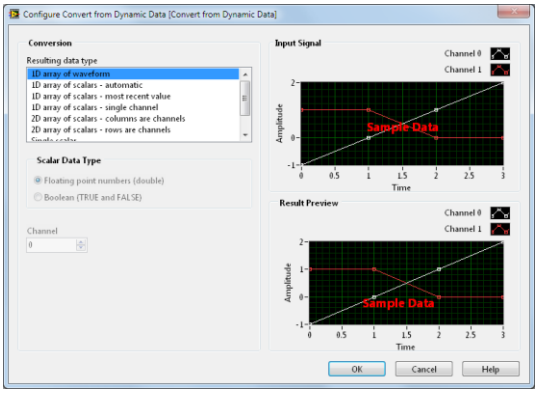


4. Wire the two split signal outputs into the Subtract function's inputs (**x** and **y**), and the Subtract output (**x-y**) into the false (**f**) terminal of the Select function.
5. In addition, wire the top split signal to the true (**t**) terminal of the Select function.
6. Wire the Effect control terminal into the input (**x**) terminal of the Not function, and the output (**not x**) into the selector (**s**) terminal of the Select function.

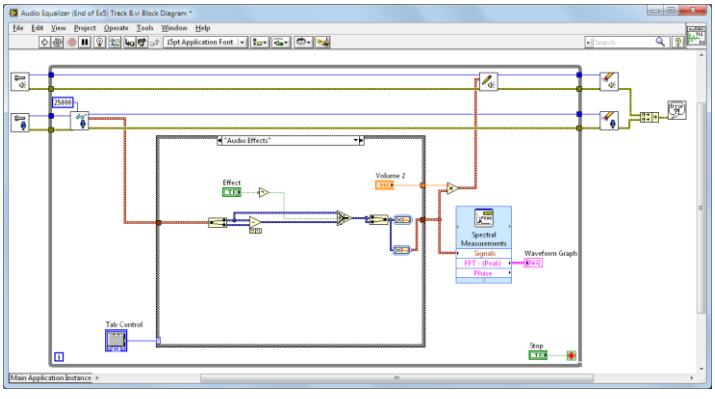


7. Place a Merge Signals function on the block diagram from **Express»Signal Manipulation**. Wire the Select function's output terminal into both Merge Signals inputs (**Signal 1** and **Signal 2**).

8. Convert the Dynamic Data type to a Waveform.
   a. Place down a Convert From Dynamic Data function. This can be found on the Functions palette by navigating to **Express»Signal Manipulation.**
   b. A dialog box will display. Under Resulting Data Type, select **1D Array of Waveform** and then click **Okay**.



   c. Wire the output of the Merge Signals VI to the input of the Convert From Dynamic Data Type.
9. Wire the output terminal into the signal out data tunnel of the case structure.



10. Return to the front panel and run the VI. Try using one of the supplied audio files (*Great Wide Open* or *Don't Worry, Be Happy*) to see how turning the Effect button on and off changes the output signal.

Did the VI behave how you expected it to? What is the Not function accomplishing? If you wanted to eliminate the Not function, how could you change the VI? What is the purpose of the Merge Signals function? Would the VI run normally without it?

End of Exercise 5 (Track B)

Video: myDAQ Optical Theramin

YouTube Link

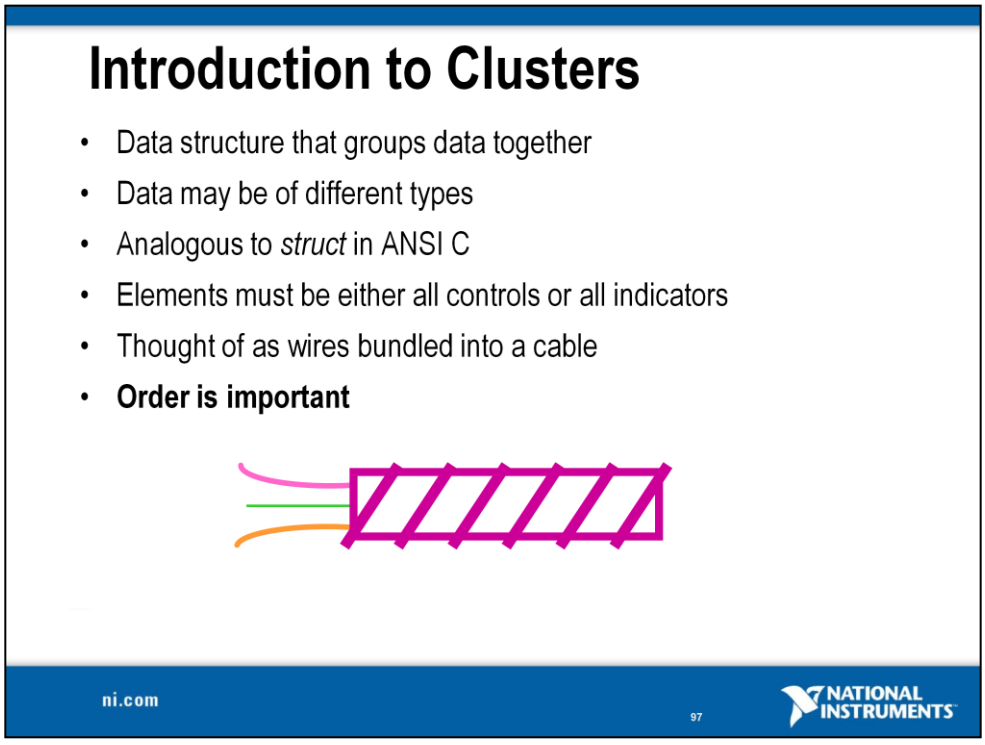
Don't forget to submit your project to **ni.com/studentdesign** for a chance to win prizes and a trip to Austin, TX

Section V – 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



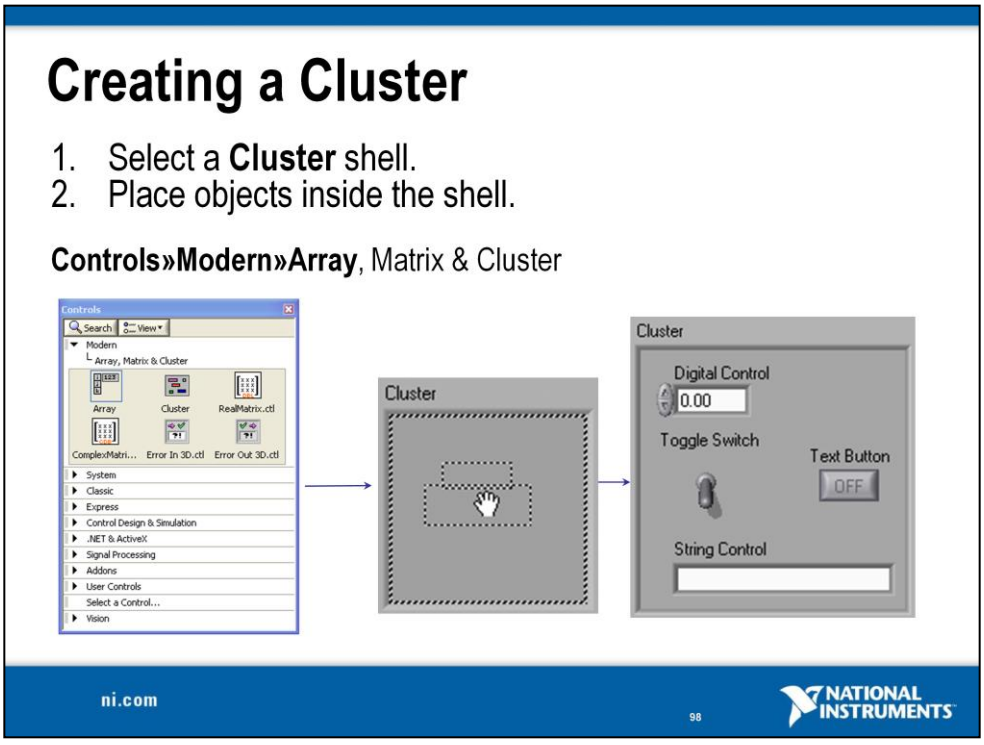
Clusters group like or unlike components together. They are equivalent to a *record* in Pascal or a *struct* in ANSI C.

