NATIONAL INSTRUMENTS™

Introduction to LabVIEW and Computer-Based Measurements

Contents

Hardware Configuration

Goals

- Connect your NI CompactDAQ system to the PC
- Use Measurement & Automation Explorer to configure your hardware and verify signal connections
- Create a simulated NI-DAQmx device (Optional)

Part A Hardware Connection and Setup

Estimated time: 5 minutes

The hardware for this seminar includes many measurement modules to give you the best possible understanding of the value of modular data acquisition platforms. Today, you can explore analog voltage, digital input, accelerometer, strain and temperature measurement as well as digital output and voltage generation. For more information about the modules you see in the chassis or to learn more about NI CompactDAQ, check out ni.com/compactdaq.

1. Before beginning any of the following exercises, ensure that your chassis is connected to your PC via the supplied USB cable and both the PC and the NI CompactDAQ chassis have power. Both the green Power LED and amber Ready LED should be lit before you begin configuring your chassis in software.

2. Take a moment to go over the connections in the demo box. Make sure that all modules are securely seated in the chassis. Additionally, if any wires are loose or something appears missing or broken, alert your instructor.

3. Once you have confirmed everything is connected and powered on, proceed to Part B.

Part B Software Configuration

Estimated time: 20 minutes

Similar to the Windows Device Manager, which manages all peripherals connected to a Windows PC, MAX manages all NI hardware and software. This application is installed with most NI software packages. In this exercise, you will look at the most used features of MAX including **Software** and **Devices and Interfaces**.

Note: You can access two systems in MAX: the locally installed hardware and software, which is listed under **My System**, and **Remote Systems**, which contains all the remote targets you have detected on your network such as NI CompactRIO hardware or a real-time PXI system. Today, you will only be working with the local PC and USB chassis, so everything will be under **My System**.

1. Once your hardware is connected and powered on, open MAX by double clicking the icon on the desktop or navigating to **Start » All Programs » National Instruments » Measurement & Automation Explorer**.

2. After MAX launches, expand **My System » Devices and Interfaces**. You should be able to see any hardware attached to your local machine:

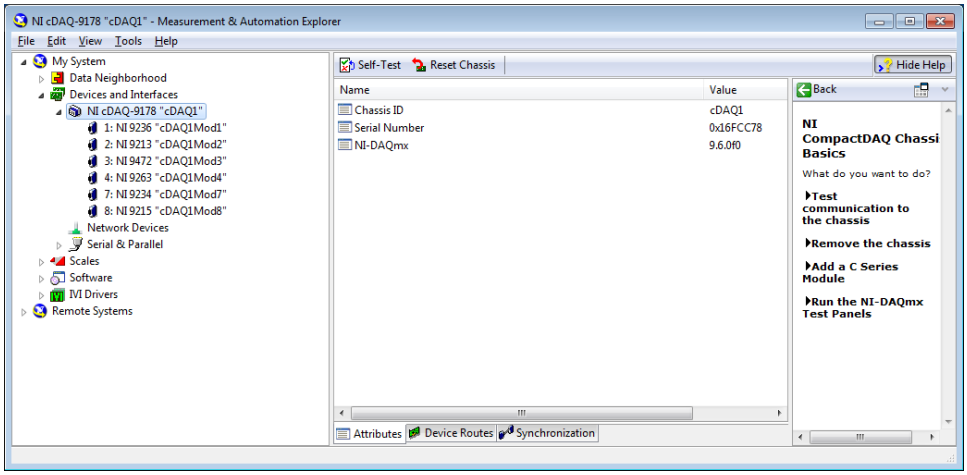


Figure 1. Expand the chassis drop-down menu to show attached modules.

Within MAX, you can ensure that your hardware is properly connected, check external connections, rename your devices, and even simulate devices for developing code without attached hardware.

3. In addition to hardware, you can see all the software installed on the machine by expanding **My System » Software**. Here you can check the driver and software versions:

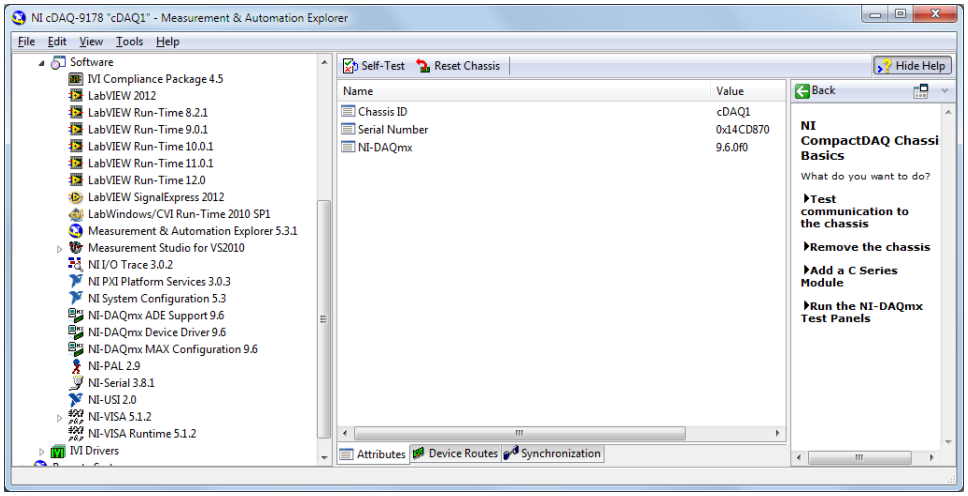


Figure 2. Make sure you have the right software and drivers installed by expanding the Software section in MAX.

Underneath the software section, you should notice that both LabVIEW and NI-DAQmx are installed – NI-DAQmx is the driver that lets your PC communicate with your hardware and LabVIEW is the graphical programming environment that we will use to program during today's exercises.

4. Now that you have navigated through and understand the basics of MAX, you can test the connectivity of the NI CompactDAQ chassis. **Right-click** the chassis under the Devices and Interfaces section and select **Self-Test**.

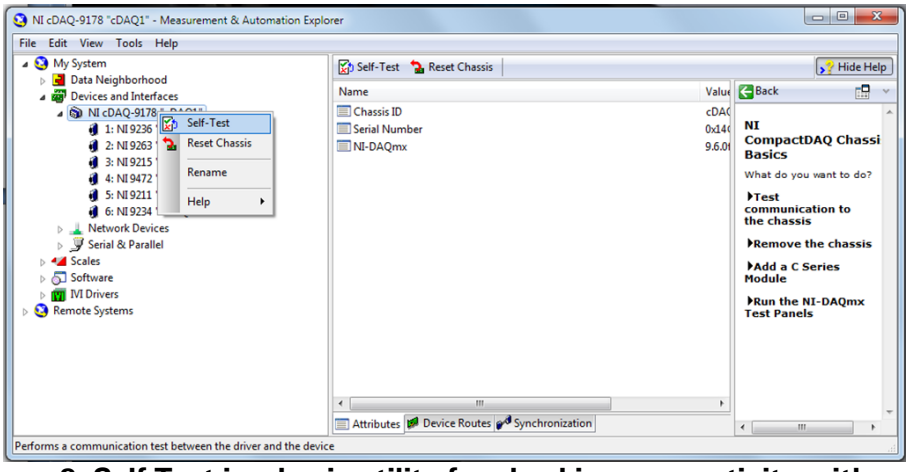
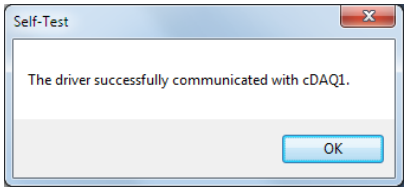


Figure 3. Self Test is a basic utility for checking connectivity with your hardware.

If your chassis is properly connected and able to communicate with your PC, then you will see the following dialog box:



Self-testing the chassis is always a good first step after installing new hardware or software.

5. To check the signal connections to the NI CompactDAQ chassis, you will use the NI-DAQmx Test Panels. As an example, we will check the signal connectivity of the NI 9213 thermocouple module.

   a. Right click on the NI 9213 and select **Test Panels…**:

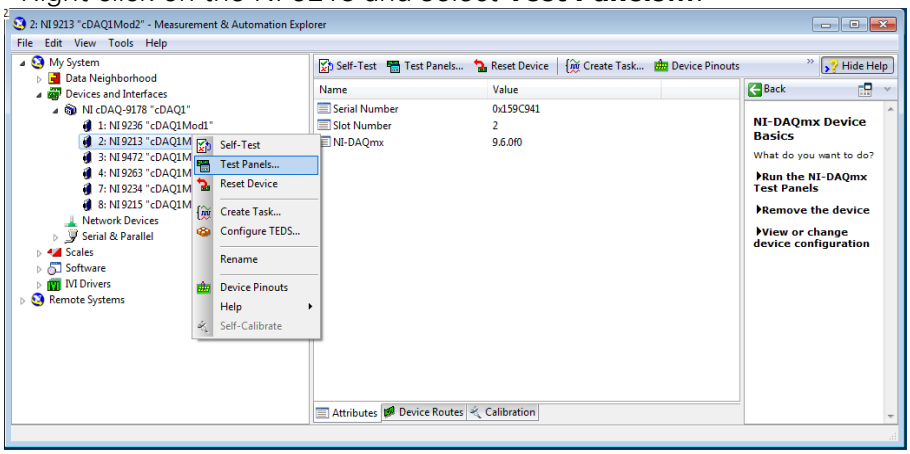


Figure 4. Test Panels let you check your signal connections without writing any code.

b. Selecting Test Panels brings up a simple utility to check the signals on each of the channels of the module. When you press **Start**, you should see the following screen for the NI 9213 test panel:

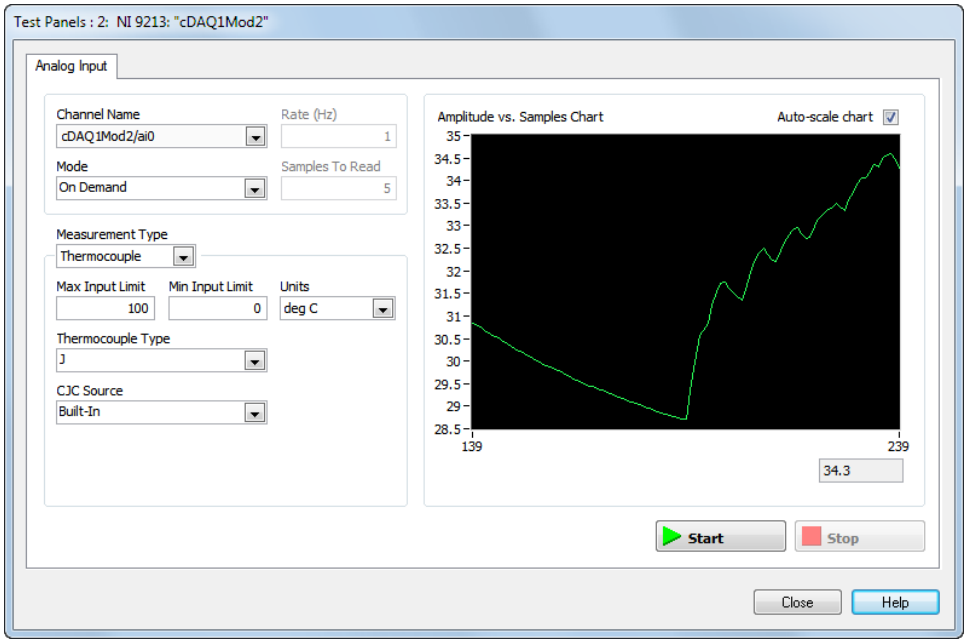


Figure 5. The NI 9213 test panel lets you read the signal on each of the channels of the device. Scaling and CJC settings are built in to simplify testing.

c. Hold the tip of the thermocouple and notice that the temperature increases.

d. Take a moment to practice with the test panels for the other modules in the chassis. Notice that test panels are slightly different, each providing the inputs for measurements supported by the module.

e. When finished, close any open Test Panel window by selecting **Close**.

6. For many applications, simply using the default assigned names is not sufficient. Within MAX, you can very easily rename your modules to match the module type or measurement.

a. To rename the module, right-click on the module and select **Rename**.

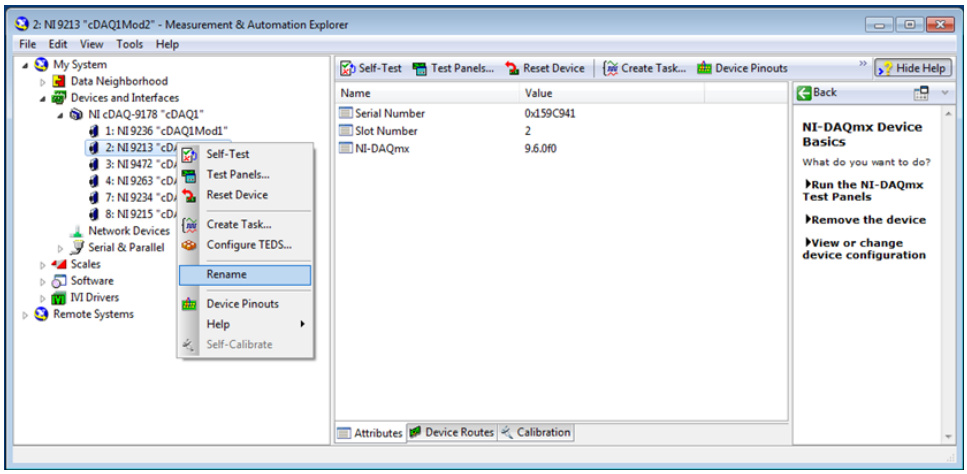
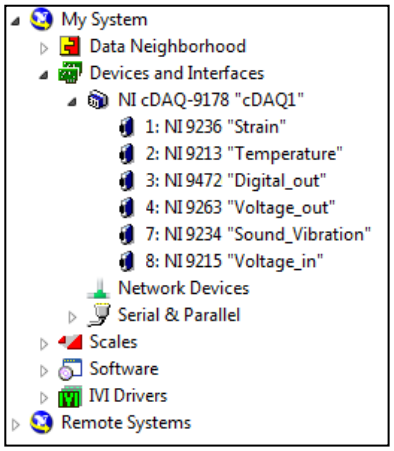


Figure 6. Select Rename **to add new descriptive names.**

b. Rename each of the channel types to descriptive names.

c. Use the following descriptive names to identify your modules



Part C Creating a Simulated Device (Optional)

Estimated time: 10 minutes

When developing your data acquisition application, oftentimes you may not have immediate access to the hardware in the initial stages of writing the software. For situations like this, you can create simulated devices in MAX.

1. To create a simulated device, right-click on **Devices and Interfaces** and select **Create New…**.

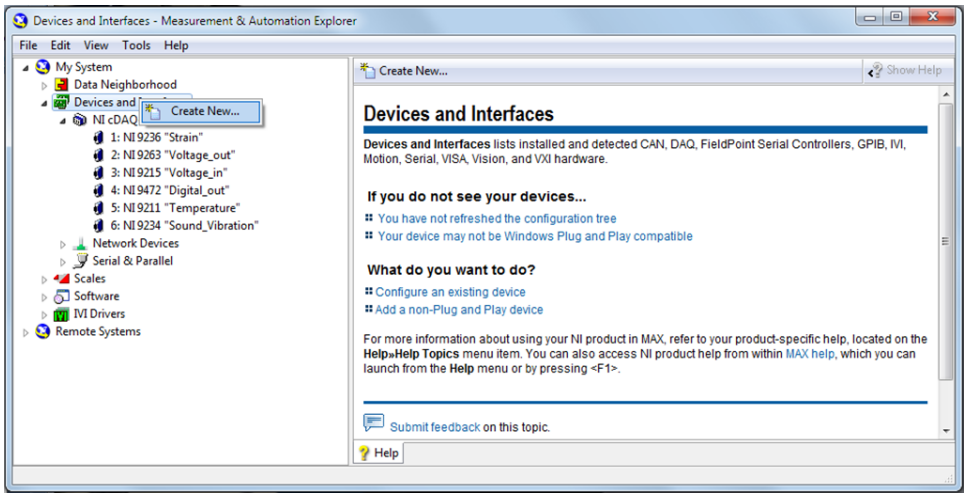


Figure 1. Creating a simulated device enables you to experiment with a device without having the actual hardware on-hand.

2. When the Create New… screen launches, select **Simulated NI-DAQmx Device or Modular Instrument** and press **Finish**.

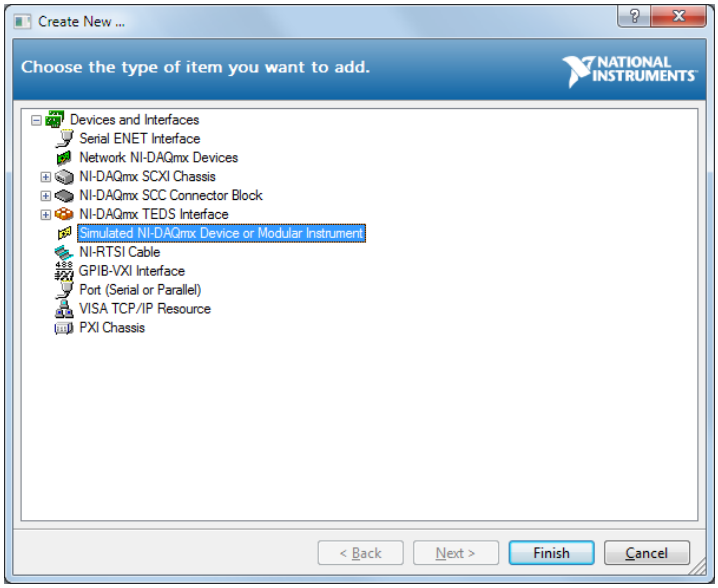


Figure 2. Within MAX, you can simulate any NI-DAQmx device and many modular instruments.

3. For this exercise, select **NI cDAQ-9174**. This is a 4-slot version of the NI CompactDAQ chassis that you are using for exercises in this booklet. Press **OK** to add the simulated device.

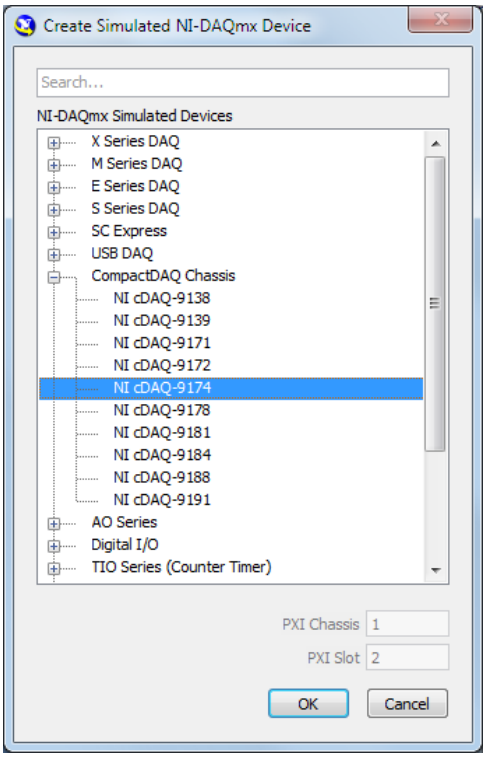


Figure 3. The NI cDAQ-9174 is just one of many options for simulated devices.

