Virtual Instrumentation With LabVIEW
Section I

• LabVIEW terms
• Components of a LabVIEW application
• LabVIEW programming tools
• Creating an application in LabVIEW
LabVIEW programs are called virtual instruments (VIs).

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

The Front Panel is used to interact with the user when the program is running. Users can control the program, change inputs, and see data updated in real time. Stress that controls are used for inputs- 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 indicate values from the program. These may include data, program states, and other information.

Every front panel control or indicator has a corresponding terminal on the block diagram. When a VI is run, values from controls flow through the block diagram, where they are used in the functions on the diagram, and the results are passed into other functions or indicators.
The front panel is the user interface of the VI. You build the front panel with controls and indicators, which are the interactive input and output terminals of the VI, respectively. Controls are knobs, pushbuttons, dials, and other input devices. Indicators are graphs, LEDs, and other displays. Controls simulate instrument input devices and supply data to the block diagram of the VI. Indicators simulate instrument output devices and display data the block diagram acquires or generates.

In this picture, the **stop button** is a boolean control. A boolean contains either a true or false value. The value is false until the button is pressed. When the button is pressed, the value becomes true. The **thermometer** is a numeric indicator. The data type is double. The level displayed on the thermometer is equal to the value of the number passed into the indicator. The **temperature history** indicator is a waveform graph. It displays multiple numbers. In this case, there are 205 values on the graph (0-204), all of which form a trace on the plot area.

The front panel also contains a toolbar, whose functions we will discuss later.
The block diagram contains this graphical source code. Front panel objects appear as terminals on the block diagram. Additionally, the block diagram contains functions and structures from built-in LabVIEW VI libraries. Wires connect each of the nodes on the block diagram, including control and indicator terminals, functions, and structures.

In this block diagram, the **Call to subVI** calls another VI as a subroutine. The results of the subroutine flow into both the **Thermometer Terminal** and **Temperature Graph**, which are front panel indicators. Values from the **Knob Terminal** and the **Numeric Constant** flow into the **While Loop**, which executes until a true value is supplied to the **Stop Loop Terminal** from the **Stop Button Terminal**.
Use the **Controls** palette to place controls and indicators on the front panel. The **Controls** palette is available only on the front panel. Select **Window»Show Controls Palette** or right-click the front panel workspace to display the **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. Select **Window»Show Functions Palette** or right-click the block diagram workspace to display the **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.
If automatic tool selection is enabled and you move the cursor over objects on the front panel or block diagram, LabVIEW automatically selects the corresponding tool from the Tools palette. Toggle automatic tool selection by clicking the Automatic Tool Selection button in the Tools palette or by pressing the <Shift-Tab> keys. In this class we will discuss the four most common tools in LabVIEW: The Operating, Positioning/Resizing, Labeling, and Wiring Tools.

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 resizeable 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.

When automatic tool selection is disabled, you can alternate among tools on the Tools palette, by pressing the <Tab> key. To toggle between the Positioning and Wiring tools on the block diagram or the Positioning and Operating tools on the front panel, press the spacebar.
• Click the **Run** button to run the VI. While the VI runs, the **Run** button appears with a black arrow if the VI is a top-level VI, meaning it has no callers and therefore is not a subVI.

• Click the **Continuous Run** button to run the VI until you abort or pause it. You also can click the button again to disable continuous running.

• While the VI runs, the **Abort Execution** button appears. Click this button to stop the VI immediately.

    **Note:** Avoid using the **Abort Execution** button to stop a VI. Either let the VI complete its data flow or design a method to stop the VI programmatically. By doing so, the VI is at a known state. For example, place a button on the front panel that stops the VI when you click it.

• Click the **Pause** button to pause a running VI. When you click the **Pause** button, LabVIEW highlights on the block diagram the location where you paused execution. Click the **Pause** button again to continue running the VI.

• Select the **Text Settings** pull-down menu to change the font settings for the VI, including size, style, and color.

• Select The **Align Objects** pull-down menu to align objects along axes, including vertical, top edge, left, and so on.

• Select the **Distribute Objects** pull-down menu to space objects evenly, including gaps, compression, and so on.
1. Select Start » Programs » National Instruments » LabVIEW 6.1 » LabVIEW to launch LabVIEW. The LabVIEW dialog box appears.

2. Click the Find Examples button. The dialog box that appears lists and links to all available LabVIEW example VIs.

3. You can browse examples by categories, or you can use a keyword search. Click the Search tab to open the keyword browser.