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 the user's security code.

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



Create a cluster front panel object 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.

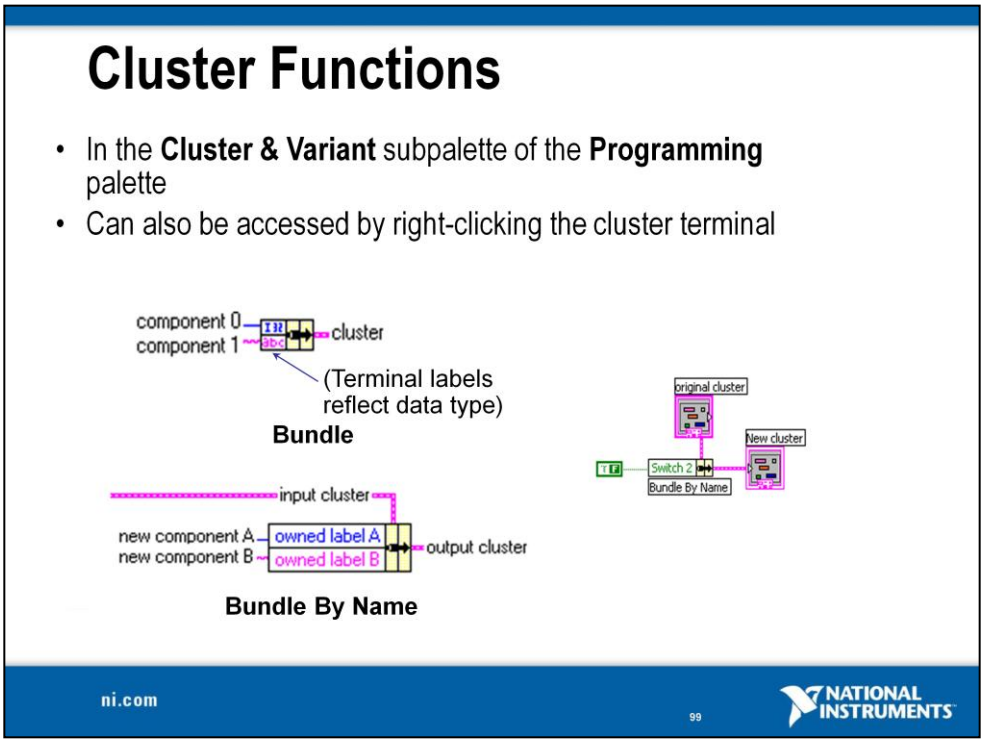
Note: You can even have a cluster inside of a cluster.

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.



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.

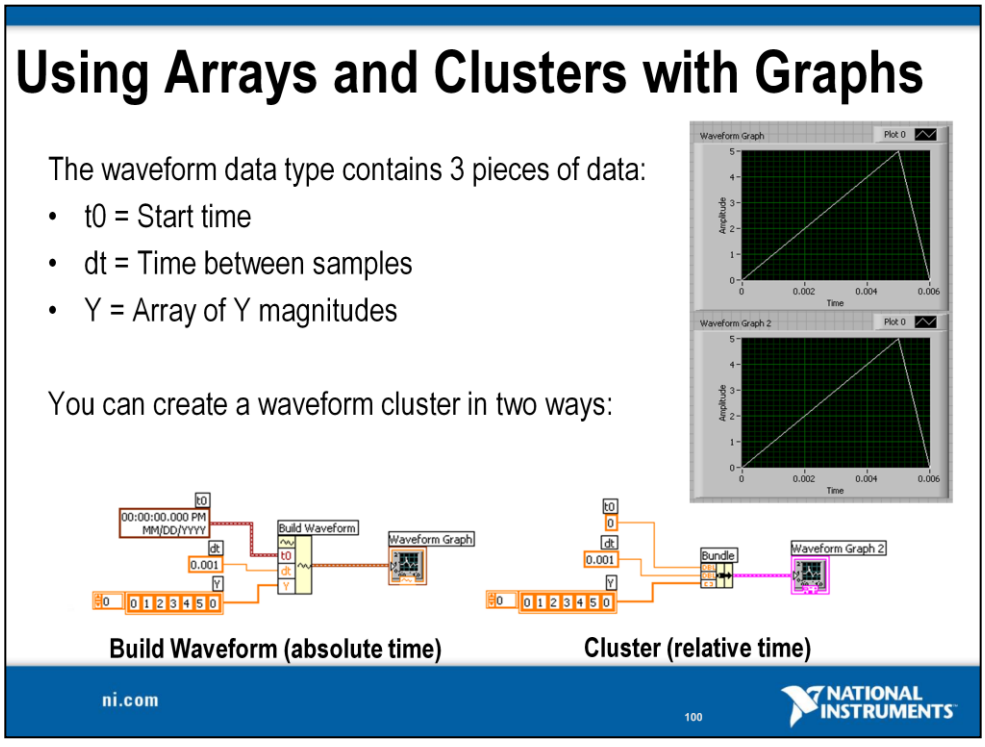
Bundle – Forms a cluster containing the given objects in the specified order.

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

Unbundle – Splits a cluster into each of its individual elements by data type.

Unbundle by Name – Returns the cluster elements whose names you specify.

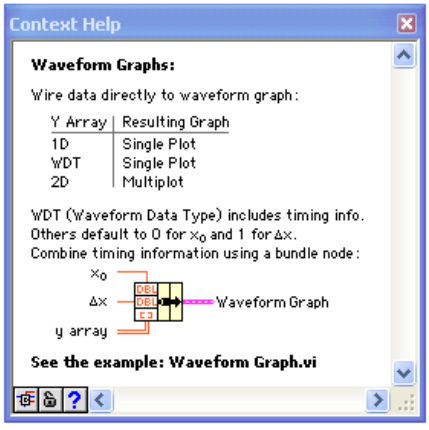
Note: You must have an existing cluster wired into the middle terminal of the function to use Bundle By Name.



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. Timestamps are accurate to real-world time and date and are very useful for real-world data recording.

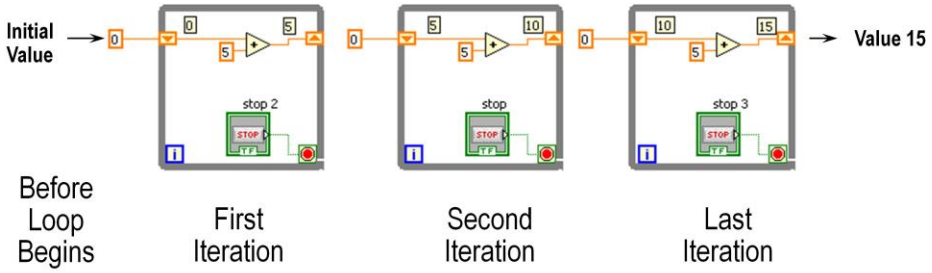


Bundle

Builds a waveform or modifies an existing waveform with a relative timestamp. The input to t_0 is a DBL. By building waveforms using the bundle, you can plot data on the negative x-axis (time).