4. The new simulated chassis will show up under Devices and Interfaces just like the real devices connected to the system but will be colored yellow. Notice that modules are not included. To add modules to the chassis, select **Configure Simulated cDAQ Chassis…**

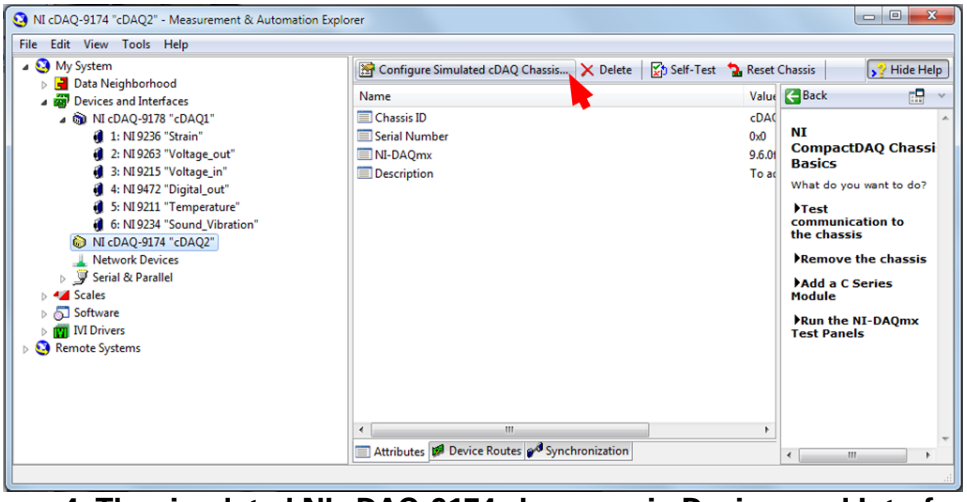


Figure 4. The simulated NI cDAQ-9174 shows up in Devices and Interfaces just like a real device, but the emblem is yellow.

5. For Slot 1, select the **NI 9201** module – this is a 12 bit voltage input module that is not included in your chassis. Press **OK** to add the module to the chassis.

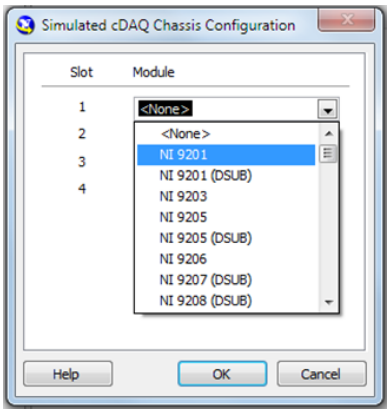


Figure 5. The NI 9201 is a basic voltage input module, but you can configure your simulated chassis for any module that the NI cDAQ-9174 supports.

6. Once the module is added to the simulated chassis, you can perform actions just like on a real device. To illustrate this, right click on the NI 9201 and select **Test Panels…**

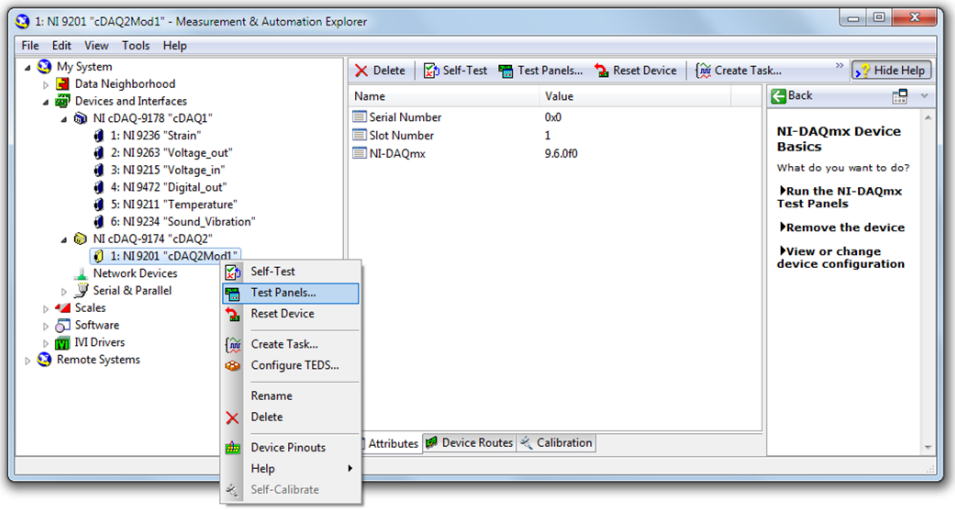


Figure 6. Select Test Panels… **to take measurements from your simulated device.**

7. When the simulated device test panel appears, press **Start** to begin visualizing the simulated data. Notice that the data is a simple sine wave. It is important to note that this data set is built into the NI-DAQmx driver and does not actually constitute real data; however, you can interact with it exactly the same way as interacting with real data. For example, you could write code to analyze the maximum and minimum values and write those values to a spreadsheet file.

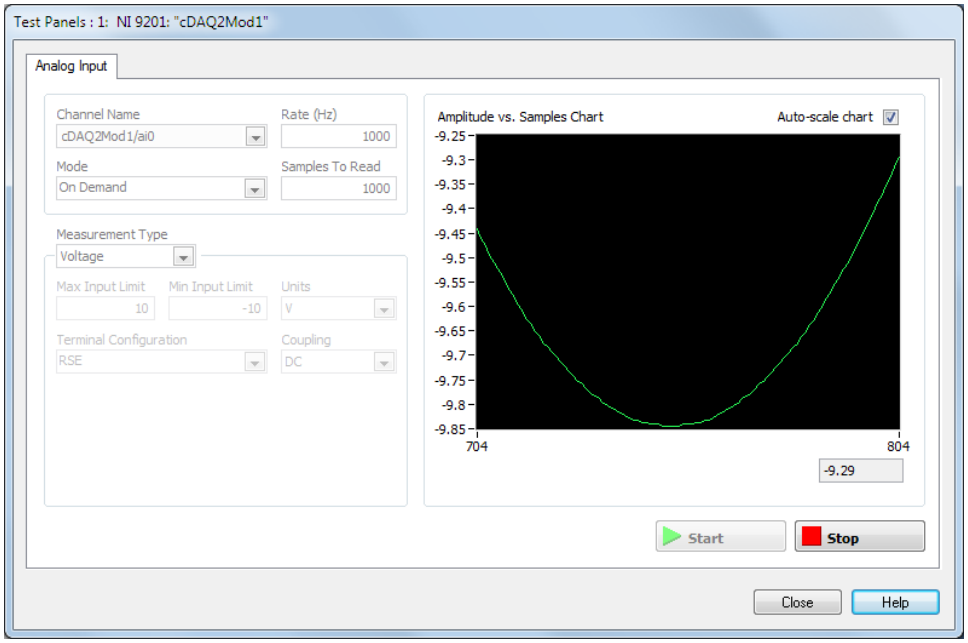


Figure 7. The data displayed in the simulated device test panel is a simple sine wave that is included in the NI-DAQmx driver.

8. Close any open Test Panel windows by selecting **Close**.

9. Delete the simulated device by right-clicking on the NI cDAQ-9174 and selecting **Delete** from the context menu.

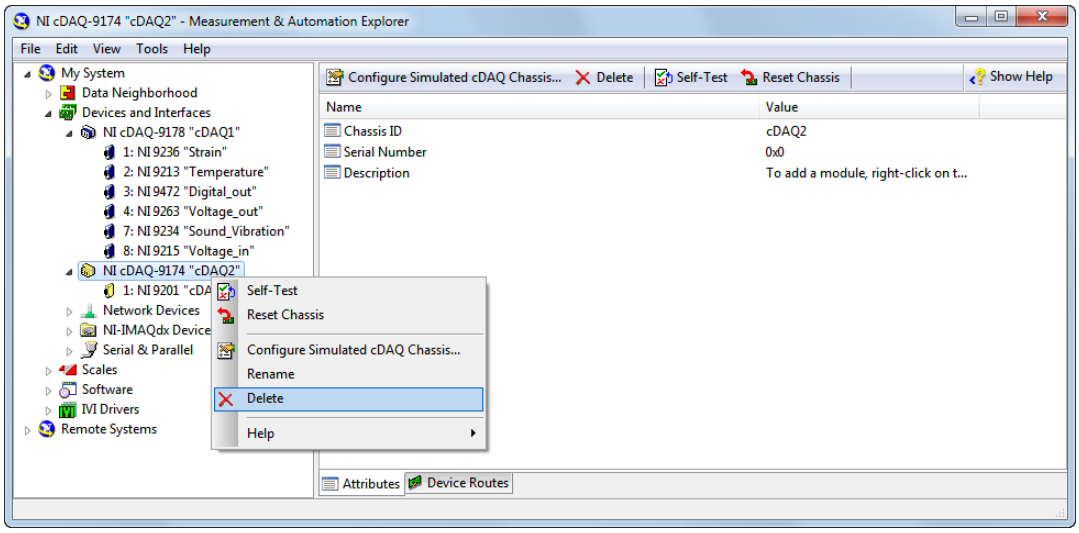


Figure 8. Remove the simulated device from the system.

<End of Exercise>

Explore the Example Finder

Goals

- Become familiar with the NI-DAQmx examples
- Run a pre-built example program to acquire your first measurement

Part A Open LabVIEW and Explore the Example Finder

Estimated time: 10 minutes

Installed with NI-DAQmx, the NI-DAQmx examples provide you with a proven starting point for creating data acquisition applications. By utilizing an example program, you can eliminate several sources of errors and save time by utilizing existing code.

1. To access the examples, open LabVIEW by navigating to **Start » All Programs » National Instruments » LabVIEW <Year>**. Upon launching, you will see the following Getting Started screen:

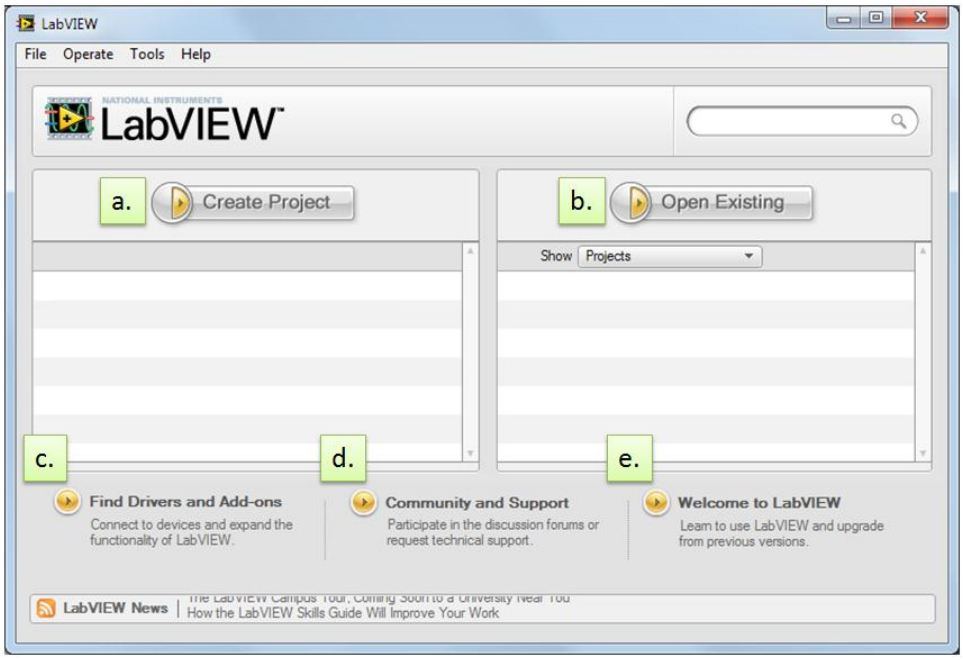


Figure 1. The LabVIEW start-up screen provides several paths for learning, support and development.

2. The opening screen provides paths for developing code, recalling previously opened programs and finding support or help resources.

    a. *Create Project* – Selecting Create Project will launch a screen with options to create new code from existing templates and sample projects. Code architectures like the State Machine and Actor Framework are included in the templates and you can quickly get up and running by creating them from this dialog box. The sample projects are ready-to-run code built using best practices for scalable development. Using the templates and sample projects, you should never have to start a LabVIEW application from scratch.

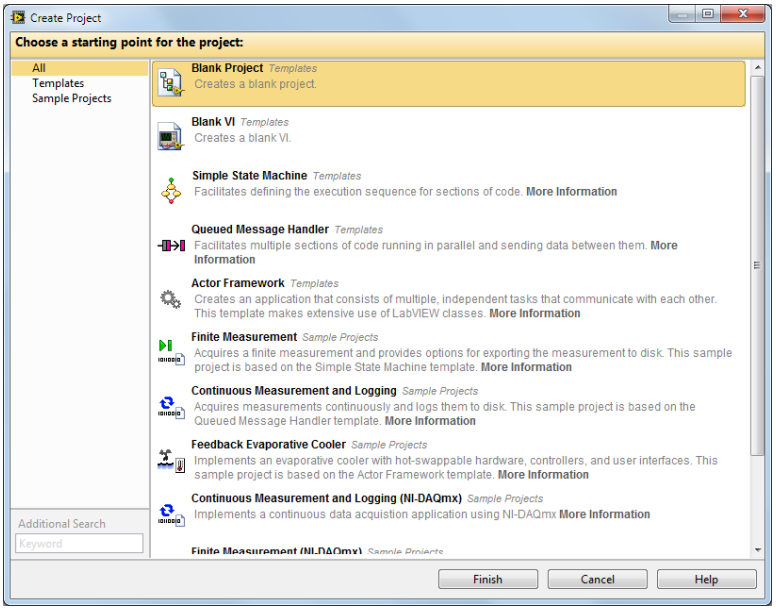


Figure 2. Templates and Sample Projects can help you create scalable applications faster with proven code and techniques.

b. *Open Existing* – As you work on programs today, you can quickly recall projects that you have worked on within the Open Existing section or press the Open Existing button to browse for your code.

c. *Find Drivers and Add-ons* – For most hardware connected to your PC, you will need to use a hardware driver. Today, you will be using NI CompactDAQ, which utilizes the NI-DAQmx driver. This same driver works for hundreds of data acquisition devices, enabling you to reuse code from device to device and application to application.

d. *Community and Support* – For those times when you get stuck, NI has a vibrant community of LabVIEW developers and data acquisition experts. The forums are a free support resource that is monitored by both NI Applications Engineers and NI enthusiasts to ensure that you will get the support you need. For more immediate support, you can call or email NI Applications Engineers directly as one of the many benefits of the LabVIEW Standard Service Program (SSP).

e. *Welcome to LabVIEW* – After today, you will be familiar with using the LabVIEW environment. To move beyond simple programming tasks and access more in-depth proficiency training, refer to the help and training resources in the Welcome to LabVIEW link.

3. Today, we will be using several examples as starting points for our code. To open the NI-DAQmx examples directory, select **Help » Find Examples** from the LabVIEW Getting Started screen.

4. Once the NI Example Finder launches, navigate to **Hardware Input and Output » DAQmx**.

5. Navigate the DAQmx directory and read some of the capabilities of the examples. Each example has a descriptive name, but you can find out more capabilities of the program by reading the descriptions in the Information section.

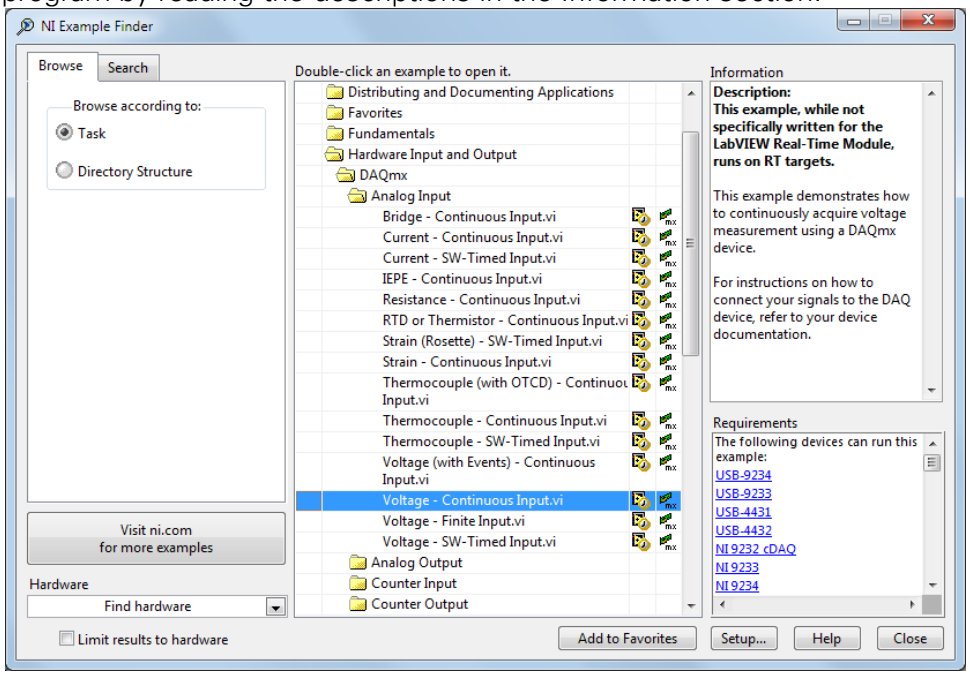


Figure 3. The NI Example Finder is a useful tool for navigating the installed NI examples.

Each section of the directory includes examples for performing certain tasks. For example, analog input includes examples for measuring simple voltage, current, strain, and temperature, whereas the analog output section includes examples for controlling either voltage or current.

Part B Run a Pre-Built Example

Estimated time: 10 minutes

Now that you are familiar with the different measurement types provided by the NI-DAQmx examples, you can open and run several that use some of the hardware in your hands-on kit. We will start with a simple voltage measurement.

1. In the NI Example Finder, navigate to **Hardware Input and Output » DAQmx » Analog Input** and double click **Voltage – Continuous Input.vi**. This will launch a VI to continuously measure a voltage channel.

2. When the VI launches, the front panel will be displayed. This is the user interface and includes controls and indicators for interacting with your code. In this particular example, notice the controls for selecting the channel, voltage input ranges, timing parameters and logging location. Additionally, the front panel includes a graph indicator for instantly viewing the data on the voltage channels.