4. In the "Type a keyword to find" box enter “Signal”

5. A list of related topics will appear in the examples window. Click on signal generation, this will list examples on the right side.

6. Click on any program to see a detailed description of the example. Double-click Signal Generation.vi to launch this example.

This will open the “Signal Generation and Processing.vi” Front Panel.

Examine the VI and run it. Change the frequencies and types of the input signals and notice how the display on the graph changes. Change the Signal Processing Window and Filter options. After you have examined the VI and the different options you can change, stop the VI by pressing the Stop button.

Note You also can open the VI by clicking the Open VI button and navigating to labview\examples\apps\demos.llb\Signal Generation and Processing.vi.
When you create an object on the Front Panel, a terminal will be created on the Block Diagram. These terminals give you access to the Front Panel objects from the Block Diagram code.

Each terminal contains useful information about the Front Panel object it corresponds to. For example, the color and symbols provide the data type. Double-precision, floating point numbers are represented with orange terminals and the letters DBL. Boolean terminals are green with TF lettering.

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

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

The program in this slide takes data from A and B and passes the values to both an Add function and a subtract function. The results are displayed on the appropriate indicators.
Creating a VI – Block Diagram

In addition to Front Panel terminals, the Block diagram contains functions. Each function may have multiple input and output terminals. Wiring to these terminals is an important part of LabVIEW programming.

Once you have some experience programming in LabVIEW, wiring will become easy. At first, you may need some assistance. Here are some tips to get you started:

• The wiring tool is used to wire to the nodes of the functions. When you “aim” with the wiring tool, aim with the end of the wire hanging from the spool. This is where the wire will be placed.

• As you move the wiring tool over functions, watch for the yellow tip strip. This will tell you the name of the terminal you are wiring to.

• As you move the wiring tool over a terminal, it will flash. This will help you identify where the wire will attach.

• For more help with the terminals, right-click on the function and select Visible Items>>Terminals. The function’s picture will be pulled back to reveal the connection terminals. Notice the colors- these match the data types used by the front panel terminals.

• For additional help, select Help>>Show Context Help, or press CTRL+H. This will bring up the context help window. As you move your mouse over the function, this window will show you the function, terminals, and a brief help description. Use this with the other tools to help you as you wire.
In addition to Front Panel terminals, the Block diagram contains functions. Each function may have multiple input and output terminals. Wiring to these terminals is an important part of LabVIEW programming.

Once you have some experience programming in LabVIEW, wiring will become easy. At first, you may need some assistance. Here are some tips to get you started:

• The wiring tool is used to wire to the nodes of the functions. When you “aim” with the wiring tool, aim with the end of the wire hanging from the spool. This is where the wire will be placed.

• As you move the wiring tool over functions, watch for the yellow tip strip. This will tell you the name of the terminal you are wiring to.

• As you move the wiring tool over a terminal, it will flash. This will help you identify where the wire will attach.

• For more help with the terminals, right-click on the function and select Visible Items>> Terminals. The function’s picture will be pulled back to reveal the connection terminals. Notice the colors- these match the data types used by the front panel terminals.

• For additional help, select Help>> Show Context Help, or press CTRL+H. This will bring up the context help window. As you move your mouse over the function, this window will show you the function, terminals, and a brief help description. Use this with the other tools to help you as you wire.
LabVIEW follows a dataflow model for running VIs. A block diagram node executes when all its inputs are available. When a node completes execution, it supplies data to its output terminals and passes the output data to the next node in the dataflow path. Visual Basic, C++, JAVA, and most other text-based programming languages follow a control flow model of program execution. In control flow, the sequential order of program elements determines the execution order of a program.

Consider the block diagram above. It adds two numbers and then subtracts 50.0 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 Subtract function is not valid until the Add function has finished executing and passed the data to the Subtract 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 code to the right, consider which code segment would execute first—the Add, Random Number, or Divide function. You cannot know because inputs to the Add and Divide functions are available at the same time, and the Random Number function has no inputs. In a situation where one code segment must execute before another, and no data dependency exists between the functions, use a Sequence structure to force the order of execution.
Section II – Data Acquisition

- Data acquisition (DAQ) basics
- Connecting Signals
- Simple DAQ application

![Diagram showing sensors, terminal block, cable, DAQ device, and computer connection]

ni.com
You must complete several steps before you can use the Data Acquisition VIs. The devices should be configured for the computers in this class.

1. NI-DAQ software must be installed on the computers with LabVIEW support.
2. You must have installed an E-series DAQ board and configured it using Measurement & Automation Explorer (MAX).