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

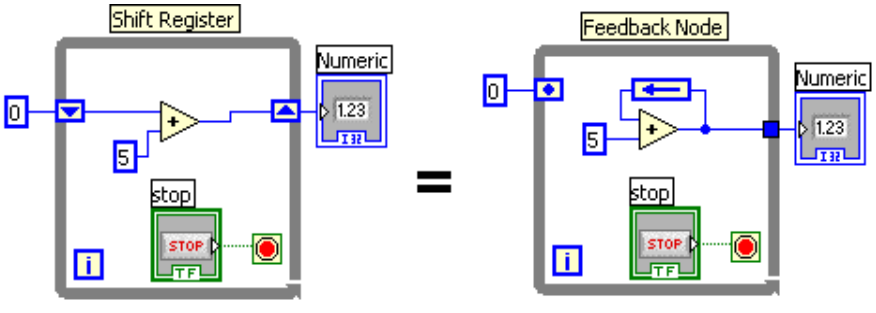


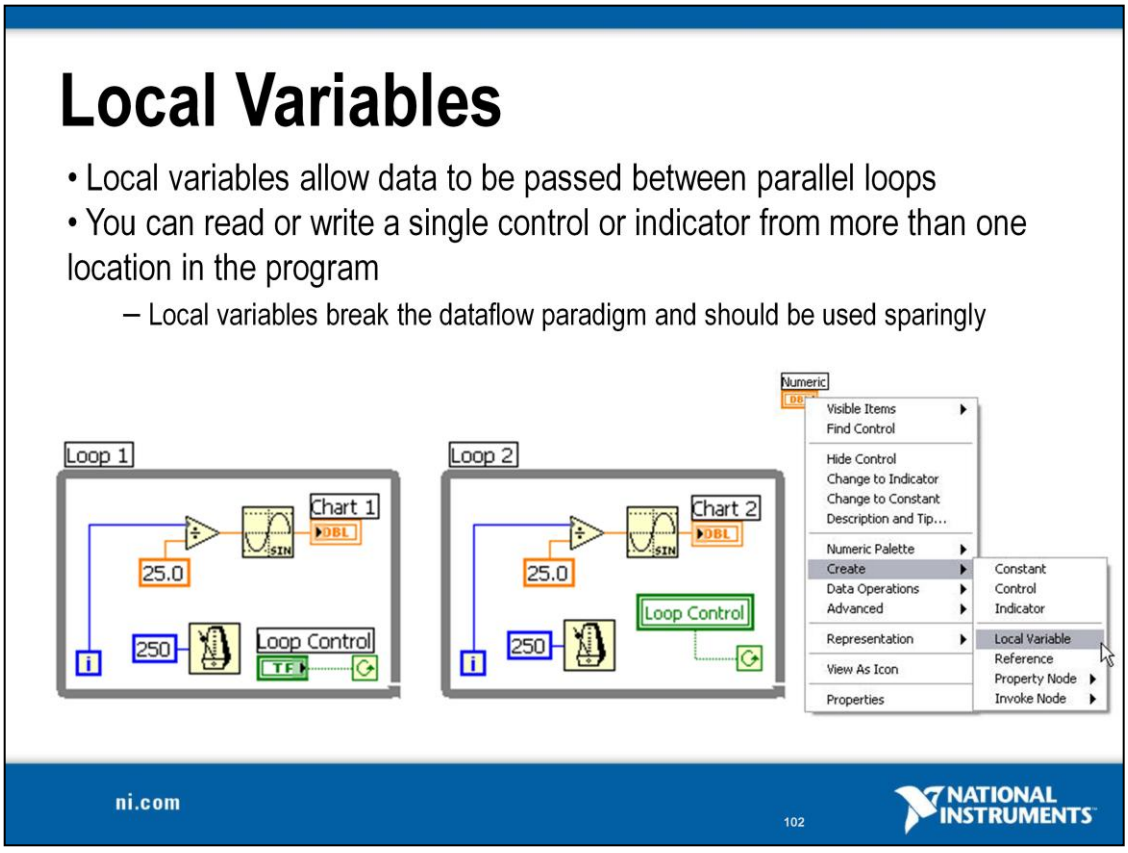
ni.com 101 NATIONAL INSTRUMENTS

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 for the first iteration, 10 for the second iteration, and 15 for the third iteration. Said another way, you use shift registers 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 (pictured below) is another representation of the same concept. Both programs pictured behave the same.



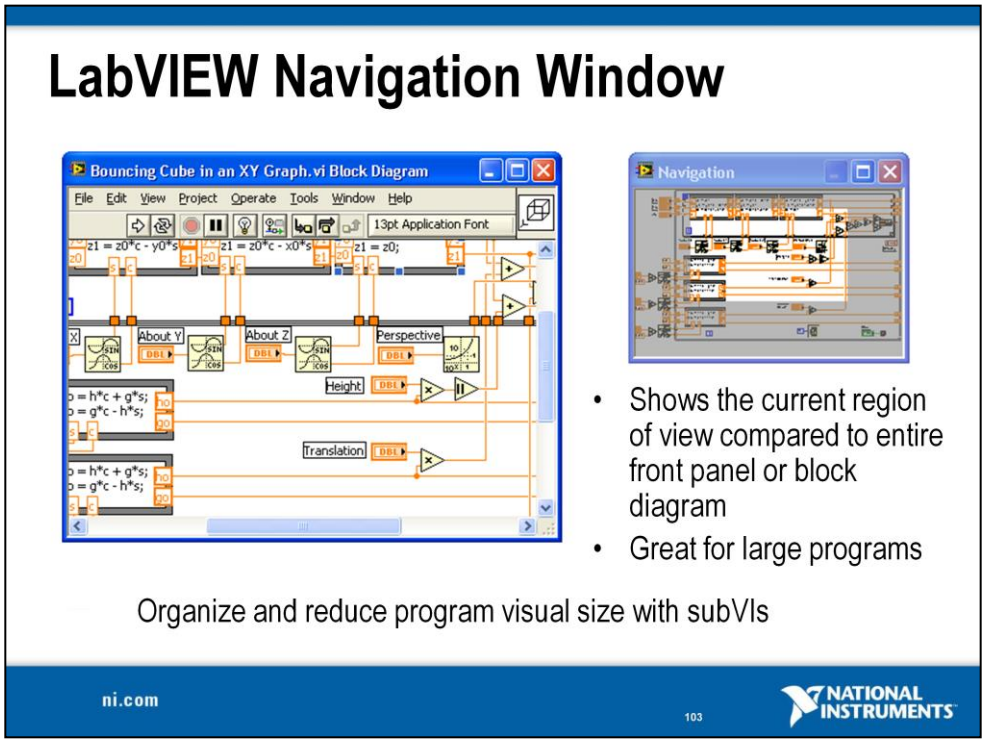


Sometimes you may need to access a front panel object from more than one place on the block diagram or to pass data between structures that you cannot connect by a wire. To accomplish these tasks, you would use a *local variable.*

Local variables are located on the **Functions** palette under **Programming»Structures**.

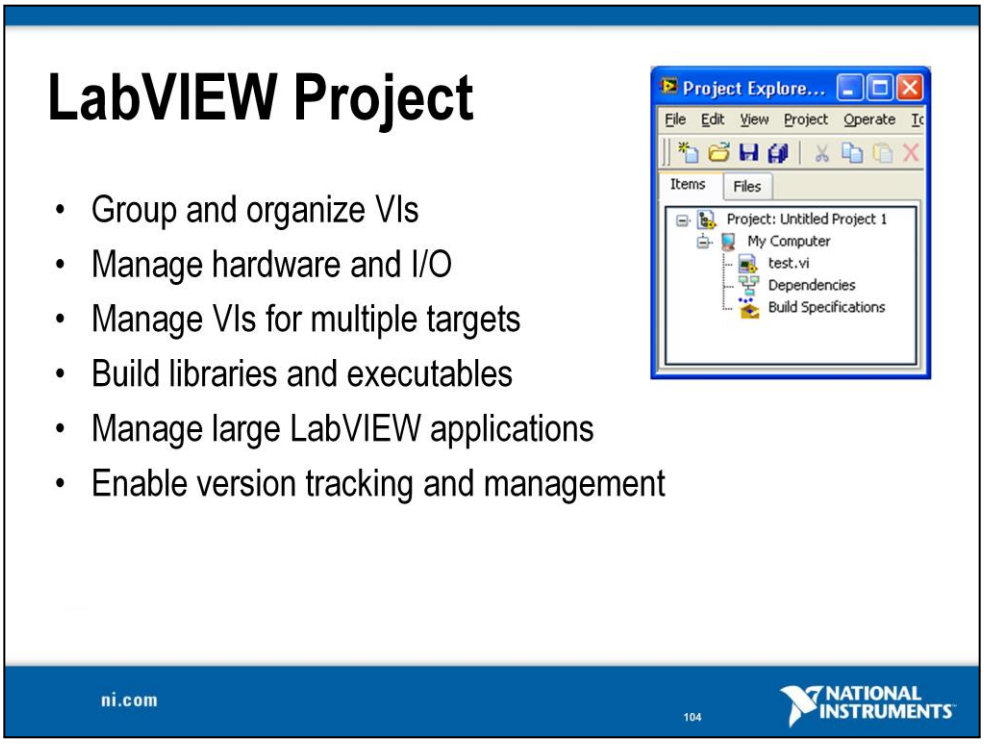
You use a local variable by first selecting the object you want to access. You can either click on the local variable with the Operating tool and select the object you want to access, or right-click on the local variable and choose the object from the **Select Item** menu.