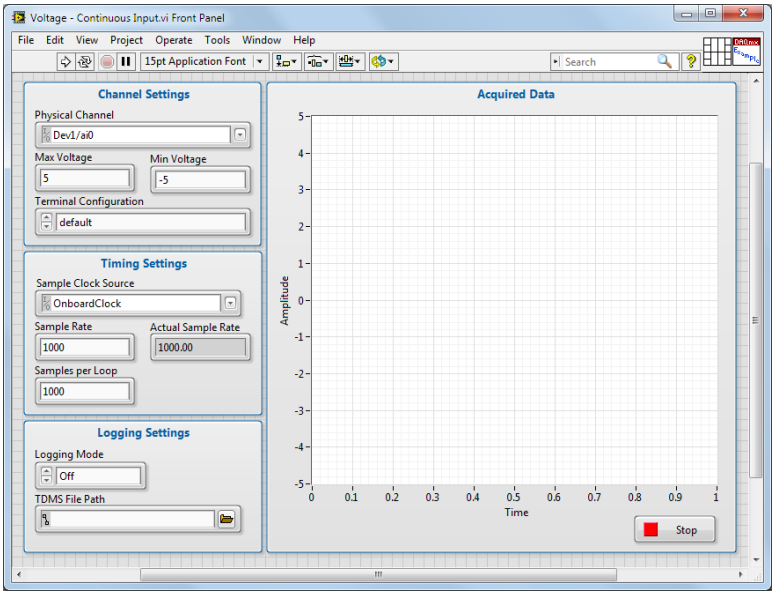


Figure 4. The front panel is the user interface and contains controls and indicators for interacting with your LabVIEW code.

3. To view the code behind the scenes of this front panel, select **Window » Show Block Diagram** or use the keyboard shortcut **<Ctrl+E>**.

4. Expand the block diagram window so that you can see more of the source code.

   You will see a series of VIs used to program your DAQ device to acquire a voltage signal. LabVIEW uses dataflow to pass information along wires to sequential VIs. This programming paradigm enables you to better program like you think.

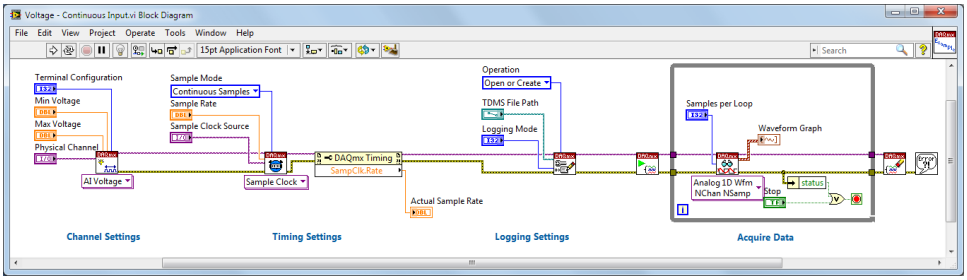


Figure 5. LabVIEW uses a graphical programming language and passes data through the code via wires that connect nodes.

5. The block diagram is organized into the following sections:

   1. **Channel Settings** – In this section, the *DAQmx Create Channel VI* configures the channel on the DAQ device that you intend to use. In this example, you can configure the range of the device and terminal configuration (single ended versus differential).

   2. **Timing Settings** – In the Timing Settings section, the *DAQmx Timing VI*

is used to configure the sample clock on the DAQ device. Using this VI, you can configure the sample rate, the sample clock source, and the sampling mode you intend you use. In this example, the default settings use the on-board clock to sample continuously at 1000 Hz.

3. **Logging Settings** – The *DAQmx Configure Logging VI* is used to set the location of the logged data file as well as the logging mode.

4. **Acquire Data** – In this section, the *DAQmx Read VI* is called inside a While Loop, allowing this code to continuously acquire data from the DAQ device until the Stop button is pressed.

Most data acquisition applications will follow this flow –

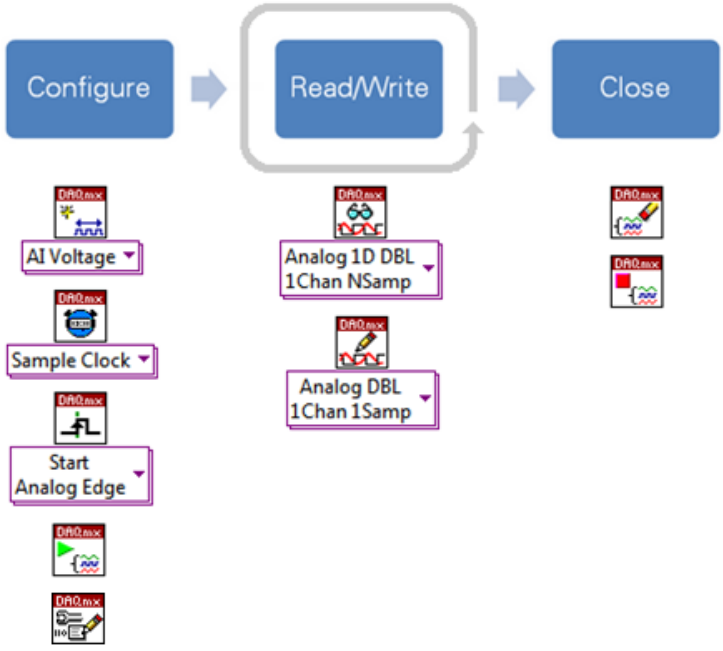


Figure 6. The most common VIs for each section are shown above.

6. Compare the example program block diagram to the VIs shown in Figure 6. If you open other DAQmx example programs, notice that most follow this exact flow.

7. For more information about the elements on your block diagram and front panel, press **<Ctrl+H>** to bring up the Context Help window. By hovering over elements in your code, you can see brief descriptions and the inputs and outputs of each VI. Required input labels are shown as bold in the Context Help. For more detailed descriptions, you can press the **Detailed help** link. Try hovering over a few elements on your block diagram.

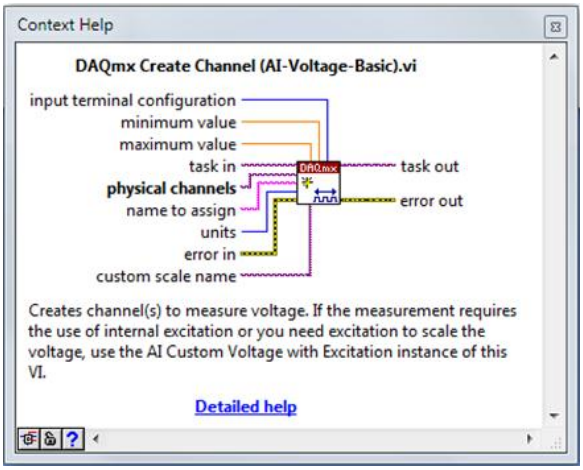


Figure 7. Context Help can provide a quick description of the elements in your front panel or block diagram.

8. Now that you are familiar with the parts of a basic program, you are ready to acquire data using this example. To do this, switch back to the front panel by selecting **Window » Show Front Panel** or by pressing **<Ctrl+E>**.

9. Change the Physical Channel to **Voltage_in/ai0**.

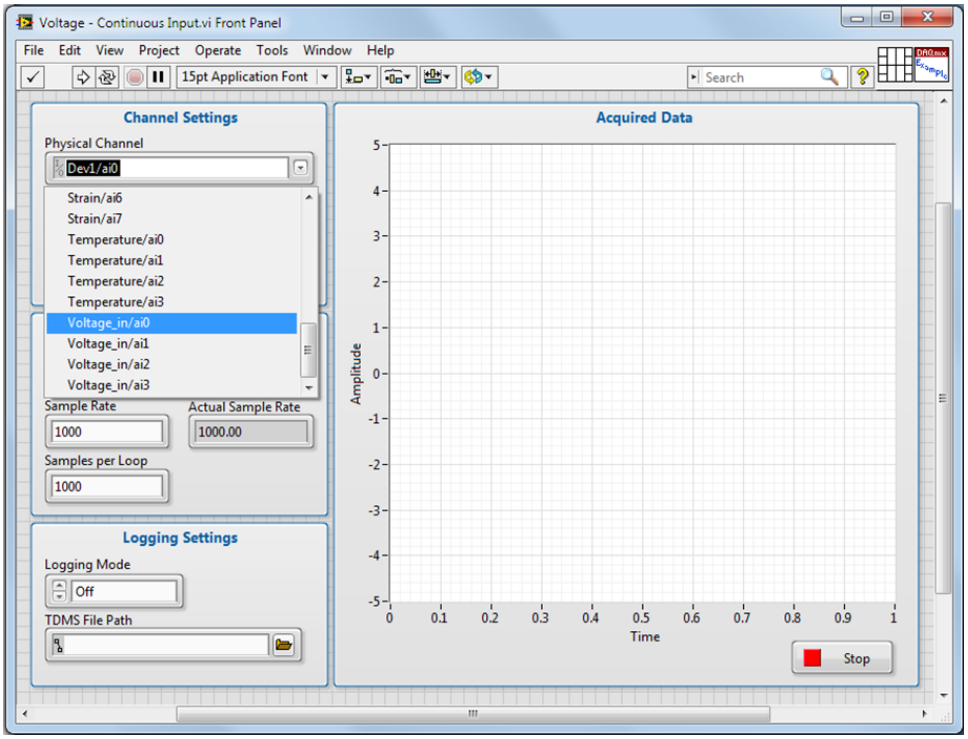


Figure 8. Use the controls on the front panel to select the channel, timing settings and logging settings.

10. Change the **Sample Rate** control to **10000** and press the Run arrow ( ) to begin acquiring data.

Because your signal is connected to the light sensor in the box, you will see the output of the light sensor. This signal will have some noise from sources in the room but you should be able to see the effects of placing your hand over the sensor. You are sampling the signal along with the noise in the room at 10 kHz - what could be some sources of noise in the room around you? What could you do to get a better idea of what is happening to the signal of interest?

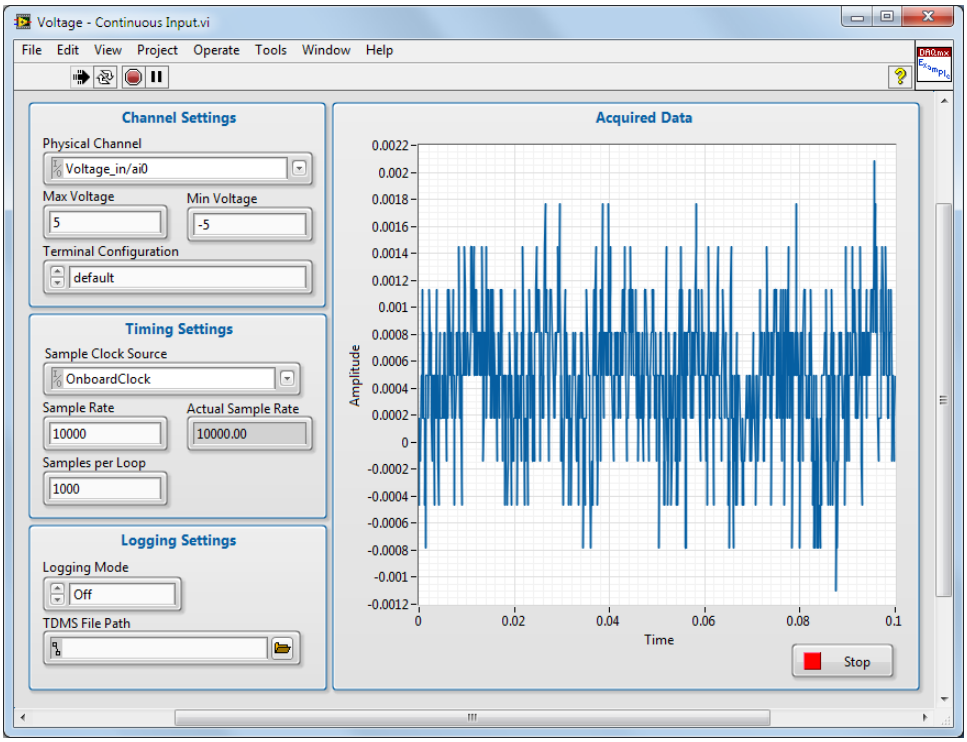


Figure 9. With this program, you are sampling a channel at 10 kHz. Throughout this seminar, you will modify and customize examples to perform many tasks.

11. Press **Stop** to stop acquiring data.

12. Exit the program without saving.

Part C Experiment With Other Example Programs (Optional)

Estimated time: 10 minutes

Try exploring other example programs. Examples that might be useful to look include

1. Analog Input / Thermocouple – Continuous Input.vi
   Note:
   - Be sure to select **Temperature/ai0** from the Physical Channel control.

2. Analog Input / Strain – Continuous Input.vi
   Note:
   - Be sure to select **Strain/ai0** from the Physical Channel control.
   - Be sure to select **Quarter Bridge I** from the Strain Configuration control.
   - Be sure to use **3.3** as the Voltage Excitation Value.

3. Digital Output / Digital – Continuous Output.vi
   Note:
   - Be sure to select **Digital_out/port0/line0:7** from the Line(s) control.
   - Be sure to select **OnboardClock** from the Sample Clock Source control.

4. Analog Output / Voltage (non-regeneration) – Continuous Output.vi
   Note:
   - Be sure to select **Voltage_out/ao1** from the Physical Channel(s) control.
   - Be sure to select **1.00** from the Waveform Settings Frequency control.
   - Be sure to select **3.00** from the Waveform Settings Amplitude control.
   - Be sure to change the *Fan Speed Control* hardware switch on the *Sound and Vibration Signal Simulator* in your demo box to **BNC**.

   - If the fan continues operating after stopping the example program, navigate to MAX and reset the NI 9263 by right-clicking on the **Voltage_out** module and choosing Reset Device.

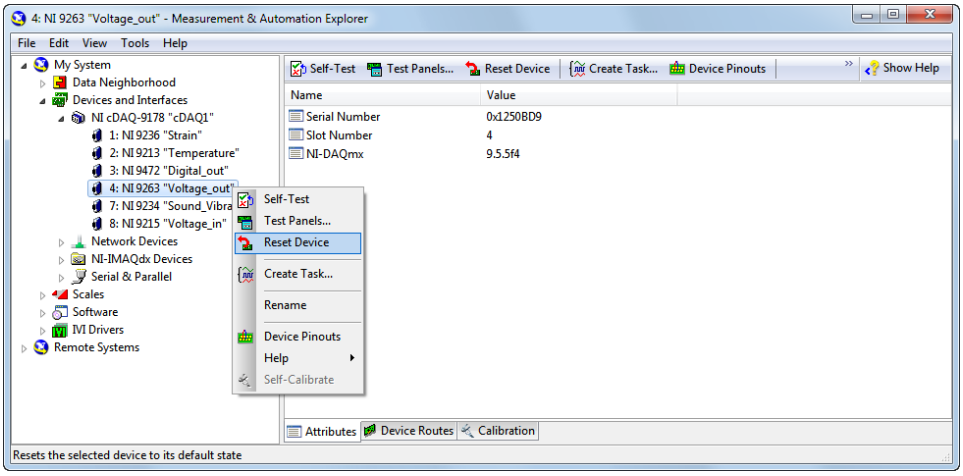


Figure 10. You can reset any hardware device to its default state in MAX.

<End of Exercise>

Temperature Measurement and Logging

Goals

- The first goal is to quickly acquire a set of temperature data using the DAQ Assistant
- The second goal is to perform analysis on that data and output a digital warning signal if the temperature goes above an adjustable warning level.

Part A

Part A gives you a chance to see what you'll complete in this exercise's final application. You'll also explore important elements of the LabVIEW environment.

1. If you have not already done so, launch LabVIEW. Click the **LabVIEW** application after navigating to **Start » All Programs » National Instruments » LabVIEW <Year>**. Once you launch LabVIEW, the Getting Started window appears:

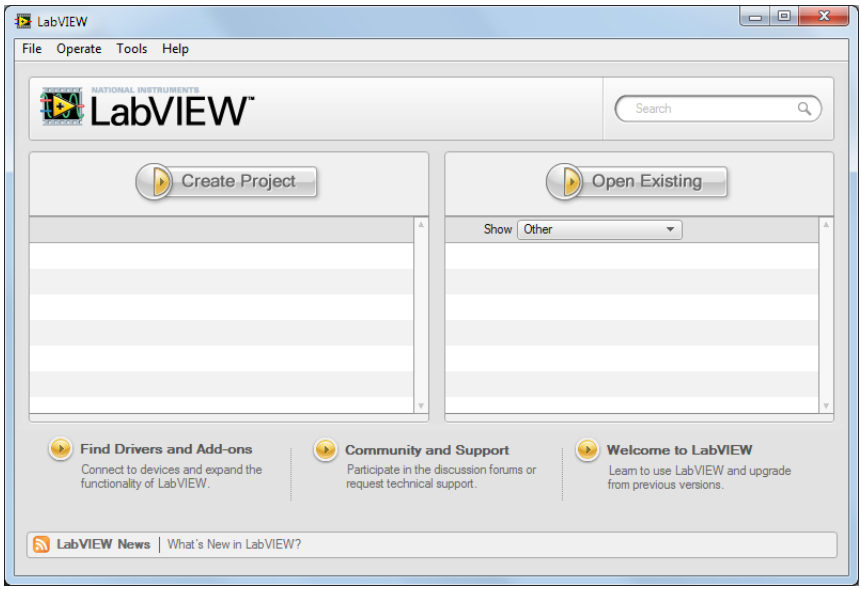


Figure 1. The LabVIEW start-up screen provides several paths for learning, support and development.

The LabVIEW Getting Started window appears each time you launch LabVIEW to assist you in creating new applications or opening existing applications. Additionally you can use links on the Getting Started window to find local and online help resources or open example programs to aid in application design.

2. Select the **Open Existing** button to open an existing project.
3. Navigate to **C:\Seminars\LV HO\Exercises\3 - Temperature** and select the **Temperature.lvproj** project. Once opened, the Project Explorer window will appear and look like this:

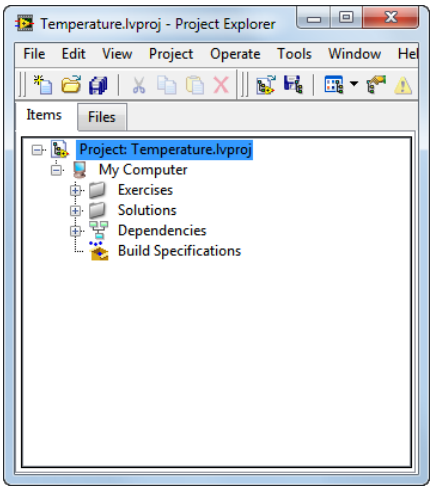


Figure 2. The LabVIEW Project is a great way to organize all the dependencies in an application.