For more information on installing and configuring National Instruments hardware, consult the DAQ Quick Start Guide:
Above is the block diagram for a VI that reads temperature from a Data Acquisition (DAQ) board. The user must specify two parameters: device number and channel. The device number specifies which DAQ device on the computer will be used. In most cases, the device number is one. The channel input tells which channel on the DAQ device will be read. Most DAQ cards can read anywhere from 8-64 analog input channels. In this example, the VI reads a value from device 1, channel 0, and displays it in the Data Point indicator on the front panel.
Data Acquisition Terminology

- **Resolution** - Determines How Many Different Voltage Changes Can Be Measured
  - Larger Resolution → More Precise Representation of Signal
- **Range** - Minimum and Maximum Voltages
  - Smaller range → More Precise Representation of Signal
- **Gain** - Amplifies or Attenuates Signal for Best Fit in Range

**Resolution:** When acquiring data to a computer, an Analog-to-Digital Converter (ADC) takes an analog signal and turns it into a binary number. Therefore, each binary number from the ADC represents a certain voltage level. The ADC returns the highest possible level without going over the actual voltage level of the analog signal. Resolution refers to the number of binary levels the ADC can use to represent a signal. To figure out the number of binary levels available based on the resolution you simply take \(2^{\text{Resolution}}\). Therefore, the higher the resolution, the more levels you will have to represent your signal. For instance, an ADC with 3-bit resolution can measure \(2^3\) or 8 voltage levels, while an ADC with 12-bit resolution can measure \(2^{12}\) or 4096 voltage levels.

**Range:** Unlike the resolution of the ADC, the range of the ADC is selectable. Most DAQ devices offer a range from 0 - +10 or -10 to +10. The range is chosen when you configure your device in NI-DAQ. Keep in mind that the resolution of the ADC will be spread over whatever range you choose. The larger the range, the more spread out your resolution will be, and you will get a worse representation of your signal. Thus it is important to pick your range to properly fit your input signal.
There are many different hardware setups possible when acquiring data. All Data Acquisition systems require some sort of connection terminal that accepts a signal from your transducer and transmits it to the DAQ card. Three such terminal blocks are the BNC-2120, SC-2075, and the SCB-68.

The BNC-2120 is a shielded connector block with signal-labeled BNC connectors for easy connectivity to your DAQ device. It also provides a function generator, quadrature encoder, temperature reference, thermocouple connector, and LED so that you can test the functionality of your hardware.

The SC-2075 provides breadboard area for prototyping and BNC and spring terminal connectivity. The built-in ±15 V or adjustable 0 to 5 V power supply and LED’s make the SC-2075 ideal for academic laboratories.

The SCB-68 is a shielded I/O connector block for rugged, very low-noise signal termination. It includes general-purpose breadboard areas (two) as well as an IC temperature sensor for cold-junction compensation in temperature measurements.
Section III – Loops and Charts

• For Loop
• While Loop
• Charts
• Multiplots
Loops

• **While Loops**
  – Have Iteration Terminal
  – Always Run Once
  – Run According to Continue Terminal

• **For Loops**
  – Have Iteration Terminal
  – Run According to input N

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

**While Loops**

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

**For Loops**

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

1. Select the loop
2. Enclose Code to Repeat

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 While or For Loop boundary encloses the section you selected. Add block diagram objects to the loop by dragging and dropping them inside the loop.
The waveform chart is a special numeric indicator that displays one or more plots. The waveform chart is located on the Controls >> Graph palette. Waveform charts can display single or multiple plots. The following front panel shows an example of a multi-plot waveform chart.

You can change the min and max values of either the x or y axis by double clicking on the value with the labeling tool and typing the new value. Similarly, you can change the label of the axis. You can also right click the plot legend and change the style, shape, and color of the trace that is displayed on the chart.
You can wire a scalar output directly to a waveform chart or display multiple plots. Bundle multiple plots together using the Bundle function located on the **Functions»Cluster** palette. The Bundle function bundles multiple outputs to plot on the waveform chart. The waveform chart terminal changes to match the output of the Bundle function. To add more plots, use the Positioning tool to resize the Bundle function.

The context help contains very good information on how the different ways to wire data into charts.
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.

You can build arrays of numeric, Boolean, path, string, waveform, and cluster data types. 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.

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. For example, $n = 9$ for the nine planets, so the index ranges from $0$ to $8$. Earth is the third planet, so it has an index of $2$.