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**.



Select **View»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

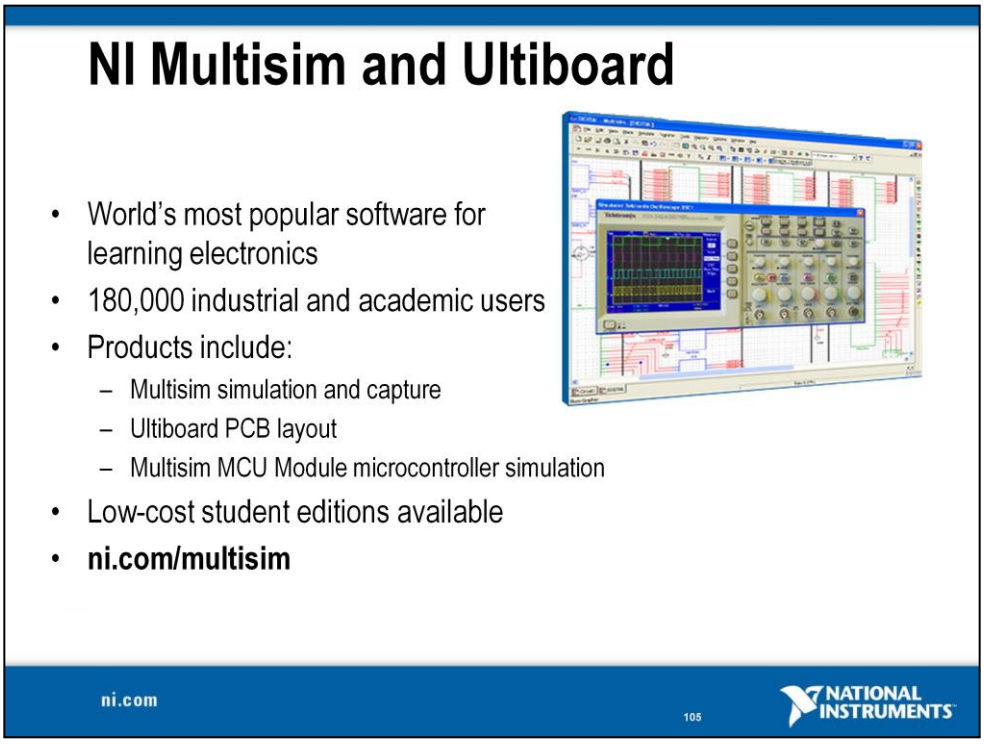
Use projects to group together LabVIEW and other files, create build specifications, and deploy or download files to targets. A target is a device or machine on which a VI runs. When you save a project, LabVIEW creates a project file (.lvproj), which includes configuration information, build information, 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 a real-time, FPGA, or PDA target. Refer to the specific module documentation for more information about using projects with the LabVIEW Real-Time, LabVIEW FPGA, and LabVIEW Mobile modules.

Project-style LabVIEW Plug and Play instrument drivers use the project and project library features. 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.



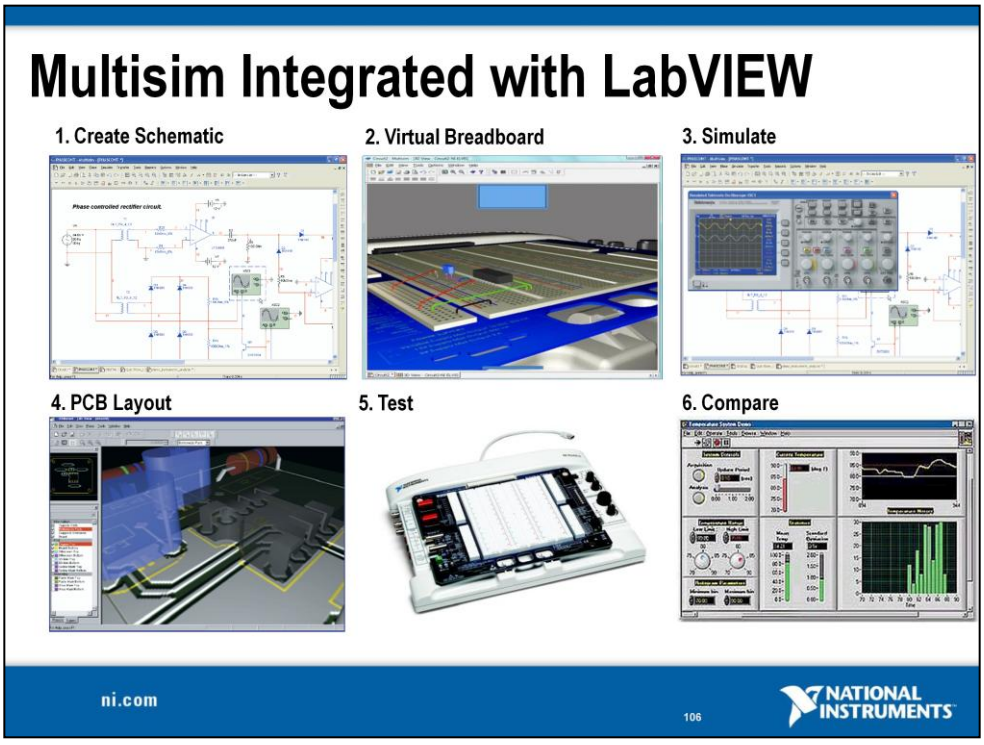
Multisim software integrates powerful SPICE simulation and schematic capture entry into a highly intuitive electronics lab on the PC. As the only company to design products specifically for the education market, National Instruments provides software that has become a teaching and learning tool of choice for thousands of educators.

Multisim – Simulation and Capture

Multisim is an intuitive, drag-and-drop schematic capture and simulation program that educators and students can use to quickly create complete circuits containing both analog and digital components. Multisim also adds microcontroller unit cosimulation capabilities, so you can include an MCU, programmed in assembly code, within your SPICE-modeled circuit.

Ultiboard – PCB Layout

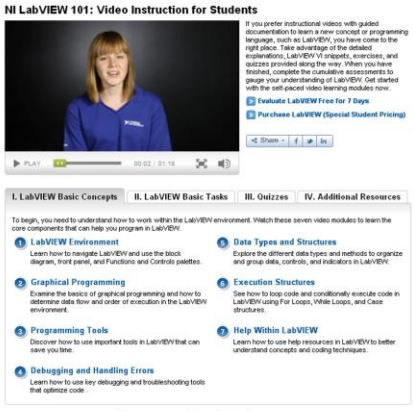
With Ultiboard, students can gain exposure to the physical implementation and manufacturing of circuits on PCBs. Students can import the Multisim schematic into Ultiboard with a single mouse click.