The Project Explorer provides a central location for you to include the different elements of an application including LabVIEW code and other files like Microsoft Word and Excel documents. You can include any file in a LabVIEW application. You can create folders and sub-folders to organize the files in an application. Here, a few folders have been created as part of the example.

4. Expand the **Solutions** folder in the Project Explorer and open the *5 - Write to File (Solution).vi* by double-clicking on it or right-clicking and selecting **Open**.
   Every LabVIEW application is made of a front panel and a block diagram. The front panel is the user interface, whereas the block diagram contains the code that controls the functionality of your application. You can toggle between the two windows by selecting **Window » Show Block Diagram** or **Window » Show Front Panel** to see the other window. You can also switch between the windows by pressing **<Ctrl+E>** on the keyboard or clicking either window if both are present on your monitor.

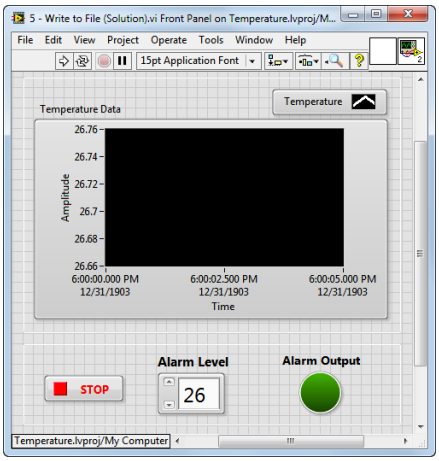
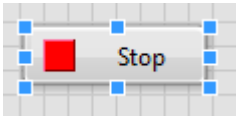


Figure 3. The application's front panel is its user interface.

Hover the cursor over the different objects on the front panel. Notice that your cursor turns into a pointer finger when above the Stop button, and turns into a text editor when

above a text field. By default, LabVIEW's Automatic Tool Selection will change the cursor depending on what operations are possible. Also notice that as you move over any object, resizing boxes appear on its edges. Try resizing a few objects' sizes.



5. Notice the menu bar at the top of the window. We will discuss many of its basic items in future pages and exercises. For now, the most important button to review is the **Run** button, found on the left edge of the menu bar.

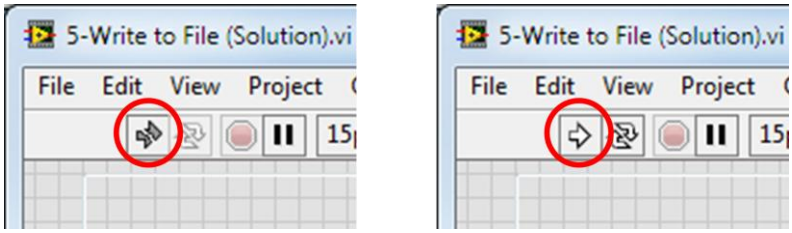


Figure 4. A broken run arrow (left) indicates that there is an error compiling the application.

You must press the **Run** button to begin any LabVIEW application, and a broken run arrow tells you that there are some unresolved errors in the code. Since LabVIEW is continually compiling code throughout development, you can press the broken **Run** button at any time and a list of current errors will appear.

6. Make sure that your NI CompactDAQ chassis is connected to your PC with a USB cable and that the modules are plugged into the chassis. Now press the **Run** button in the LabVIEW application and watch as the application begins to record temperature data from the module plugged into the first slot of the CompactDAQ chassis. Contact the instructor if your application isn't running as described.

7. Hold the end of the thermocouple and watch the values on the graph rise and fall accordingly. Change the "Alarm Level" control to different values and hold the thermocouple so that it rises above and below the value you've entered on the front panel.

As temperature rises and falls around the Alarm Level, look at the **NI 9472 module** in the CompactDAQ chassis. One digital output line on this module has been programmed to drive a 5 V signal whenever temperature is greater than the value of Alarm Level. The module's LEDs indicate the status of each digital line. These lines could be connected to other hardware, like a light or buzzer, or other 5 V device

8. Press the **Stop** button on the front panel once you are ready to move on.

9. Navigate to the block diagram by selecting **Window » Show Block Diagram**.

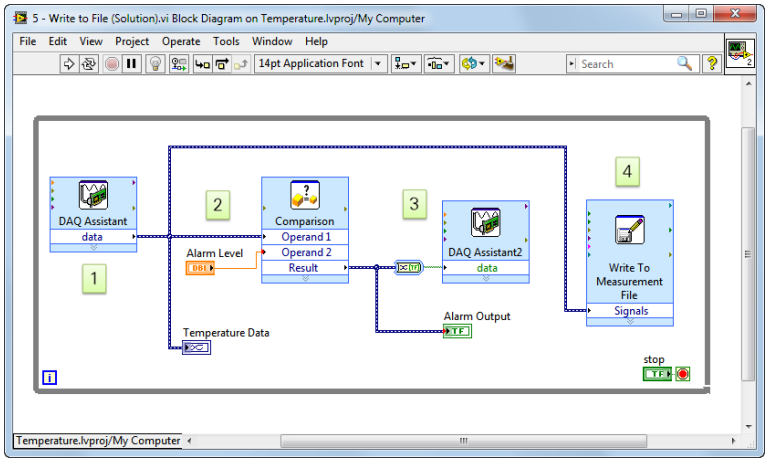


Figure 5. The block diagram is the source code of an application.

LabVIEW's graphical programming uses dataflow to control the flow of an application. In this case our application does the following:

1. Acquires temperature data with the DAQ Assistant and displays it on a chart.
2. Compares acquired data with Alarm Level.
3. Outputs 0 V or 5 V to the digital output module based on the comparison in #2.
4. Writes acquired data to file.

10. Distribute the front panel and block diagram windows so that both are visible in your monitor. Navigate to **Window » Tile Left and Right** to tile the front panel and block diagram on your monitor, or press **<Ctrl+T>**.

Notice that for every object on the front panel, there is a terminal with the same name on the block diagram. The functions and wires on the block diagram connect the inputs ("controls") and outputs ("indicators") on the front panel. As you add objects to the front panel in future exercises you'll see that terminals are automatically created on the block diagram.

Additional Steps

11. The LabVIEW help system is a great way to learn about LabVIEW and answer your programming questions. Press **<F1>** on the keyboard to start the help system. More assistance can be found from the **LabVIEW » Help** menu.

12. Within the LabVIEW help system, expand **Fundamentals » LabVIEW Environment** and explore the information available here. Feel free to explore and get a feel for the depth of content and how it is organized.

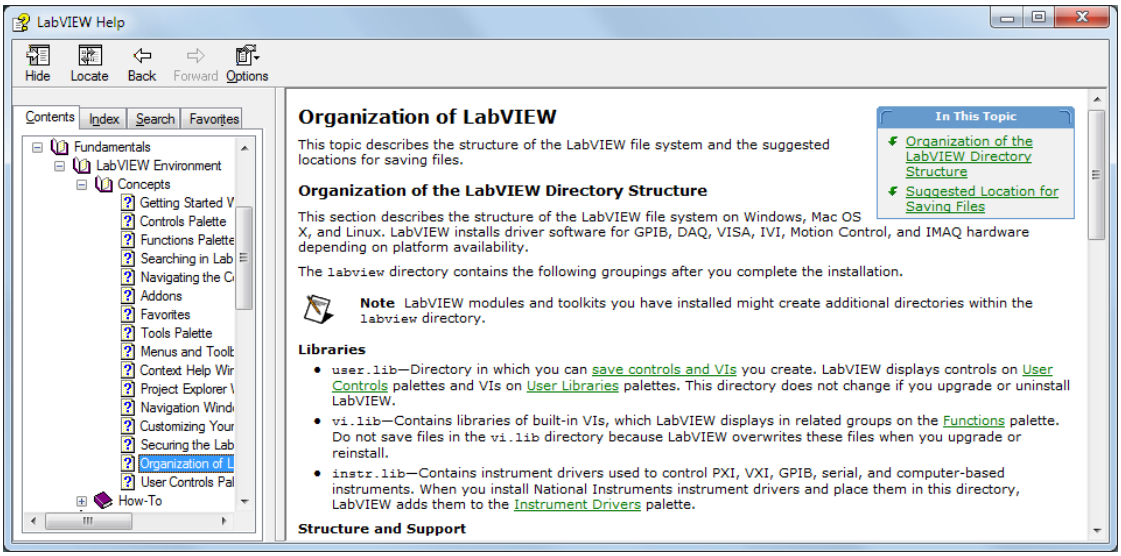


Figure 6. The LabVIEW help is an exhaustive resource for building applications.

13. Take a few minutes to explore other topics in the help system.
14. Click the **Search** tab and try searching on analysis functions for features you might need in your applications.

Part B

This exercise will review the LabVIEW environment basics you have learned so far. You will create an application that simulates a signal inside of LabVIEW and display that signal to a chart.

You should still have the *Temperature* project open and you will work out of this for the next exercises.

1. Open a blank VI from the *Temperature* Project Explorer by right-clicking the Exercises folder and selecting **New » VI**.

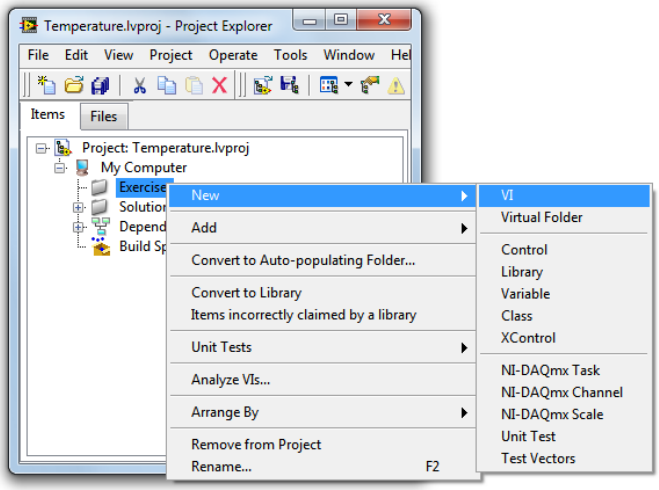


Figure 1. You can create a new VI from directly within a Project.

2. Press **<Ctrl+T>** to tile the Block Diagram and Front Panel.

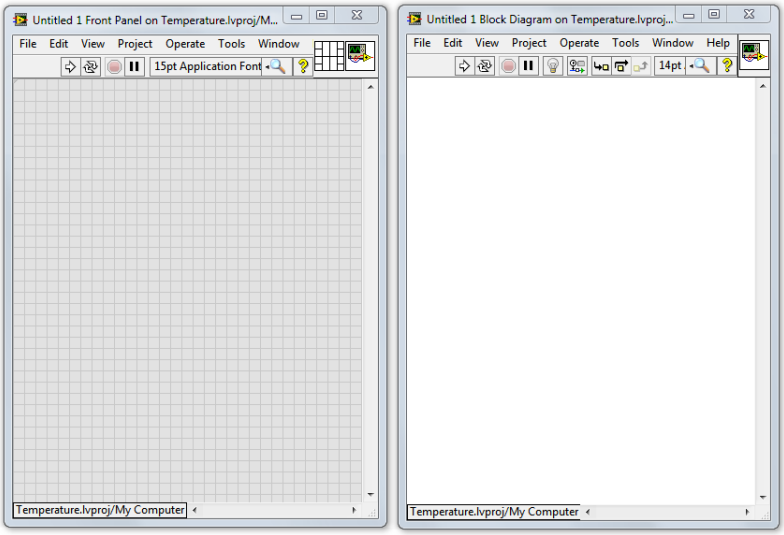


Figure 2. It is often helpful to view user interface and source code side-by-side.

3. Save this VI in the *\LV HO\Exercises\3 – Temperature\Exercises* folder by selecting **File » Save** and naming it **2 - Simulate Signal to Graph.vi**.
4. Right-click any empty space on the block diagram to bring up the Functions palette, and then navigate to **Programming » Structures » While Loop**. Left-click the While Loop and it will be automatically placed on your cursor. Keep in mind your palette may not look exactly like the one pictured below.

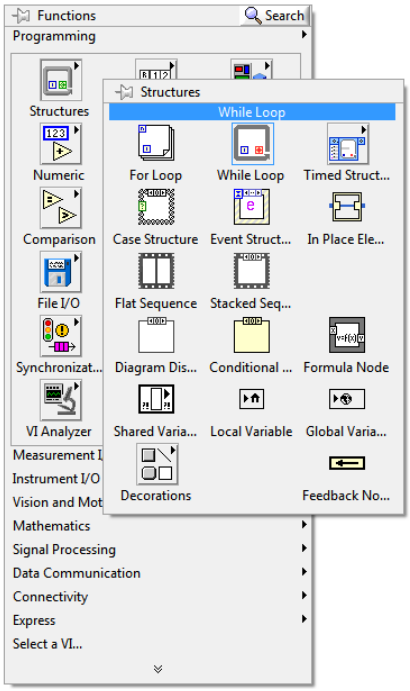


Figure 3. The Functions palette is a hierarchical organization of LabVIEW functions.

Click and then drag diagonally to form the While Loop to the area you desire. You can resize the While Loop by dragging any of the resizing boxes that appear when your cursor hovers above the loop's edges.

1. You can also create a While Loop by pressing **<Ctrl+Space>** to bring up the Quick Drop dialog. Begin typing "While Loop" and it will appear in the list of possible objects. Double-click its name and it will appear on your cursor for use on the block diagram. Since you've already placed the While Loop, release the While Loop you found using Quick Drop by **right-clicking** to cancel the operation.

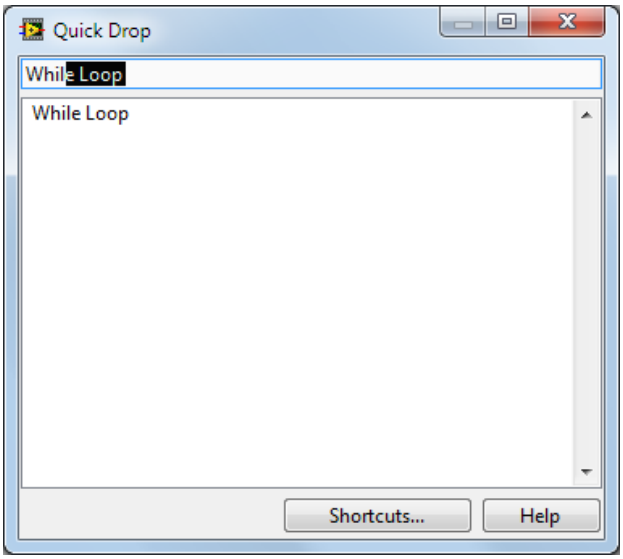


Figure 4. If you know the name of what you're looking for, there is no faster way to program than by using Quick Drop.

5. While Loops have two terminals in their bottom left and right corners.

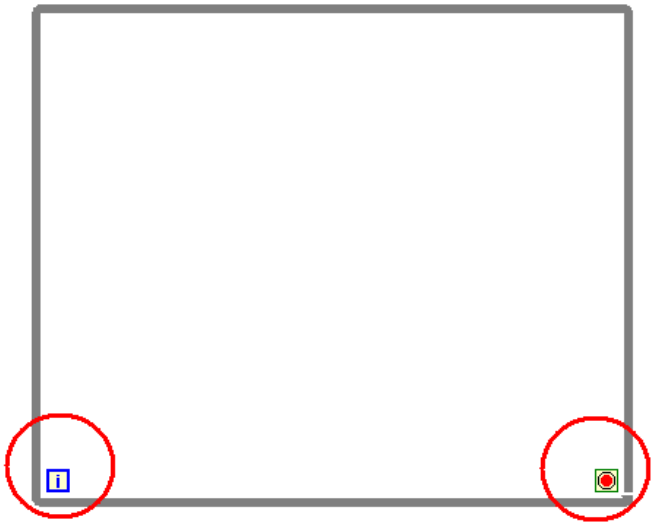


Figure 5. Notice the two terminals on a standard LabVIEW While Loop.

The most important of the two is the loop condition. The conditional terminal is on the lower right side and looks like a stop sign by default. Since While Loops run until told to stop, we must provide a (Boolean) stop command so that the loop won't run indefinitely.

Notice the broken run arrow ( ) in the upper left corner of the screen. LabVIEW cannot execute an application that contains a While Loop with an un-wired conditional terminal. For our application, we need to create a Stop button that the user will press to halt the While Loop and exit the program.

6. On the front panel, right-click on any empty space to bring up the Controls palette and navigate to **Silver » Boolean » Stop Button**. Left-click on the Stop button and it will be automatically placed on your cursor.

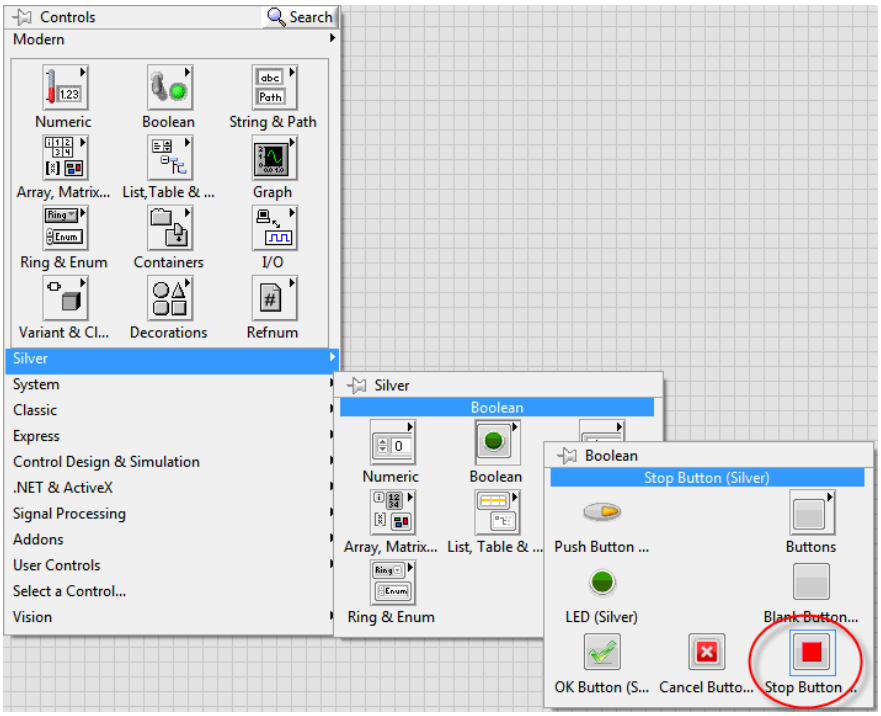


Figure 6. LabVIEW contains dozens of engineering-specific user interface elements.

7. Left-click where you would like to place the button on the front panel. If you prefer, enlarge the Stop button by moving your cursor to one of the button's edges and dragging the resizing boxes.