File I/O operations pass data to and from files. Use the File I/O VIs and functions located on the **Functions»File I/O** palette to handle all aspects of file I/O. In this class we will cover reading and writing spreadsheet files.
Adding an Array to the Front Panel

From the **Controls >> Array and Cluster** subpalette, select the **Array Shell** and drop it on the screen.

To create an array control or indicator as shown, select an array on the **Controls>>Array & 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.
Adding an Array (cont.)

Place data object into shell (e.g. digital control).

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

• Loops accumulate arrays at their boundaries

If you wire an array to a For Loop or While Loop input tunnel, you can read and process every element in that array by enabling auto-indexing. When you auto-index an array output tunnel, the output array receives a new element from every iteration of the loop. The wire from the output tunnel to the array indicator becomes thicker as it changes to an array at the loop border, and the output tunnel contains square brackets representing an array, as shown in the following illustration.

Disable auto-indexing by right-clicking the tunnel and selecting Disable Indexing from the shortcut menu. For example, disable auto-indexing if you need only the last value passed to the tunnel in the previous example, without creating an array.

**Note** Because you can use For Loops to process arrays an element at a time, LabVIEW enables auto-indexing by default for every array you wire to a For Loop. Auto-indexing for While Loops is disabled by default. To enable auto-indexing, right-click a tunnel and select Enable Indexing from the shortcut menu.

If you enable auto-indexing on an array wired to a For Loop input terminal, LabVIEW sets the count terminal to the array size so you do not need to wire the count terminal. If you enable auto-indexing for more than one tunnel or if you wire the count terminal, the count becomes the smaller of the choices. For example, if you wire an array with 10 elements to a For Loop input tunnel and you set the count terminal to 15, the loop executes 10 times.
Creating 2D Arrays

You can use two For Loops, one inside the other, to create a 2D array. The outer For Loop creates the row elements, and the inner For Loop creates the column elements.
File I/O operations pass data 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

In this course we will examine the functions that allow you to create and read a spreadsheet file. It may be important to point out that a “spreadsheet” file is merely an ASCII text file containing data with some sort of row and column delimiters. The most common types of spreadsheet files are tab delimited files, where columns are separated by tab keys and rows by carriage returns, and comma separated value (CSV), where columns are separated by commas and rows by carriage returns. Spreadsheet programs such as Microsoft Excel can read these types of files, but do not save in that format by default. If you want to make data from Excel available to LabVIEW, you will need to save the file as a tab delimited or CSV file.
Use the high-level File I/O VIs located on the top row of the Functions»File I/O palette to perform common I/O operations, such as writing or reading the following types of data:

- Characters to or from text files
- Lines from text files
- 1D or 2D arrays of single-precision numerics to or from spreadsheet text files
- 1D or 2D arrays of single-precision numerics or signed 16-bit integers to or from binary files

High-level File I/O VIs include the following:

- **Write Characters to File**—Writes a character string to a new file or appends it to an existing file. The VI opens or creates the file before writing to it and closes it afterwards.

- **Read Characters From File**—Reads number of characters from a file beginning at start of read offset. The VI opens the file before reading from it and closes it afterwards.

- **Write to Spreadsheet File**—Converts a 2D or 1D array of single-precision numerics to a text string and writes the string to a new file or appends it to an existing file. You also can transpose the data. The VI opens or creates the file before writing to it and closes it afterwards. You can use this VI to create a text file readable by most spreadsheet applications.
The above VI generates a 10x25 array of random numbers, displays them on a waveform graph, and writes them to a spreadsheet file named data.txt on the root directory of the hard drive.
Where Do I Go From Here?

- Example programs (Help» Find Examples…)
- Web resources (ni.com)
  - NI Developer Zone (zone.ni.com)
  - Application Notes
  - Info-labview newsgroup (www.info-labview.org/)
  - Instrument Driver Library (www.ni.com/idnet)

Where do you go from here?

- National Instruments offers a wide range of instructional courseware to expand your knowledge. Please visit ni.com/academic for programs and resources available.
- The LabVIEW Student Edition is available from our website. It includes Learning With LabVIEW, a textbook written by Dr. Bob Bishop from the University of Texas at Austin.
- The web is the best place to turn in order to find information on LabVIEW. Ni.com is designed to be the one stop resource to find information.
  - The NI Developer Zone (“NIDZ”) is a place for developers to meet, discuss design issues, and post content.
  - Application Notes can be downloaded from ni.com/support on a variety of topics.
  - Info-labview is a newsgroup maintained by a third party.
  - There is an exhaustive library of LabVIEW instrument drivers available for download from NIDZ.