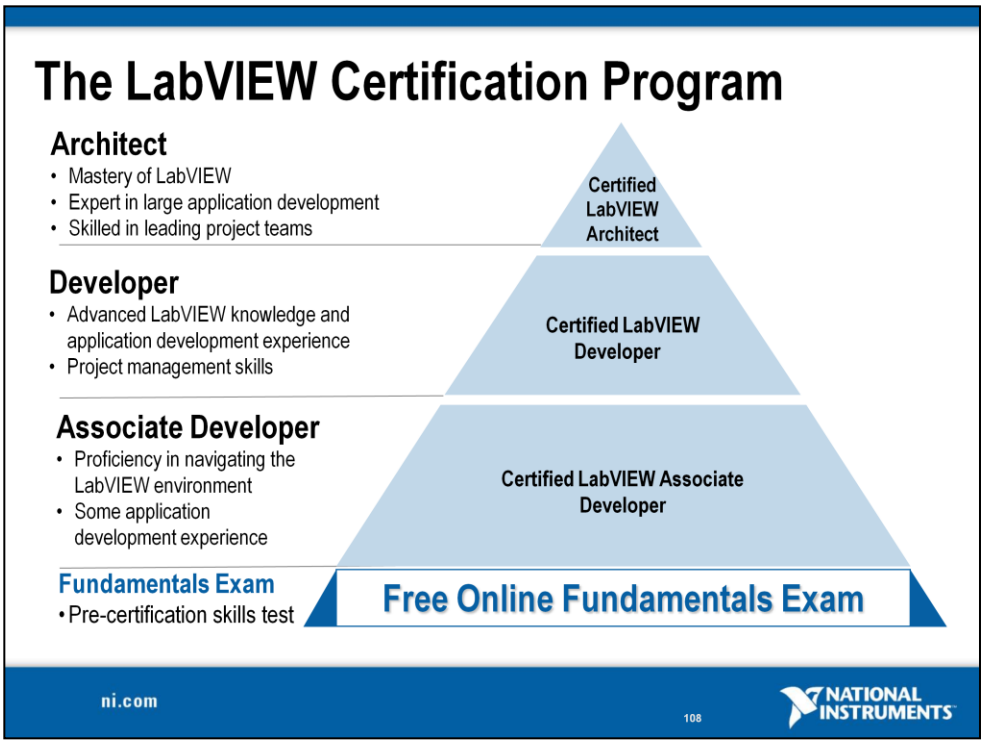
1. Multisim – Schematics
   - Easy-to-use schematics
   - Simple click and drag
   - 3D animated parts
   - Wire drag without breaking connections
2. Multisim – Virtual Breadboard
   - Breadboarding techniques
   - Synchronized with schematic
   - Wiring report for NI ELVIS (step 5)
3. Multisim – Simulation
   - 13,000+ part library
   - 22+ NI ELVIS virtual instruments
   - Changes on the fly
   - Microcontroller simulation
   - Animated parts (LEDs and 7-segment displays)
4. Ultiboard – PCB Layout
   - Integrated with Multisim
   - Flexible interface
   - 3D view
   - Design rule check
   - Built-in autorouting
5. NI ELVIS – Test
   - Instrumentation
   - Data acquisition
   - Prototyping
6. LabVIEW – Compare
   - Automatically import:
     - Multisim virtual data
     - NI ELVIS measured data
   - Compare ideal and real data

Additional Resources

- NI Academic Web and Student Corner
  - ni.com/students
  - ni.com/lv101
  - ni.com/textbooks
  - Get your own copy of the LabVIEW Student Edition
- NI KnowledgeBase
  - ni.com/kb
- NI Developer Zone
  - ni.com/devzone
- LabVIEW Certification
  - LabVIEW Fundamentals Exam (free on ni.com/academic)
  - Certified LabVIEW Associate Developer Exam (industry-recognized certification)



ni.com/lv101



Today, more and more companies and hiring managers are looking for LabVIEW expertise when they evaluate job candidates. 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 a student, your Certified LabVIEW Associate Developer certification differentiates your LabVIEW skills for employment opportunities and helps potential employers recognize your LabVIEW expertise. The CLAD is a one-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 professional certifications 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 four-hour practical exams are conducted by National Instruments.

The NI LabVIEW Academy provides classroom curriculum and hands-on exercises to colleges and universities. After completion of the LabVIEW Academy program, students have the knowledge and tools to attempt the Certified LabVIEW Associate Developer certification exam with confidence.

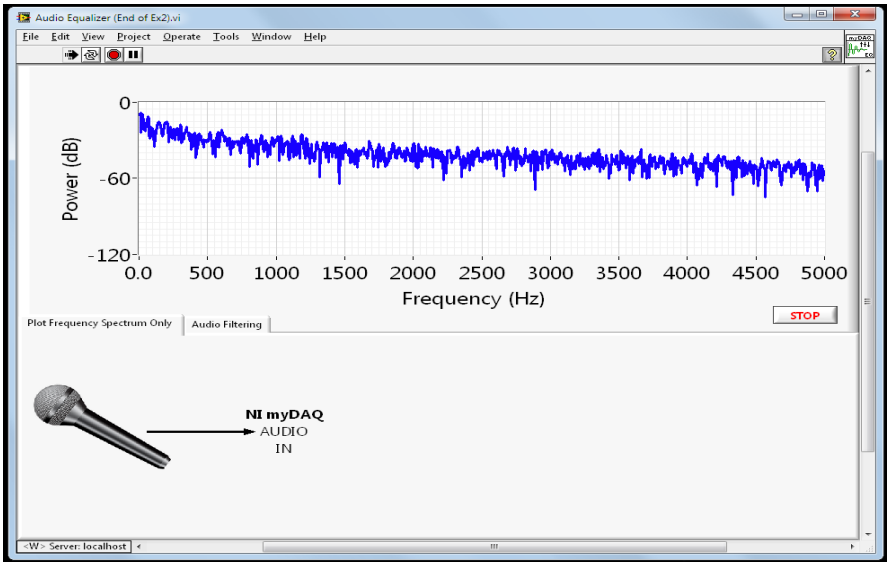
Your Next Step

LabVIEW Skills Evaluation Quiz:
https://lumen.ni.com/nicif/us/infolvcoursefinder/content.xhtml
CLAD Exam Practice:
https://lumen.ni.com/nicif/us/ekitcladexmprp/content.xhtml
CLAD Exam Prep Webcast:
http://zone.ni.com/wv/app/doc/p/id/wv-566

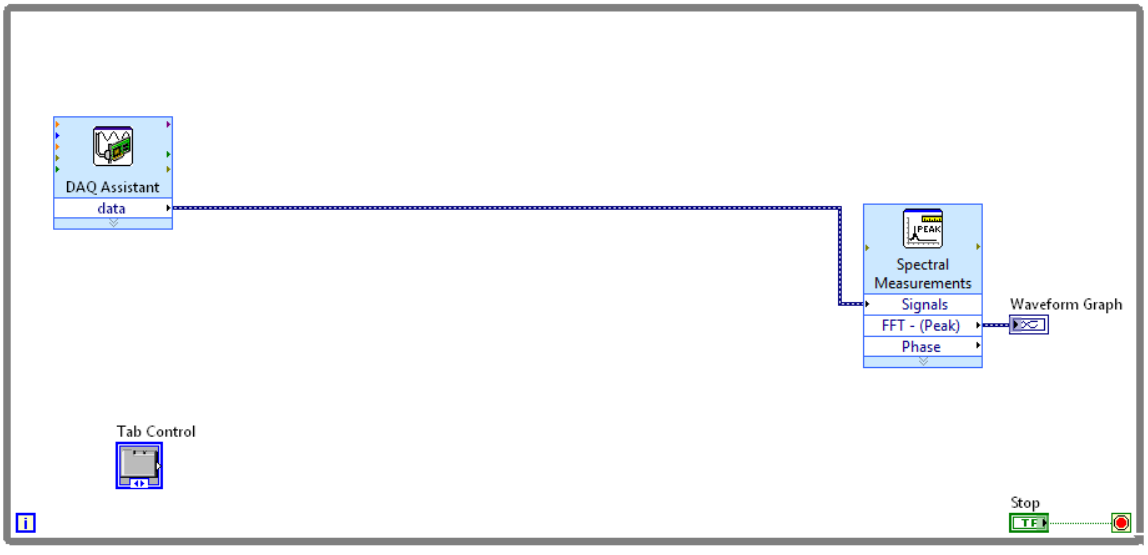
Your first step to LabVIEW certification!