8. Look again at the block diagram. Notice that a terminal for the stop button has appeared. This terminal acts as the connector from the front panel to the functionality of the block diagram. Click the stop button terminal and drag it next to the loop condition terminal ( ) in the While Loop.

9. Move your cursor to the right edge of the stop button terminal and notice that the edge of the terminal is blinking and the cursor now looks like a spool. This is the wiring tool that lets you draw wires between different objects on the block diagram.

10. Left-click on the right edge of the stop button terminal.

11. Left-click on the left edge of the While Loop's condition terminal.

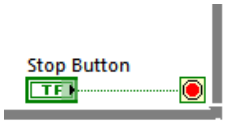


Figure 7. The wire carries data between the button and the conditional terminal.

With the While Loop now having a way to exit (the state of the stop button), the broken Run arrow ( ) is replaced with a Run arrow ( ). Your application is ready to run, but you'll need to add more code to accomplish the tasks of this exercise.

The other terminal in the While Loop, the loop iteration counter ( ), outputs the number of times the While Loop has iterated. That information may be useful depending on your application, but we will not be using it today; it is not required that we do anything with it in order to run our program.

12. Press **<Ctrl+Space>** to bring up the Quick Drop dialog and begin to type "Simulate Signal." Double-click *Simulate Signal* once you see it in the Quick Drop window and the *Simulate Signal Express VI* will automatically appear on your cursor.

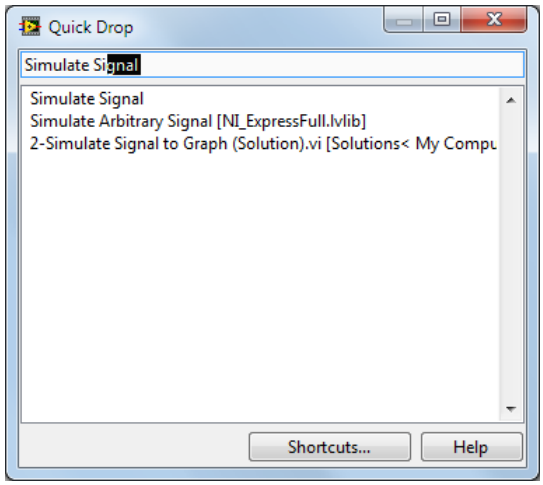


Figure 8. Quick Drop pre-indexes all LabVIEW functions for expedited search.

13. Left-click to place the Simulate Signal Express VI inside the While Loop and its configuration dialog will automatically appear.

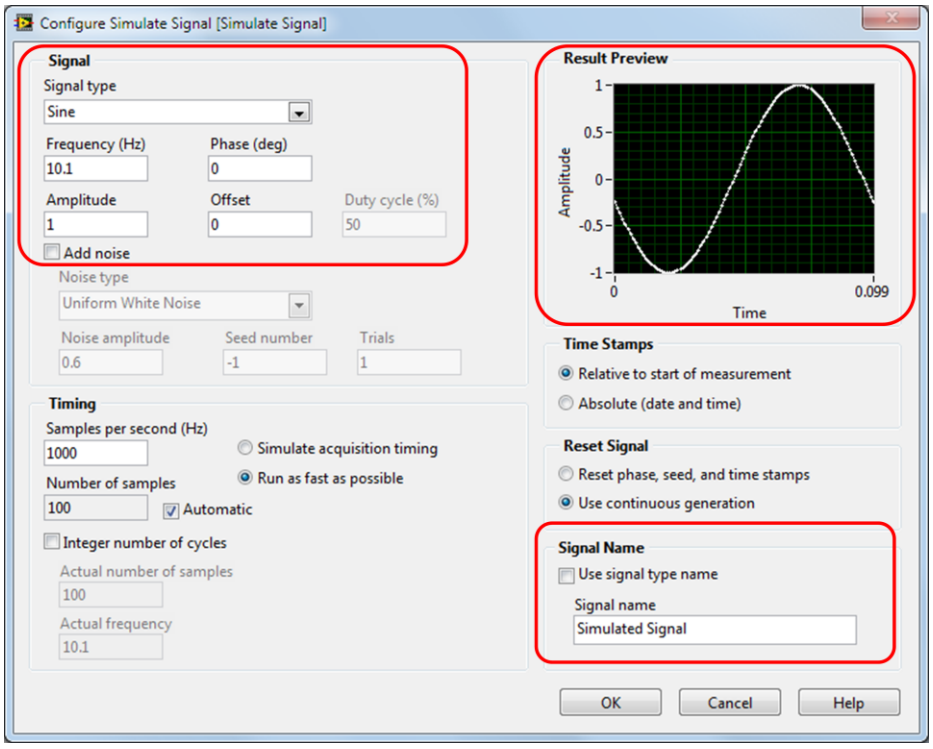


Figure 9. All LabVIEW Express VIs have easy-to-use configuration dialogs.

14. Try changing the Signal Type, Amplitude, Frequency, Offset, and Phase values in the *Signal* portion of the dialog and see the changes in the *Results Preview* portion.

15. Deselect the **Use signal type name** box in the Signal Name section and enter *Simulated Signal* as the name.

16. Once you have chosen the signal you want to display, press **OK**. The Simulate Signal Express VI has now been customized based on the settings you provided.

17. Switch to the front panel and use **<Ctrl+Space>** to launch Quick Drop.

18. Type the word *chart*. Double-click on the Waveform Chart (Silver) control.

19. Place the Waveform Chart (Silver) on the front panel in any the location you prefer.

20. Return to the block diagram and move the chart's icon into the While Loop (if it isn't already) to the right of the Simulate Signal Express VI.

21. Use a wire to connect the output of the Simulate Signal Express VI ("Simulated Signal") to the input Waveform Chart terminal. Notice that the chart terminal changed colors to reflect the data type it received.

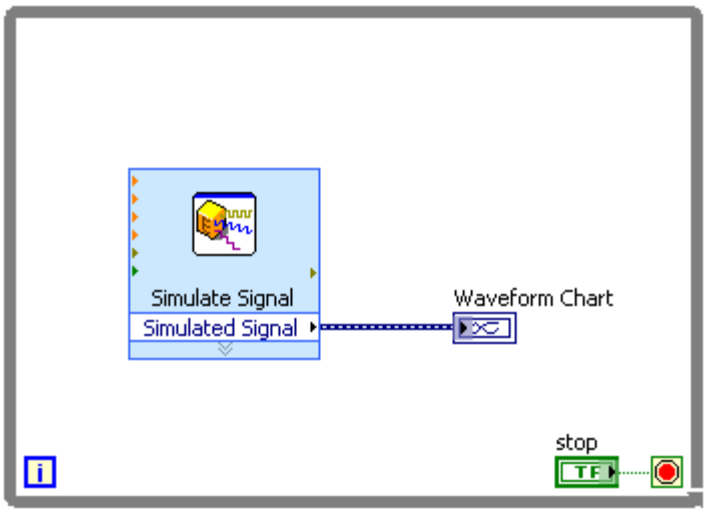


Figure 10. The Simulated Signal will be displayed on the Waveform Chart during each iteration of the While Loop.

22. Return to the front panel and **Run** the VI by pressing the Run arrow ( ). The simulated signal you created in the Express VI is now displayed on the chart. Press the **Stop** button when you are ready to move on.

23. Right-click on an empty space on the front panel to bring up the Controls palette, find the knob control (**Silver » Numeric » Knob (Silver)**) and place it on the front panel. Double-click on the knob's label and change it to Amplitude.

24. Repeat step 23 to make another knob. Change its label to *Frequency*. Double-click the maximum value on Frequency's scale (default = 10) and change it to 50.

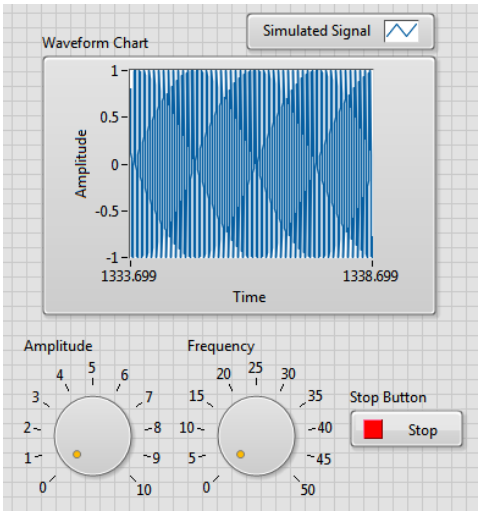


Figure 11. The front panel with controls for the signal's amplitude and frequency.

25. On the block diagram, move the *Amplitude* and *Frequency* controls inside of the While Loop.

26. Hover your cursor over the right side of each of the Amplitude and Frequency terminals until the wiring tool appears on the cursor. Left-click and drag the connection to the identically named input on the Simulate Signal Express VI. Your block diagram should look like the image below.

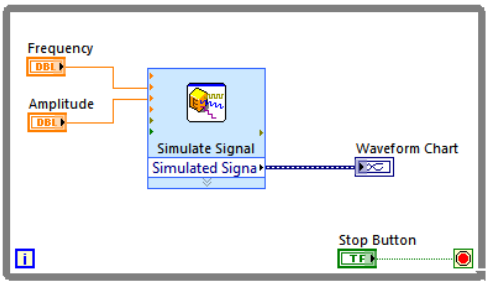


Figure 12. You can now programmatically change the frequency and amplitude.

27. Press the **Run** arrow, manipulate *Amplitude* and *Frequency* knobs and notice that the chart display changes accordingly. The Chart's Y-axis auto-scales to maximize the signal's size on the display. To disable that feature, right-click the chart and deselect AutoScale Y.

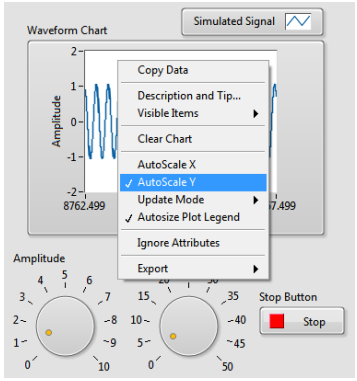


Figure 13. Sometimes you want to manually control the scale of a graph.

You can now change the upper and lower ranges of the Y-axis by clicking on the numbers along the axis and typing in new values.

28. Stop the VI by pressing the **Stop** button.

Additional Steps

LabVIEW provides several tools that can help you develop your applications. The next few steps will show how to use some of the most important programming assistance tools.

Block Diagram Cleanup

As you program, and especially as you learn how to program in LabVIEW, you are not always thinking about layout and readability. This can result in a poorly organized block diagram.

LabVIEW's Block Diagram Cleanup is a built-in tool that organizes your code, making it easier for you and others to understand how your program functions.

29. Press the Block Diagram Cleanup icon found on the menu bar.

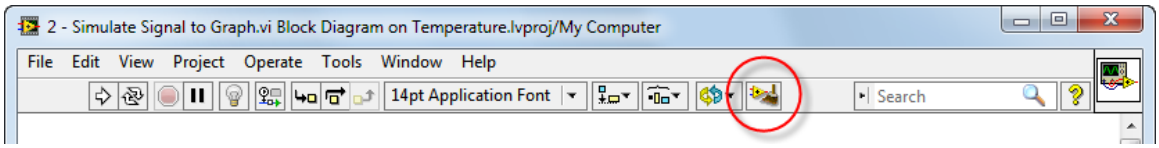


Figure 14. Block Diagram Cleanup will automatically organize your code.

Your block diagram should now be organized, with cleaner wires and an even distribution of code elements.

Highlight Execution

30. Press the **Highlight Execution** button on the menu bar. Notice that the light bulb icon now appears to be on.

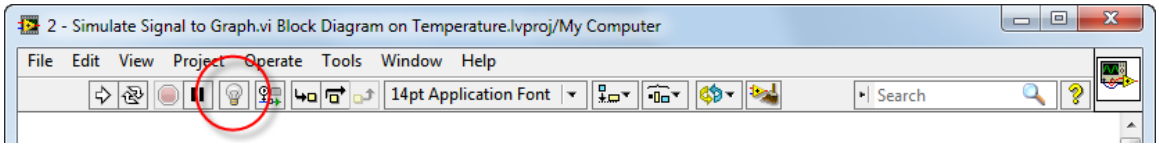


Figure 15. Highlight Execution slows your code to let you visualize dataflow.

31. Run your application with Highlight Execution turned on. Press the **Run** arrow and watch as your code executes step-by-step. While not always necessary for simple applications, the Highlight Execution tool is a powerful resource for troubleshooting complex programs and determining if your code performs as expected.

Context Help

32. Press the **Context Help** button in the upper right portion of the block diagram.

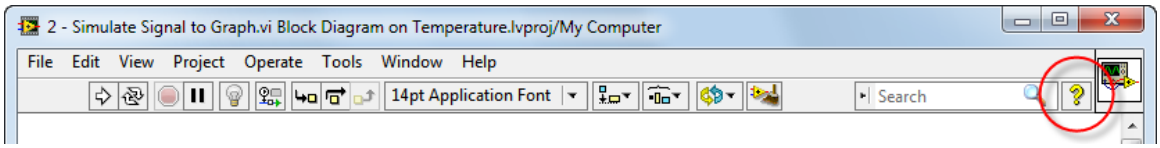


Figure 16. Context Help displays a context-based help dialog.

33. With Context Help active, hover your cursor over different objects on the block diagram and front panel of Simulate Signal to Graph.vi. As you do so, the Context Help Window provides details including descriptions and wiring diagrams.

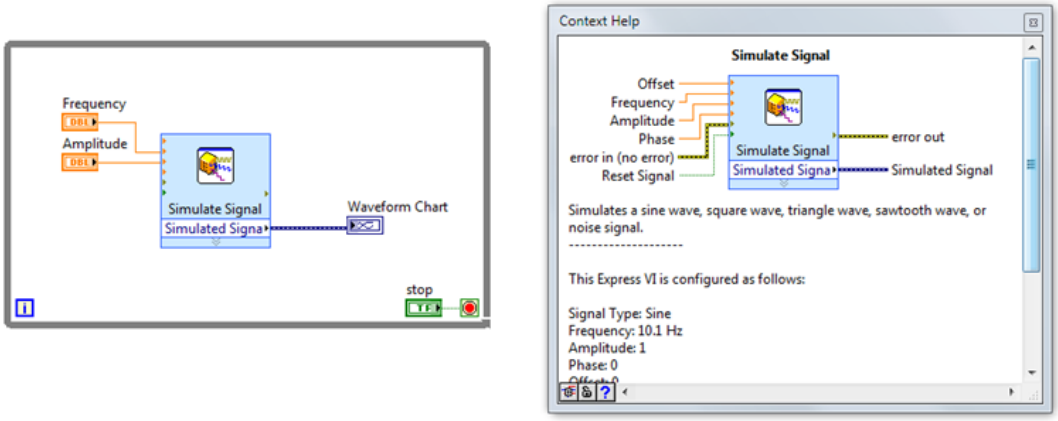


Figure 17. Context Help is a quick way to get a VI's overview, inputs, and outputs.

34. Right-click the block diagram and navigate around the palettes. Notice that the Context Help window provides details on the objects while they are in the palettes. Also notice that for some objects, the Context Help window provides a link for Detailed Help. This link will open the *LabVIEW Help* and give you more information.

35. Save *2 - Simulate Signal to Graph.vi* and close.

Part C

The purpose of this exercise is to use LabVIEW and NI CompactDAQ to quickly set up a program to acquire temperature data.

1. Right-click the Exercises folder and select **New » VI**. Once opened, save the VI in the Exercises folder under the name *3 - Basic Measurement.vi*.

2. Press **<Ctrl+T>** to tile front panel and block diagram windows.

3. Pull up the Functions Palette by right-clicking the white space on the LabVIEW block diagram window.

4. Move your mouse over the **Express » Input** palette and click the *DAQ Assistant Express VI*. Left-click the empty space to place it on the block diagram.

Figure 1. The DAQ Assistant is a configuration-based way to acquire data from real or simulated hardware.

5. The *Create New…* window then appears:

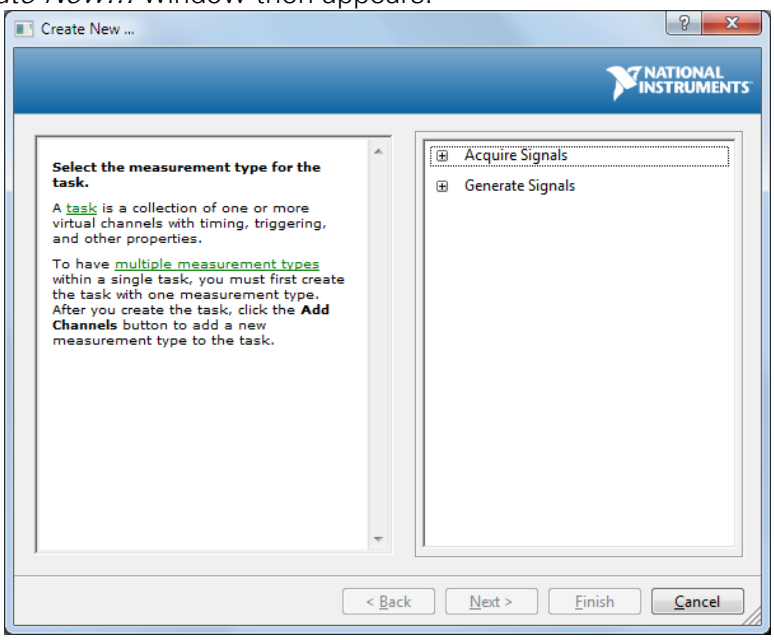


Figure 2. NI-DAQmx tasks are a collection of channels with homogenous timing and triggering properties.

6. To configure a temperature measurement application with a thermocouple, click **Acquire Signals » Analog Input » Temperature » Thermocouple**.

7. Click the **+** sign next to the module named **Temperature (NI 9213)**, highlight channel **ai0**, and click **Finish**. This adds a physical channel to your measurement task.

8. Within the DAQ Assistant configuration dialog, change the CJC Source to **Built In** and Acquisition Mode to **Continuous Samples**.

9. Change the *Samples to Read* to **100**.

10. Change the *Rate (Hz)* to **1k**.

11. Click the **Run** button. You will see the temperature readings from the thermocouple in test panel window.

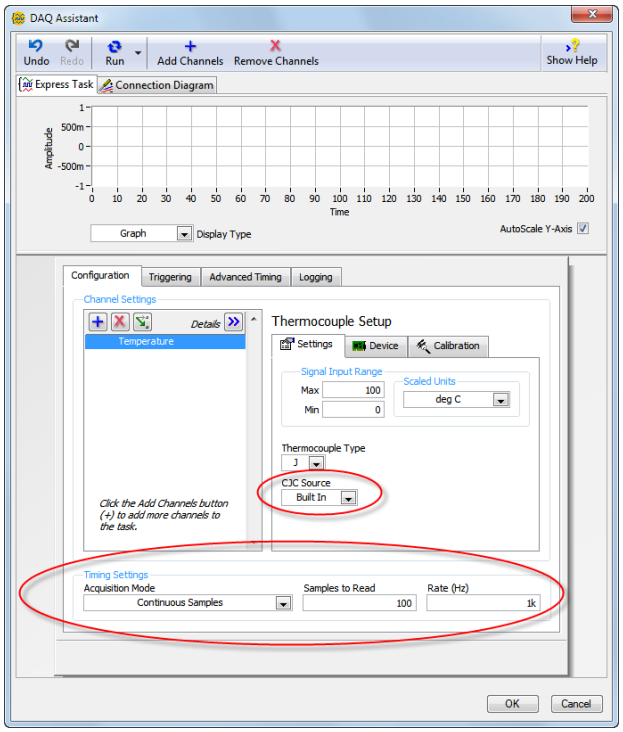


Figure 3. You can configure and test your measurement from within the DAQ Assistant window.

12. Click **Stop** and then click **OK** to close the Express block configuration window to return to the LabVIEW block diagram.

13. LabVIEW automatically creates the code for this measurement task. Since we select a Continuous Samples measurement, it makes sense for the code to be placed within a While Loop, which LabVIEW informs us. Click **Yes** to automatically create a While Loop.

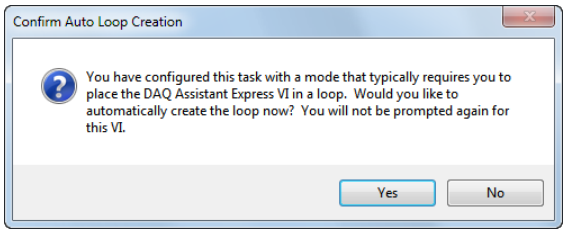


Figure 4. LabVIEW notifies us that the code should likely be placed within a loop.

14. Right-click the data terminal output on the right side of the DAQ Assistant Express VI and select **Create » Graph Indicator**.

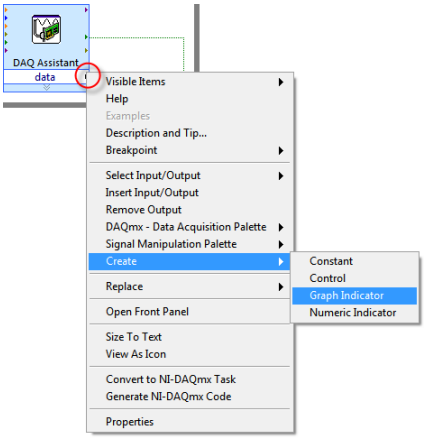


Figure 5. You can automatically create a front panel object from the block diagram.

15. Rename the new Waveform Graph indicator from *Data* to *Temperature*.

16. Notice that a corresponding graph indicator is placed on the front panel.

17. Your block diagram should now look like the figure below. The While Loop automatically adds a Stop button to your front panel that allows you to stop the execution of the loop.

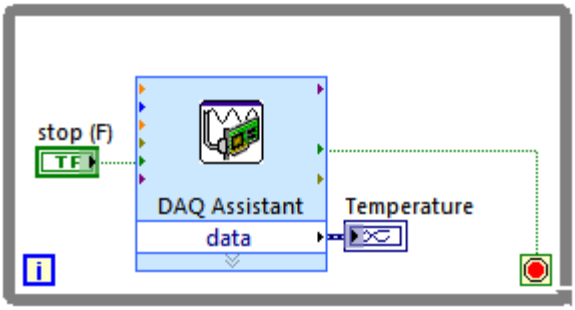


Figure 6. The fully-functional application that acquires temperature data.

18. Feel free to **Run** the VI to test its functionality. Press **Stop** when you are finished.

19. **Save** the VI.

Additional Steps

Express VIs make creating basic applications very easy. Their configuration dialogs allow you to set parameters and customize inputs and outputs based on your application requirements. However, to optimize your DAQ application's performance and allow for greater control, you should use standard DAQmx driver VIs. The DAQ Assistant can automatically generate standard DAQmx code; not all Express VIs have this functionality.

Before you generate standard DAQmx code from the DAQ Assistant, you need to remove any unnecessary code that was automatically created by the Express VI.

20. Right-click the While Loop and select **Remove While Loop**. Then click the **Stop** button and press the **<Delete>** key to remove the Stop button. Repeat these

actions for the Temperature Graph as well as any additional wires that may remain. You can press **<Ctrl+B>** to remove all unconnected wires from a block diagram.

21. When you are left with only a DAQ Assistant, right-click on the DAQ Assistant Express VI you created in this exercise and select **Generate NI-DAQmx Code**.

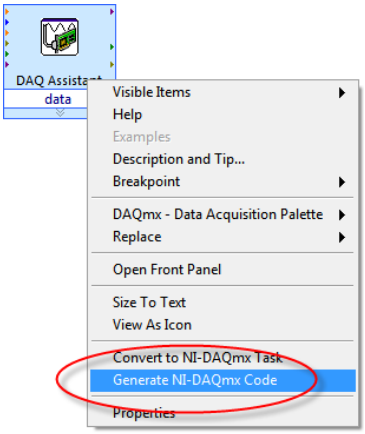


Figure 7. Generate NI-DAQmx code from the DAQ Assistant's configuration.

Your block diagram should now appear something like this:

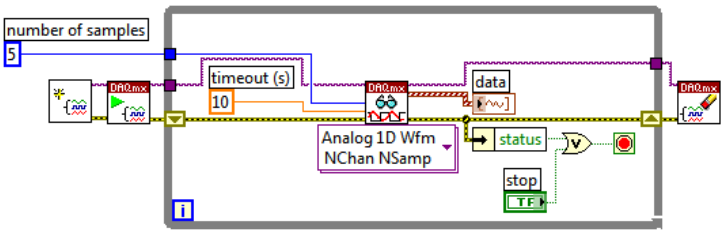


Figure 8. The lower-level DAQmx equivalent of the DAQ Assistant's configuration.

The Express VI has been replaced by four VIs. We'll examine their functionality in the following steps.

22. Open Context Help by clicking the **Context Help** icon ( ) on the upper-right corner of the block diagram.

23. Hover your cursor over each VI and examine their descriptions and wiring diagram.

24. *DAQmx Read.vi* reads data based on the parameters it receives from the *Untitled VI* on the far left.

25. Double-click the *Untitled VI* and open that VI's block diagram (code shown below).

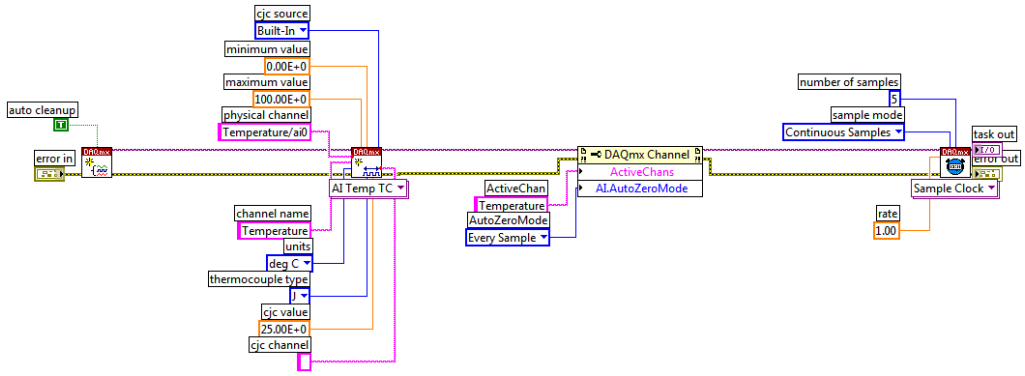


Figure 9. The Untitled VI contains all of the channel configuration information including but not limited to measurement type, scaling, and timing.

All the parameters that are wired as inputs to the different DAQmx setup VIs reflect the settings you originally configured in the DAQ Assistant Express VI.

Note: By moving these parameter and setup VIs onto the block diagram, you could now programmatically change their values without having to stop your application and open the Express VI configuration dialog. This can save development time and possibly optimize performance by eliminating unnecessary settings depending on your application.

26. Close the *Untitled VI*. When prompted, **do not save**.

27. Close *3 – Basic Measurement.vi*. When prompted, **do not save**.

Part D

The purpose of this exercise is to expand our application to include thresholding that will trigger a user-defined alarm.

1. From within the *Exercises* folder of the open *Temperature* project, open **4 – Analysis and Output.vi**. This VI is functionally equivalent to the VI we created in the previous exercise, but already provides additional space on the block diagram to add additional code.

2. Right-click the front panel to open the Controls palette and navigate to **Controls » Silver » Numeric**. Select and place a **Numeric Control** on the front panel.

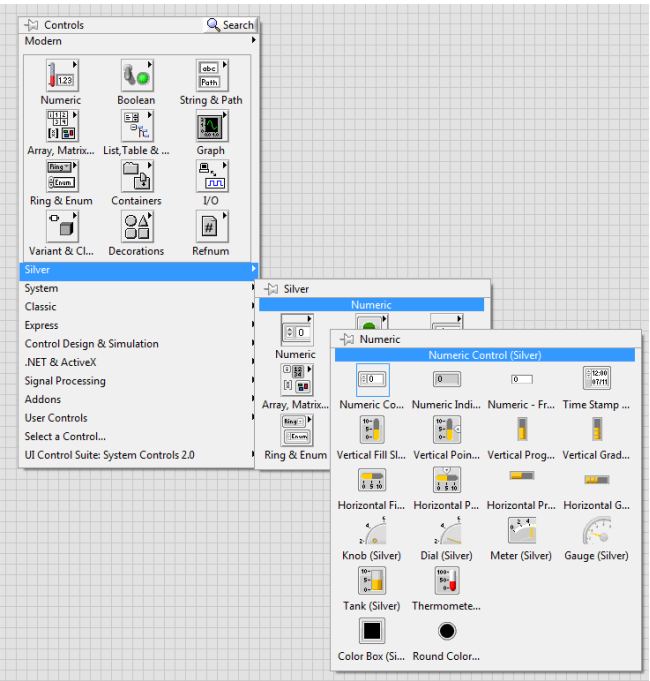


Figure 1. Add a numeric control to the front panel for specifying the alarm threshold.

3. Double-click the numeric control's label and rename the control *Alarm Level*.

4. Switch to the block diagram; if it isn't already, drag-and-drop the Alarm Level control into the while loop.

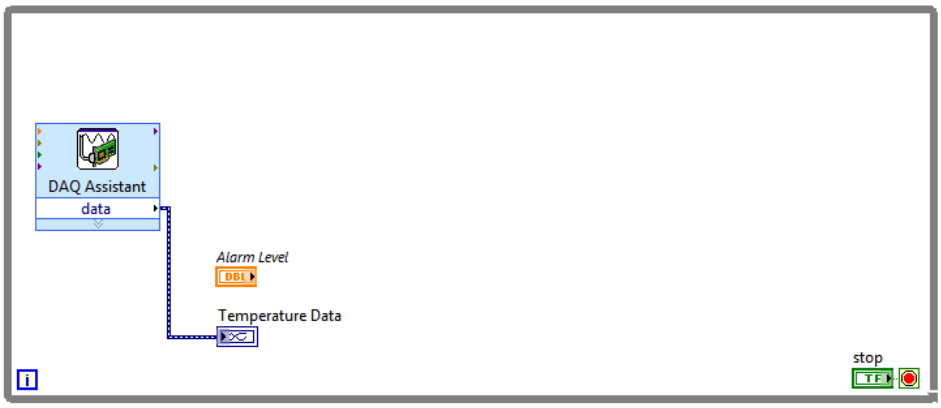


Figure 2. The resized while loop allows us more space to add code.

5. Press **<Ctrl+Space>** to launch Quick Drop and type "Comparison" to find the *Comparison Express VI*.

6. Double-click *Comparison* in the Quick Drop results list and place it within the while loop.

Once placed on the block diagram, the Comparison Express VI's configuration dialog will appear.

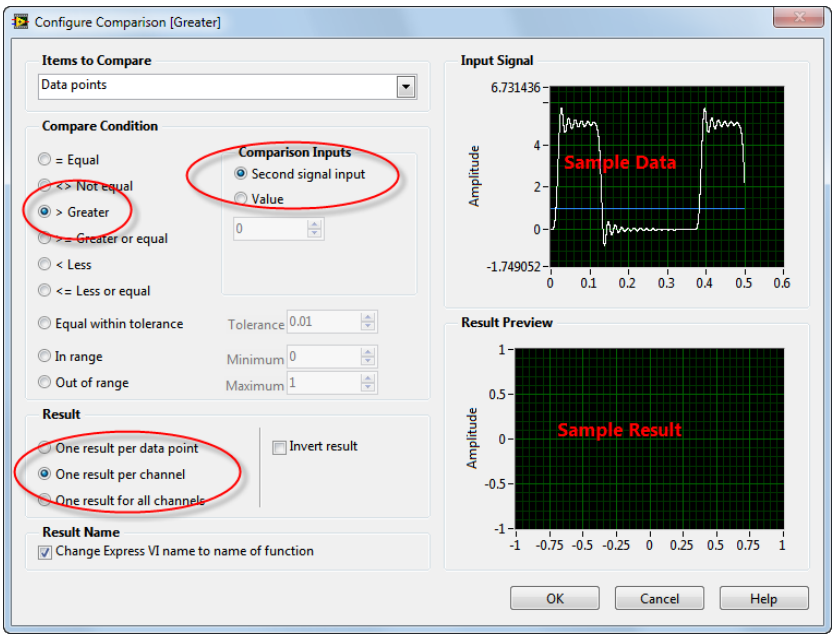


Figure 3. The configuration dialog of the Comparison Express VI.

7. Select **> Greater** in the Compare Condition section, **Second signal input** from the Comparison Inputs section, and **One result per channel** from the Result section. Click **OK**.

8. Hover over the *data* output of the DAQ Assistant until the spool icon appears on your cursor, then left-click and drag your mouse to the **Operand 1** input on the Comparison Express VI. Perform the same hover, drag, and connect to wire the Alarm Level control and the **Operand 2** input on the Comparison Express VI.

9. Your block diagram should now look something like this:

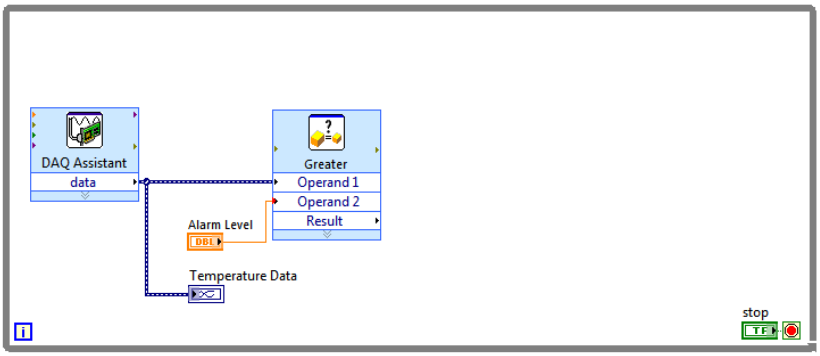


Figure 4. Comparing the acquired data to a user-specified alarm level.

10. On the front panel, right-click and navigate to **Controls » Silver » Boolean » LED (Silver)**. Place the LED on the front panel and resize the LED so that it is easier to see.

11. Rename the LED *Alarm*. Your front panel should look like this:

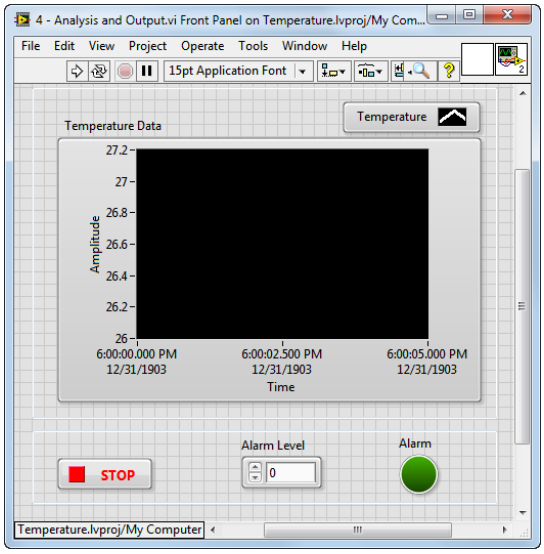


Figure 5. The UI now contains mechanisms for user input and data display.

12. If it isn't already, drag-and-drop the Alarm indicator into the while loop.

13. On the block diagram, wire the output of the *Comparison Express VI* to the input of the *Alarm* indicator's terminal.

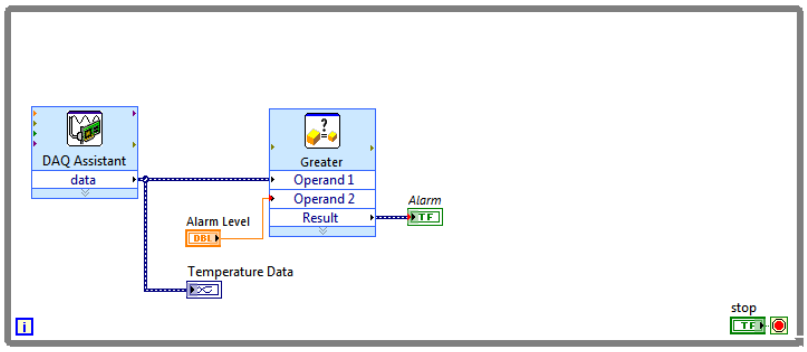


Figure 6. At this point, the application will notify the user if the measurement exceeds a specified threshold.

14. Press the **Run** button and change the Alarm Level control to some level above the current acquired temperature signal. Hold the thermocouple until the temperature exceeds the Alarm Level value. The Alarm LED turns on when the acquired temperature signal goes above the level set on the front panel. Next, we will update a digital line depending on this LED status.

15. Stop the VI using the **Stop** button on the front panel.

16. Switch to the block diagram. Right-click to access the functions palette and select the DAQ Assistant Express VI from **Functions » Express » Output**. Place the DAQ Assistant on the block diagram within the while loop.