Solutions Section

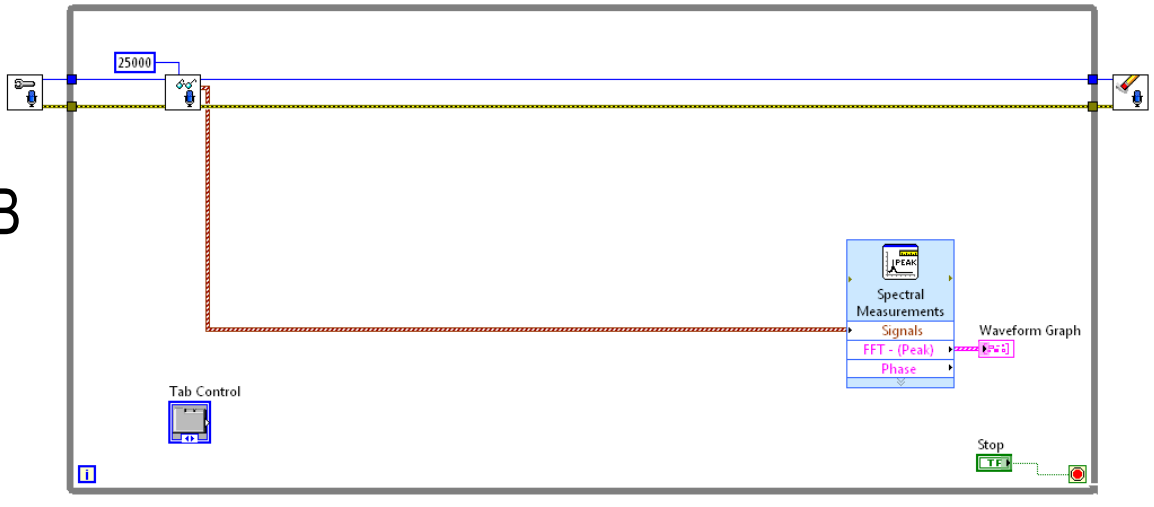
Exercise 2



Track A Code

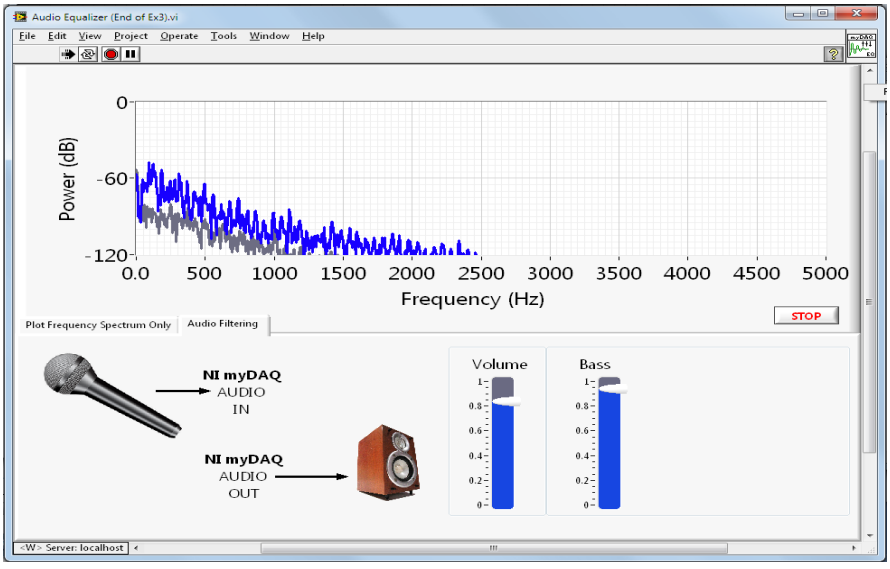


Track B Code

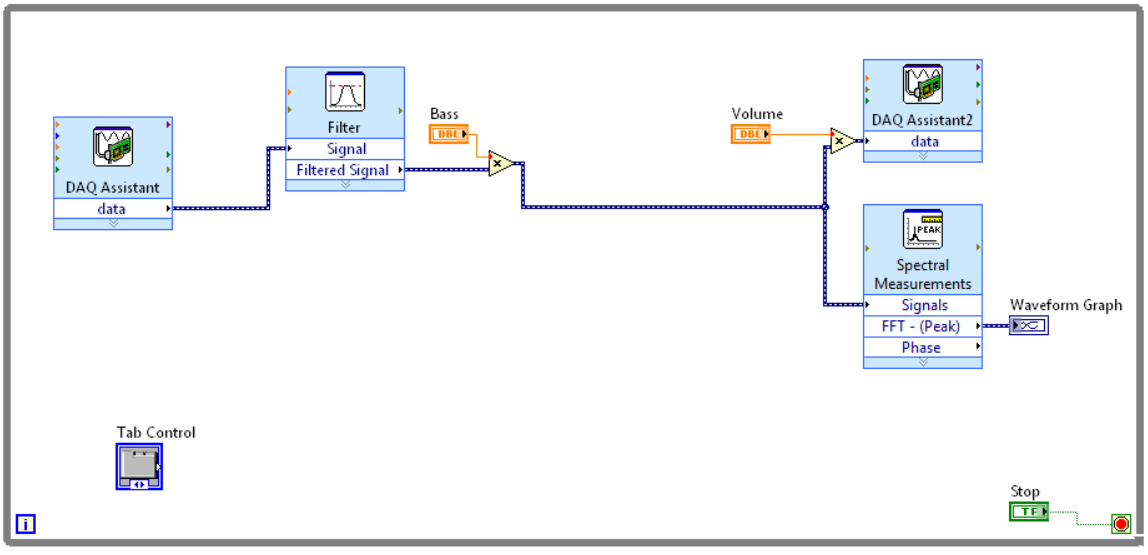


Solutions Section

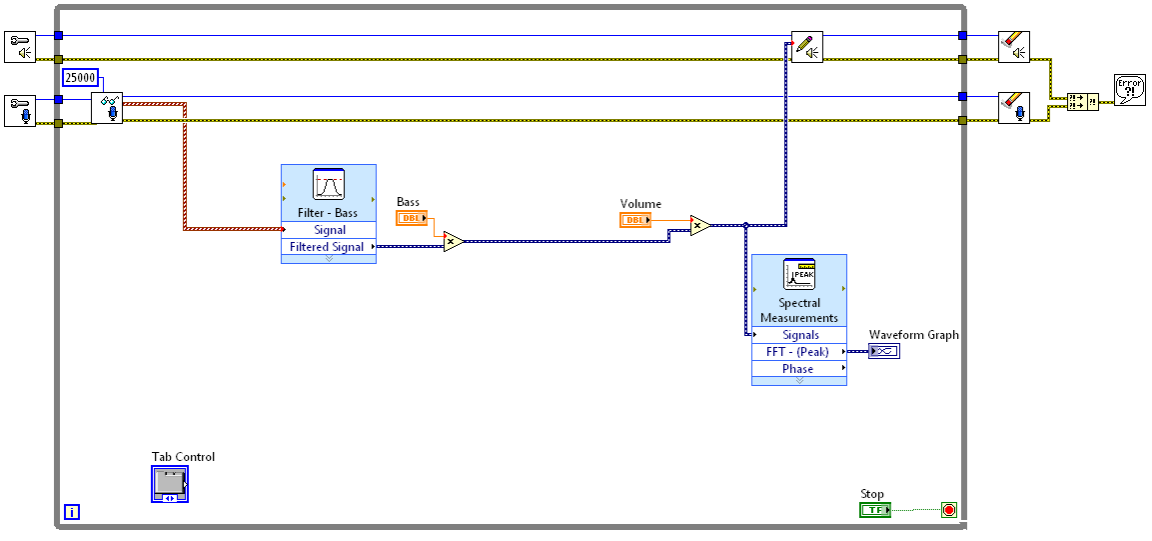
Exercise 3



Track A Code



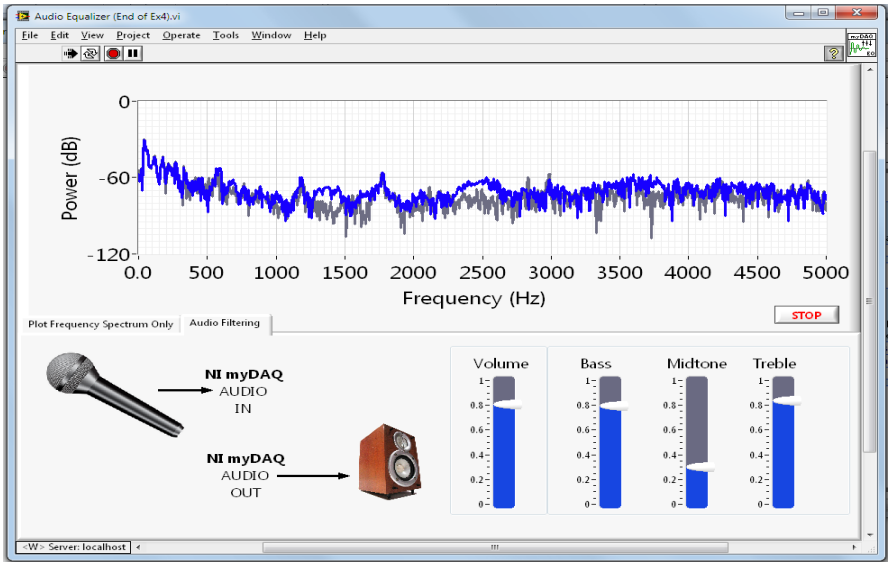
Track B Code



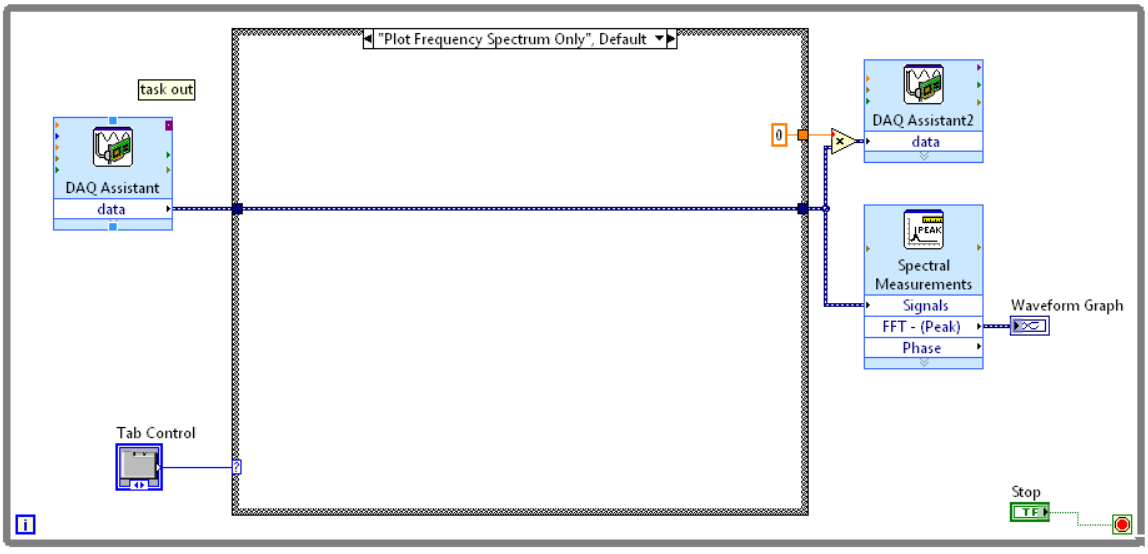
Solutions Section

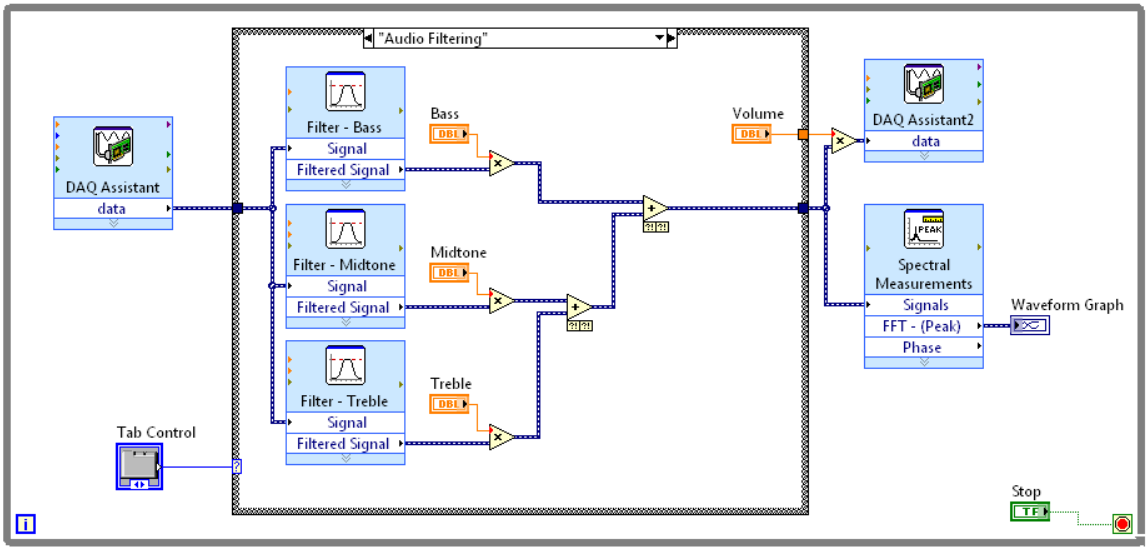
Exercise 4

Track A



Track A Code

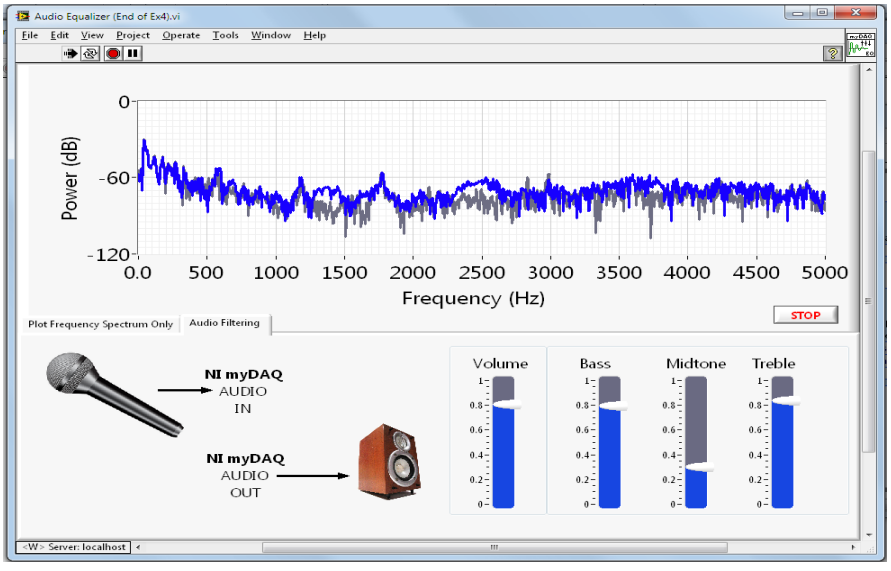