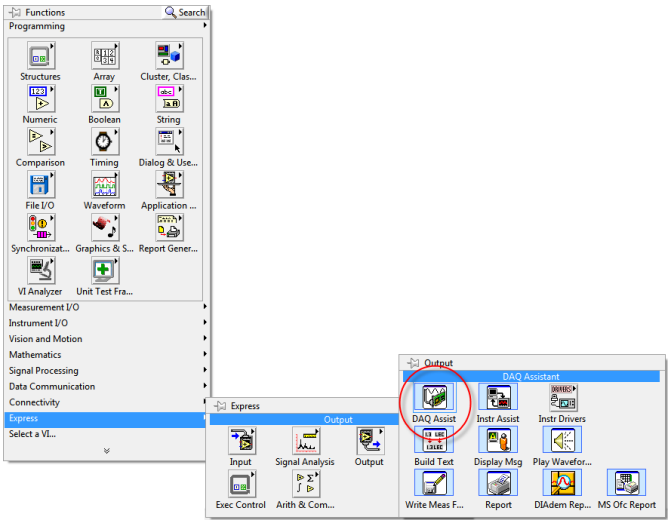


Figure 7. Adding another DAQ Assistant will allow us to control the digital module.

17. Within the *Create New…* dialog, select **Generate Signals » Digital Output » Line Output**

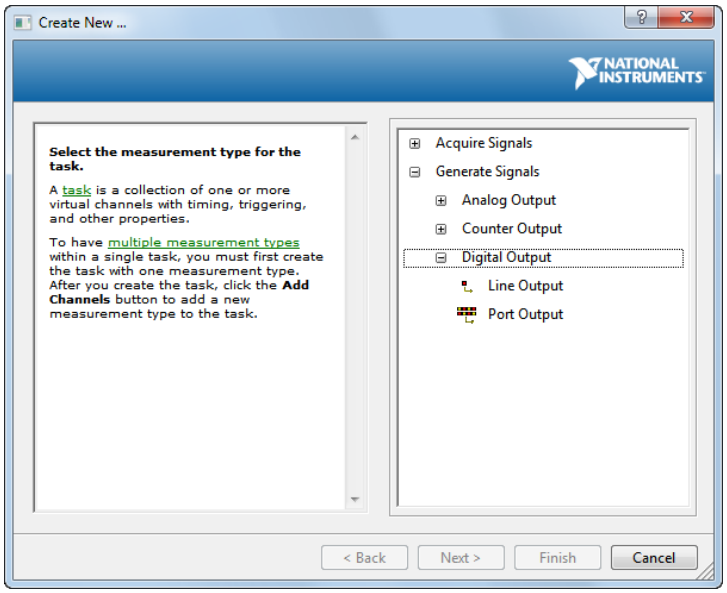


Figure 8. A digital line output allows us to generate data to one single digital line.

18. Expand the **+** sign next to the *Digital_out* module in the following window and select **port0/line0**, then click **Finish**.

19. Press **OK** in the DAQ Assistant window that appears, since all of its default settings are correct for the application.

20. Create an additional wire that connects the Comparison Express VI's **Result** output to the **data** input on the new *DAQ Assistant2 Express VI*.

A *Convert from Dynamic Data* function appears automatically. LabVIEW will always try to coerce unlike data types when two nodes are wired together. In this case, the output of

the Comparison Express VI is a Dynamic Data type, and the input of the DAQ Assistant is Boolean. LabVIEW placed the *Convert from Dynamic Data* node in between the two nodes so they could be connected without a syntax error. Your block diagram should now look like this:

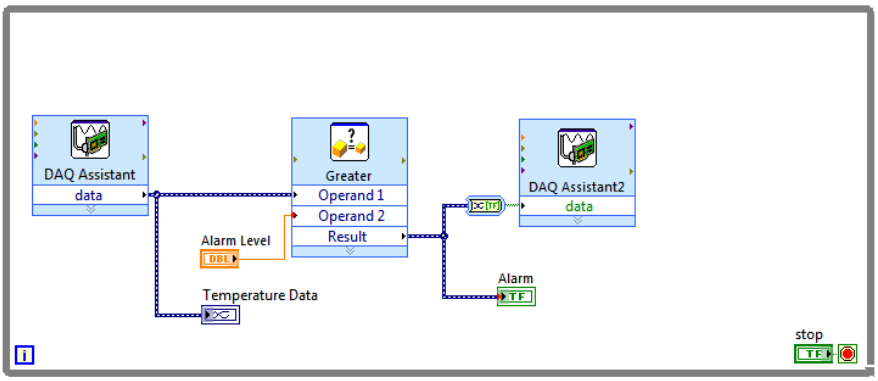


Figure 9. The threshold comparison result will be output to the digital module.

21. Press the **Run** button. Notice that the LED bank on the CompactDAQ 9472 module turns on and off to match *Alarm's* value on the front panel.

22. Save the VI.

Part E

The purpose of this exercise is to save our acquired data to file for future processing, which could include sharing with colleagues, turning into a report, running additional analysis, or all of the above.

1. The *4 – Analysis and Output.vi* should still be open from the previous exercise. If not, open it now from the *Exercises* folder in the *Temperature* project.

2. Right-click the block diagram and select **Functions » File I/O » Write to Measurement File**. Place the Express VI inside the While Loop on the block diagram.

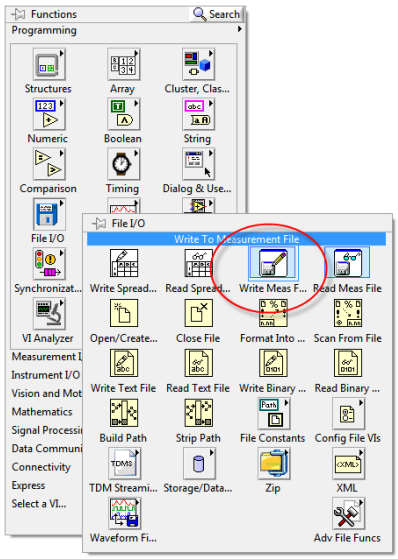


Figure 1. LabVIEW includes dozens of options for reading and writing files.

3. Since this is an Express VI, a configuration window will appear. Configure the window as shown below, but be sure to make a mental note of the file location in the Filename field. When finished, click **OK**.

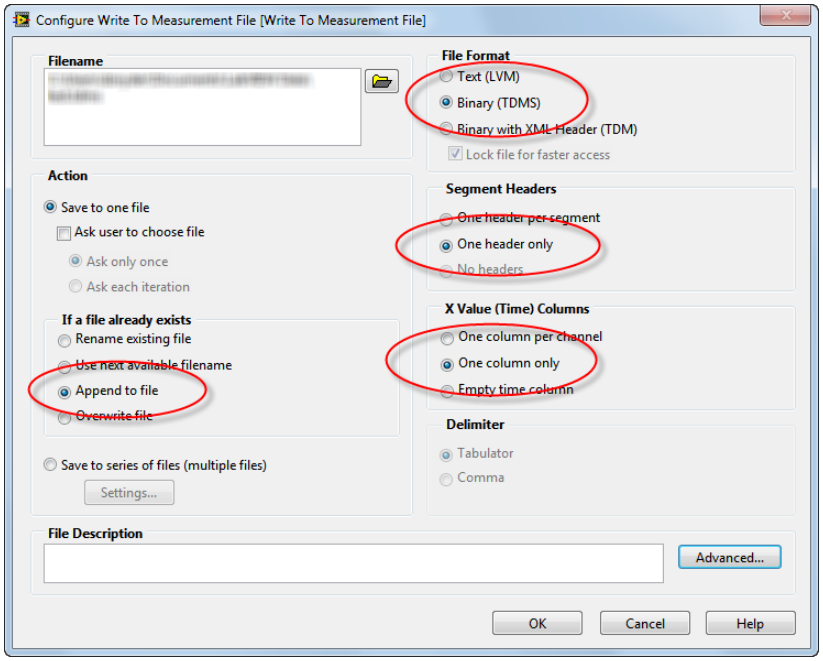


Figure 2. This configuration will save a binary file that is able to be post-processed in common software such as NI DIAdem or Microsoft Excel.

4. Wire the output of the *DAQ Assistant Express VI* to the input of the *Write to Measurement File Express VI*.

5. Your block diagram should now resemble the following figure.

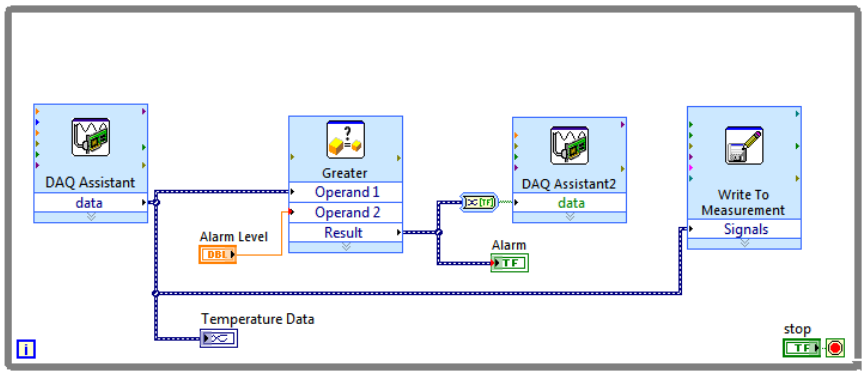


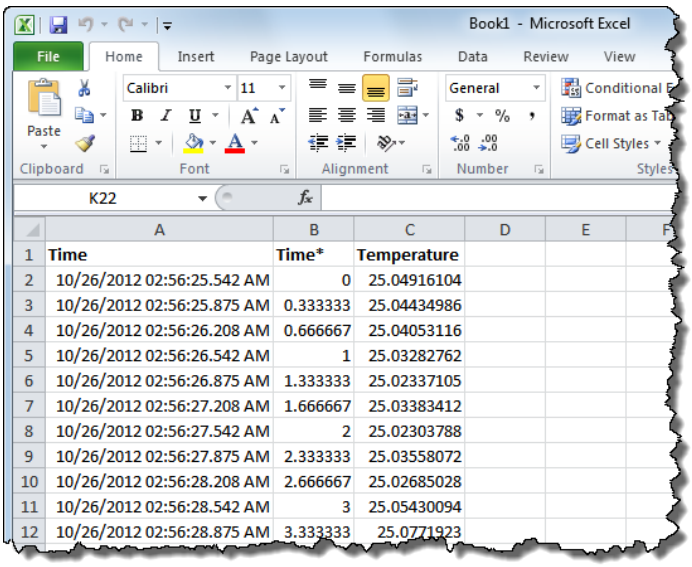
Figure 3. This completed application acquires temperature data, outputs a digital signal if the temperature exceeds a threshold, and saves all of the data to file.

6. Run the VI momentarily and press **Stop** to stop the VI.

7. Your file will be created in the folder specified. If you can't remember the location of the file, double-click the Write to Measurement File Express VI.

8. Using Windows Explorer, navigate to the location of the data file on disk.

9. Right-click on the file in Windows Explorer and select **Open With » Excel Importer** to open it with Microsoft Excel. Review the header and temperature data saved in the file. Note that they may be contained within different Excel worksheets (tabs).



	A	B	C	D	E
1	**Time**	**Time***	**Temperature**		
2	10/26/2012 02:56:25.542 AM	0	25.04916104		
3	10/26/2012 02:56:25.875 AM	0.333333	25.04434986		
4	10/26/2012 02:56:26.208 AM	0.666667	25.04053116		
5	10/26/2012 02:56:26.542 AM	1	25.03282762		
6	10/26/2012 02:56:26.875 AM	1.333333	25.02337105		
7	10/26/2012 02:56:27.208 AM	1.666667	25.03383412		
8	10/26/2012 02:56:27.542 AM	2	25.02303788		
9	10/26/2012 02:56:27.875 AM	2.333333	25.03558072		
10	10/26/2012 02:56:28.208 AM	2.666667	25.02685028		
11	10/26/2012 02:56:28.542 AM	3	25.05430094		
12	10/26/2012 02:56:28.875 AM	3.333333	25.0771923		

Figure 4. The resulting temperature data file that was saved from the completed LabVIEW application.

10. Close the data file, if it is still open.

11. Close the LabVIEW VI, if it is still open.

12. Close the Temperature Project, if it is still open.

<End of Exercise>

Combining Measurements from Light and Temperature

Goals

- In this exercise, you will acquire a light signal using NI-DAQmx
- In the second part of this exercise, you will modify the example to scale the data and measure the ambient light in the Measurements Demo Box

Part A Open and Run Our Starting Point

Estimated time: 20 minutes

Today's exercises start with a pre-built single-channel measurement system that measures temperature data from a thermocouple. We will expand this system to acquire mixed-measurement data from two sensors (a thermocouple and a solar cell) using two different modules and write data to a scalable file format.

1. If you have not done so already, launch NI LabVIEW by selecting **Start » All Programs » National Instruments » LabVIEW <Year> » LabVIEW**.
2. Open the Light Sensor Project.
    a. In the LabVIEW Getting Started Window, select **Open Existing**.
    b. Navigate to **C:\Seminars\LV HO\Exercises\4 – Light**.
    c. Select and open **Light Sensor Project.lvproj**.
3. Open and run the starting point application.
    a. In the LabVIEW Project, double-click to select **Exercises » 1 – Write Temperature to ASCII File (Original).vi**.
    b. Run the application by clicking the **Run** arrow () on the front panel as shown in Figure 1.
    c. While the application is running, use your finger to warm the thermocouple attached to the Analog Input 0 (**ai0**) channel of the NI 9213 module.
    d. Verify that the temperature signal on the LabVIEW front panel graph rises and falls appropriately, and that the LED indicator turns on when the temperature exceeds 26 °C. When the LED indicator turns on, the physical LED on the NI 9472 digital output module should also light.
    e. Stop the VI by selecting the **Stop button** on the LabVIEW front panel as shown in Figure 1.

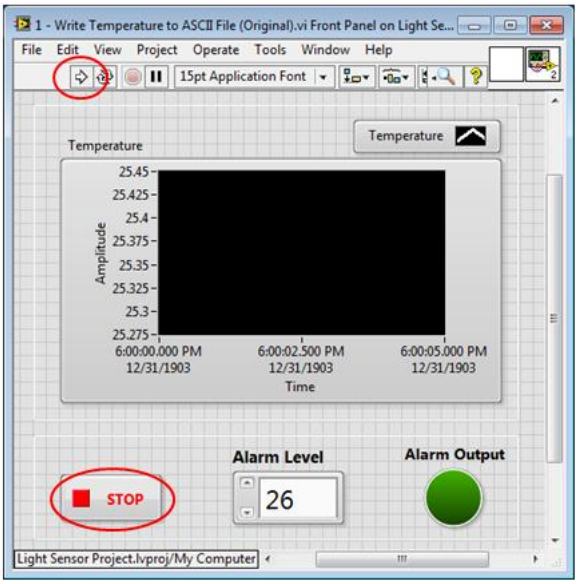


Figure 1. Use the Run arrow to run the LabVIEW application.

4. Press **<Ctrl+E>** to view the Block Diagram.

   This LabVIEW code uses the DAQ Assistant to acquire data - an Express VI with built-in configuration steps, eliminating the need for many of the most common device configurations. That said, the DAQ Assistant has some limitations when you need more control of your DAQmx device, like synchronization or specific device parameters that are outside the scope of this seminar. In the coming exercises, you will learn the basics of using the standard (non-Express) NI-DAQmx API. This lower-level programming method opens up the capabilities of the device and in the long run, will make you a better user.

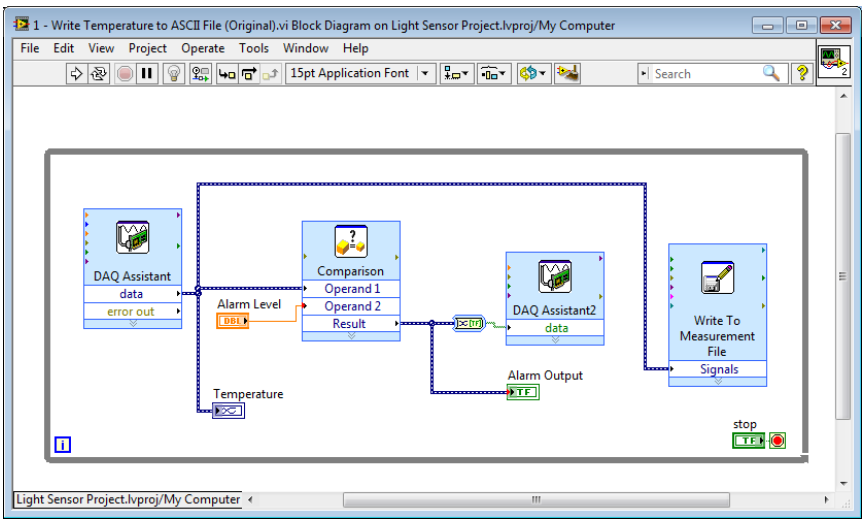


Figure 2. Using a DAQ Assistant is an easy way to get up and running acquiring data.

5. **Close** the VI.

Part B Use the Low-Level NI-DAQmx API

Estimated time: 20 minutes

Though the DAQ Assistant used in the last exercise got the job done, it isn't quite as flexible or scalable for mixed-measurement applications. This exercise will replace the DAQ Assistant and accomplish the equivalent measurements using the more flexible lower-level NI-DAQmx VIs.

1. Open the exercise application.
    a. In the LabVIEW Project, double-click to select **Exercises » 2 – Use the NI-DAQmx VIs (Original).vi**.
2. Examine the lower-level NI-DAQmx code.
    a. Select **Window » Show Block Diagram** or press **<Ctrl+E>** to examine the LabVIEW code.
    b. If it is not already open, select **Help » Show Context Help** or press **<Ctrl+H>** to turn on LabVIEW Context Help.
    c. Navigate to the left-most section of code on the diagram, as shown in Figure 3.

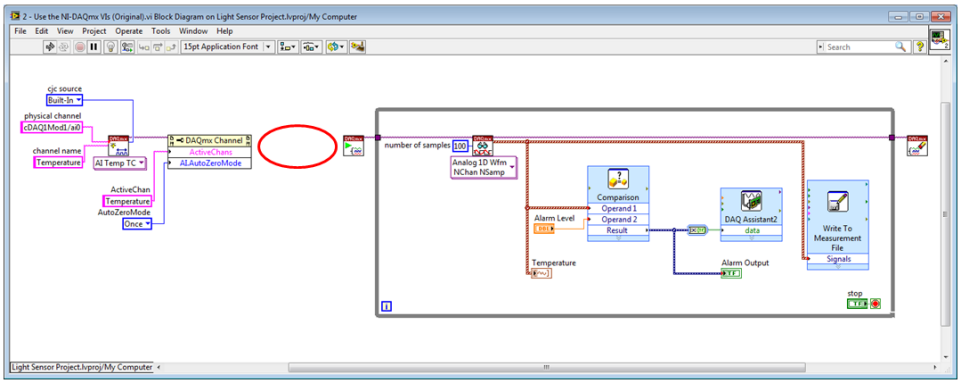


Figure 3. The lower-level NI-DAQmx API follows a logical pattern.

    d. One at a time, hover over the VIs on the block diagram and examine their functionality as described in the Context Help window, as shown in Figure 4.