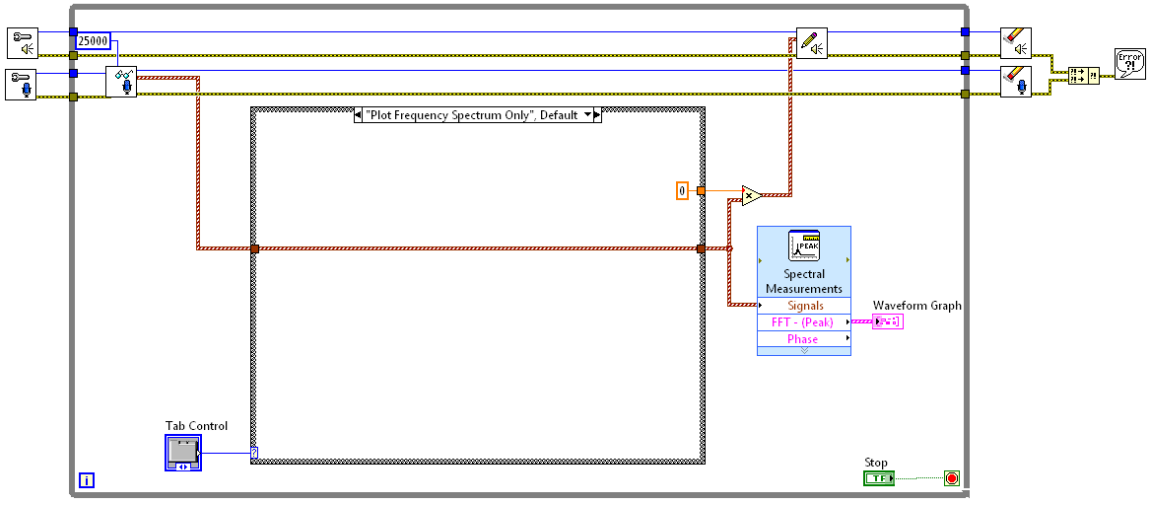
Solutions Section

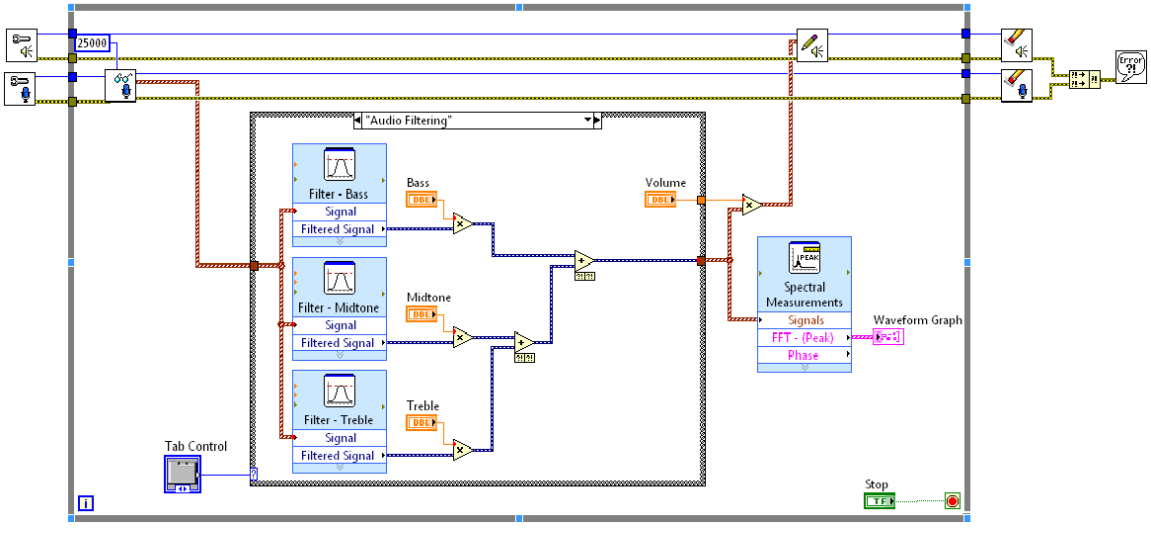
Exercise 4

Track B



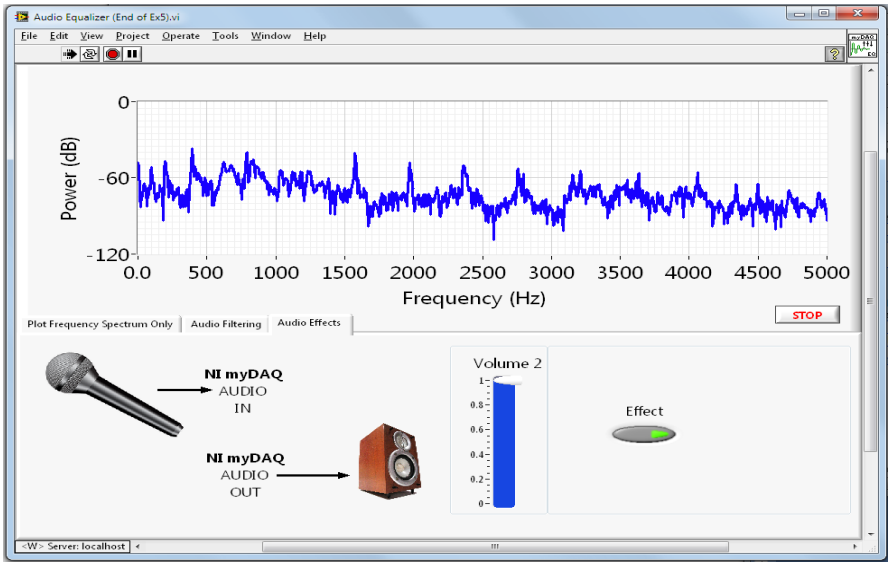
Track B Code



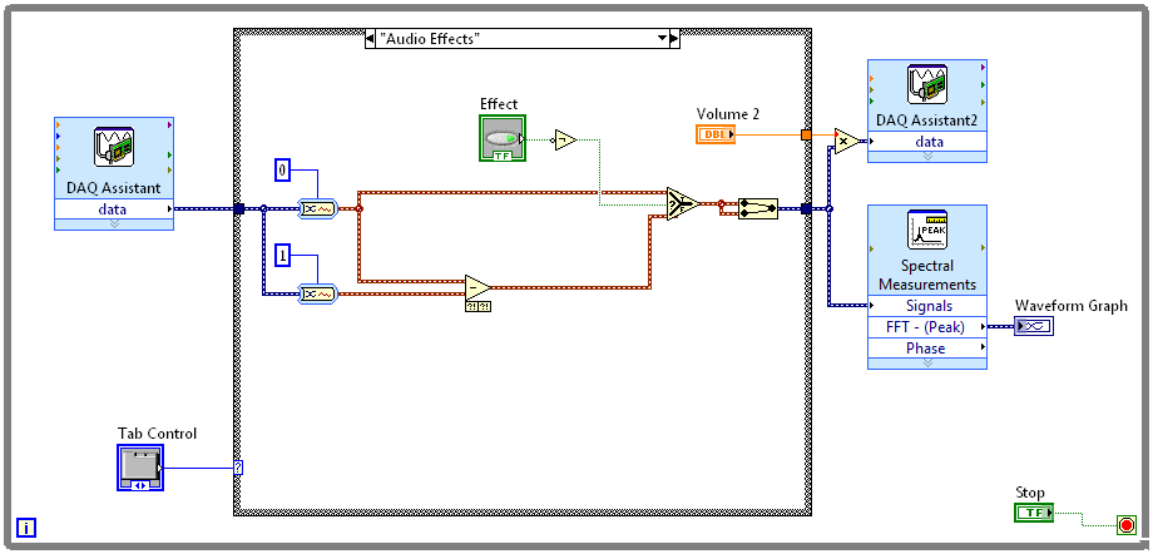


Solutions Section

Exercise 5



Track A Code



Track B Code