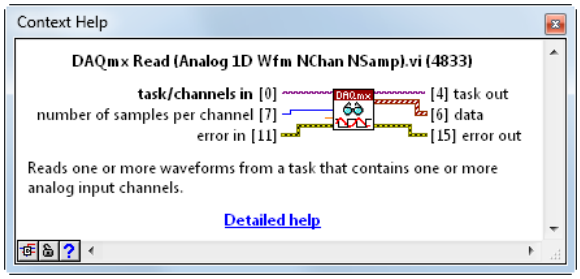


Figure 4. The Context Help dialog gives you detailed information about every VI, as well as its expected inputs and output values.

Notice that the logical flow of the NI-DAQmx API begins by configuring an analog input channel that corresponds to a physical channel of measurement.

Just before the while loop, the acquisition is started. Within the loop, samples are read from the NI-DAQmx buffer repeatedly until the user stops the loop, at which point the acquisition setup is cleared from memory.

3. Add Timing configuration to the lower-level NI-DAQmx code.

    a. Right-click in the white space on the block diagram to access the functions palette.

    b. Navigate to **Measurement I/O » NI-DAQmx** and select **DAQmx Timing.vi**.

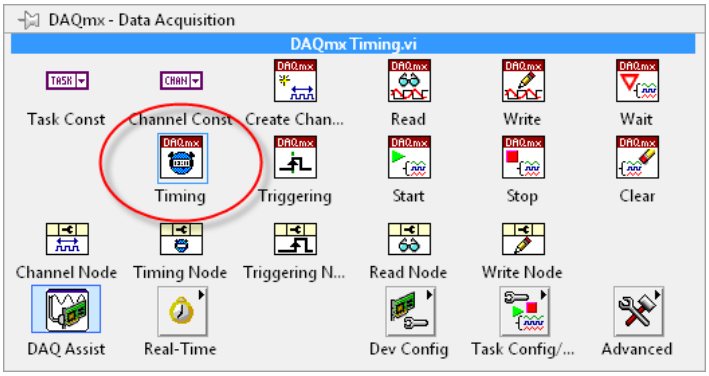


Figure 5. The NI-DAQmx palette houses all of the data acquisition functions built into the LabVIEW environment by the NI-DAQmx driver.

    c. Left-click to place the DAQmx Timing.vi between the *DAQmx Create Channel VI* and the *Start Task VI* as shown in Figure 6.

    d. Right-click on the *Sample Mode* input terminal of the DAQmx Timing.vi and select **Create » Constant** from the right-click context menu.

    e. Instead of Finite Samples, change the Sample Mode input to **Continuous Samples**.

    f. Right-click on the *Rate* input and select **Create » Constant** from the right-click context menu. Leave the default rate value of 1000 Hz.

    g. Right-click on the *Samples per Channel* input and select **Create » Constant**.

    h. Double-click the Samples per Channel constant and change the value to **100**.

    i. Click on the output of the **Task Out** output of the *DAQmx Create Channel.vi*, then click again on the **Task/Channels In** input of the *DAQmx Timing.vi.* This will connect a wire between the two nodes.

j. Similarly, wire the **Task Out** output of the DAQmx Timing.vi to the **Task/Channels In** input of the DAQmx Start Task.vi.

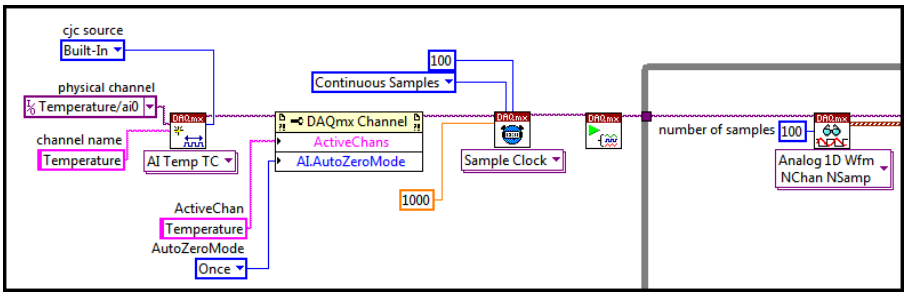


Figure 6. Complete the configuration of the DAQmx Timing.vi as shown.

4. Run the VI.
   a. Using the **Run** arrow (), run the VI and ensure that it behaves with the lower-level NI-DAQmx VIs exactly as it did using the DAQ Assistant.

   b. Stop the VI by selecting the **Stop button** on the LabVIEW front panel.

5. Select **File » Save** to save your changes.

6. Keep the VI open for the next exercise.

Part C Use an NI-DAQmx Task

Estimated time: 20 minutes

The lower-level NI-DAQmx VIs offer the greatest flexibility in programming NI LabVIEW data acquisition applications. With the lower-level VIs, you can programmatically control the lowest level of details to create everything from simple acquisition or generation applications to those with multiple simultaneous channels using complex timing and triggering.

As a compromise between the DAQ Assistant (which offers maximum ease of use while sacrificing some relative flexibility) and the lower-level NI-DAQmx VIs (which offer maximum flexibility while sacrificing some relative ease of use), the NI-DAQmx Task combines configuration-based setup with low-level control – a "best of both worlds" approach that works well for scalable applications.

1. Re-save the VI for use in Exercise 3.
   a. If it isn't already open, within the Light Sensor Project, double-click to select **Exercises » 2 – Use the NI-DAQmx VIs (Original).vi**.

   b. Select **File » Save As…**

   c. Choose **Substitute copy for original** and select **Continue…**

d. Save the VI in the <Exercises> directory as **3 – Use a NI-DAQmx Task.vi**.

2. If it isn't already open, select **Start » All Programs » National Instruments » Measurement & Automation Explorer** to open Measurement & Automation Explorer.

3. Add a NI-DAQmx Task to your Data Neighborhood.

   a. Underneath the My System tree, right-click on Data Neighborhood and select **Create New…**.

   b. Choose **NI-DAQmx Task** and select **Next**.

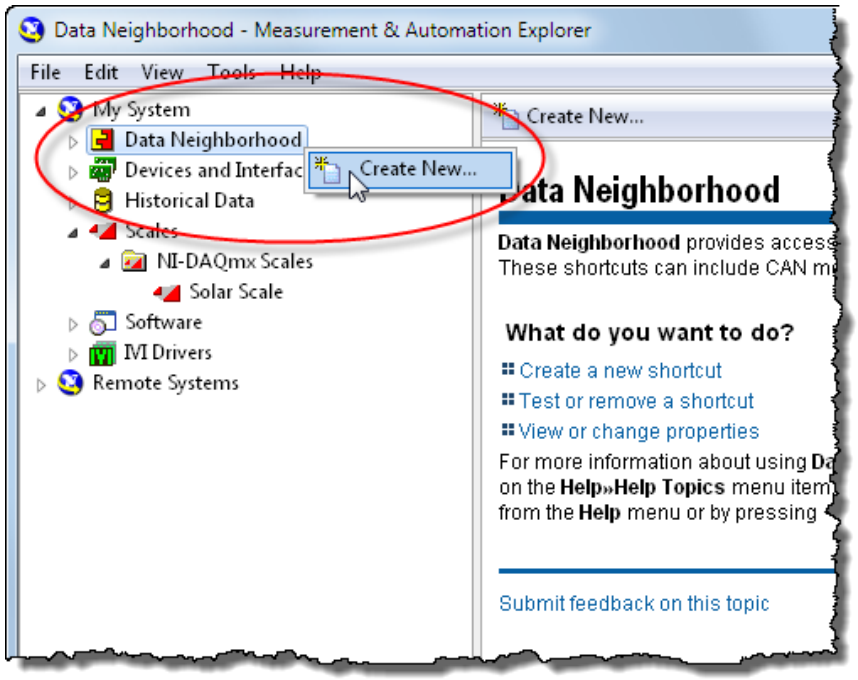


Figure 7. Use the Data Neighborhood section of the My System tree to create descriptively named shortcuts to physical channel configurations.

4. Add and configure a thermocouple analog input channel.

   a. Select **Acquire Signals » Analog Input » Temperature » Thermocouple** as shown in Figure 8.

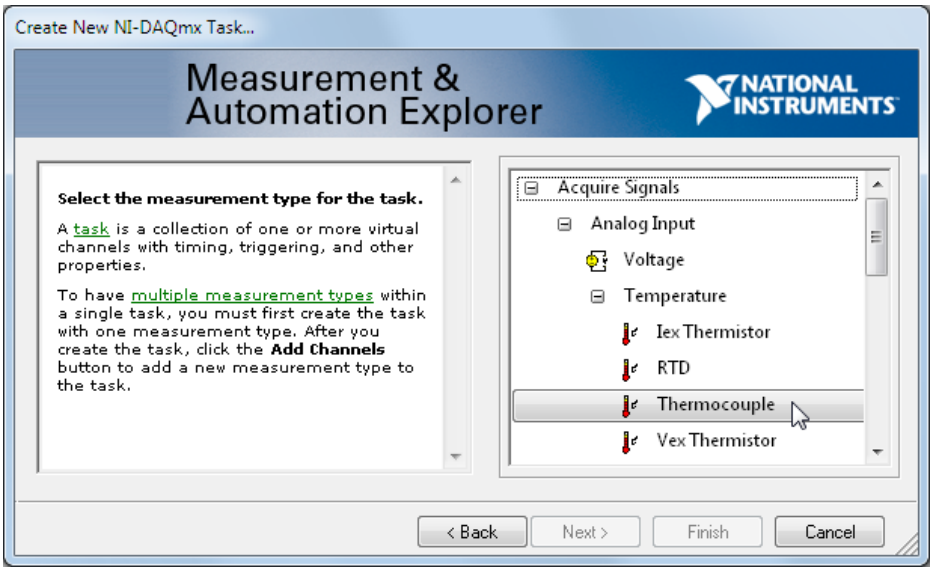


Figure 8. NI-DAQmx makes it easy to correctly configure acquisition applications since all configuration settings are grouped by acquisition type.

b. Choose **Temperature (NI 9213) » ai0** and select **Next** to configure the thermocouple input on analog input channel 0 of the NI 9213 C Series thermocouple module.

c. Rename the NI-DAQmx Task to *Measure Temperature and Light* and select **Finish**.

   Notice that the built-in configuration entry dialog that appears is identical in every way to the DAQ Assistant in LabVIEW. As a fully integrated driver, the user experience will be seamless whether configuring NI-DAQmx from LabVIEW, Measurement & Automation Explorer, or any other NI software. This contributes to both easier application flexibility and scalability.

d. Instead of **1 Sample (On Demand)**, change the Acquisition Mode setting to **Continuous Samples**.

e. Change the **Samples to Read** setting to **100**.

f. Change the **Rate (Hz)** setting to **1000**.

5. Add a second channel to the task to measure light.

   a. Within the Channel Settings section, use the **Add Channel** button ( ) to select **Voltage** as shown in Figure 9.

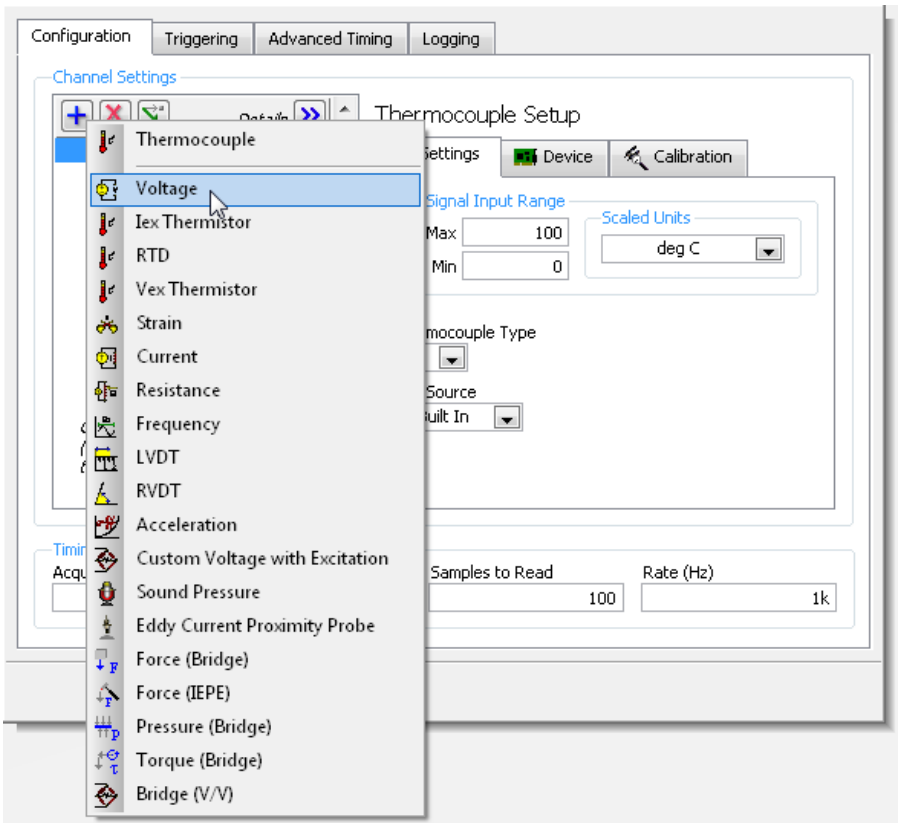


Figure 9. Both acquisition channels can be a part of the same NI-DAQmx Task since they both share the same timing and triggering settings.

a. Within the *Add Channels to Task* dialog, choose **Voltage_in (NI 9215) » ai0** and select **OK** to configure the voltage input on analog input channel 0 of the NI 9215 C Series analog input module.

b. Right-click on the Voltage channel and select **Rename…**

c. Rename the Voltage channel to *Solar Energy* and select **OK**.

6. Setup a NI-DAQmx Custom Scale.

a. Underneath the My System tree, right-click on **Scales** and select **Create New…**

b. Choose **NI-DAQmx Scale** and select **Next**.

c. Select **Map Ranges** as shown in Figure 10.

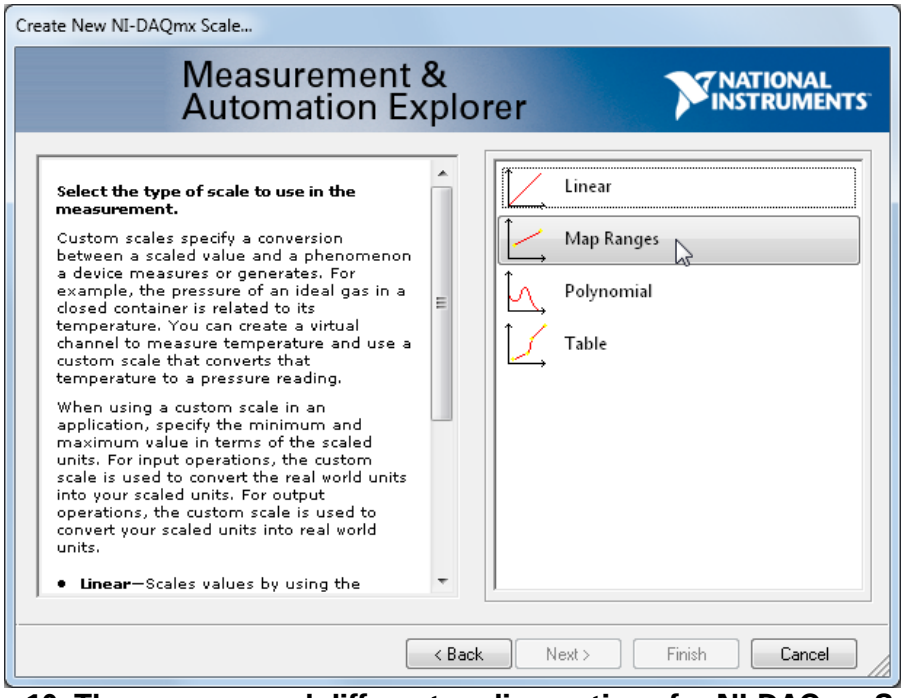


Figure 10. There are several different scaling options for NI-DAQmx Scales.

d. Name the scale *Solar Scale* and select **Finish**.

e. Configure the Map Ranges Scale dialog as shown in Figure 11.

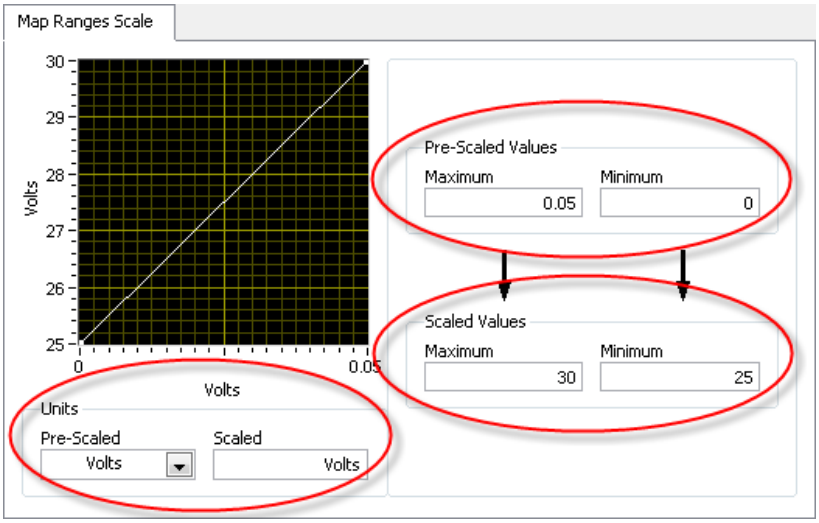


Figure 11. The NI-DAQmx Map Ranges scale maps one range of values to another range of values.

f. Save the changes to the NI-DAQmx Scale by selecting **Save**.

7. Apply the NI-DAQmx Scale to the Task.

a. Underneath the My System tree, select **NI-DAQmx Tasks » Measure Temperature and Light**.

b. Select the **Solar Energy** channel in the task.

c. Change the Custom Scaling setting to **Solar Scale**. If it does not appear in the list, you did not save your scale in this exercise during Part C, Step 6F!

d. Change the **Signal Input Range Min** setting to **25**.

e. Change the **Signal Input Range Max** setting to **30**.

f. Save the Measure Temperature and Light task by selecting **Save**. The Solar Energy channel should be configured as shown in Figure 12.

The specifications for the solar cell sensor dictate that it outputs a voltage value between 0 Volts (no light present) and .5 Volts (full sunlight). Light in the seminar room will output a tiny voltage value, so our scale will map it onto larger voltage values. Though the thermocouple also outputs values in this (small) voltage range, the thermocouple channel type in the task automatically maps the thermocouple output values to a degrees Celsius (°C) scale between 0 and 100.

Applying our custom scale to the solar measurements will map the solar values into the same range of values being represented by the thermocouple channel at approximately room temperature.

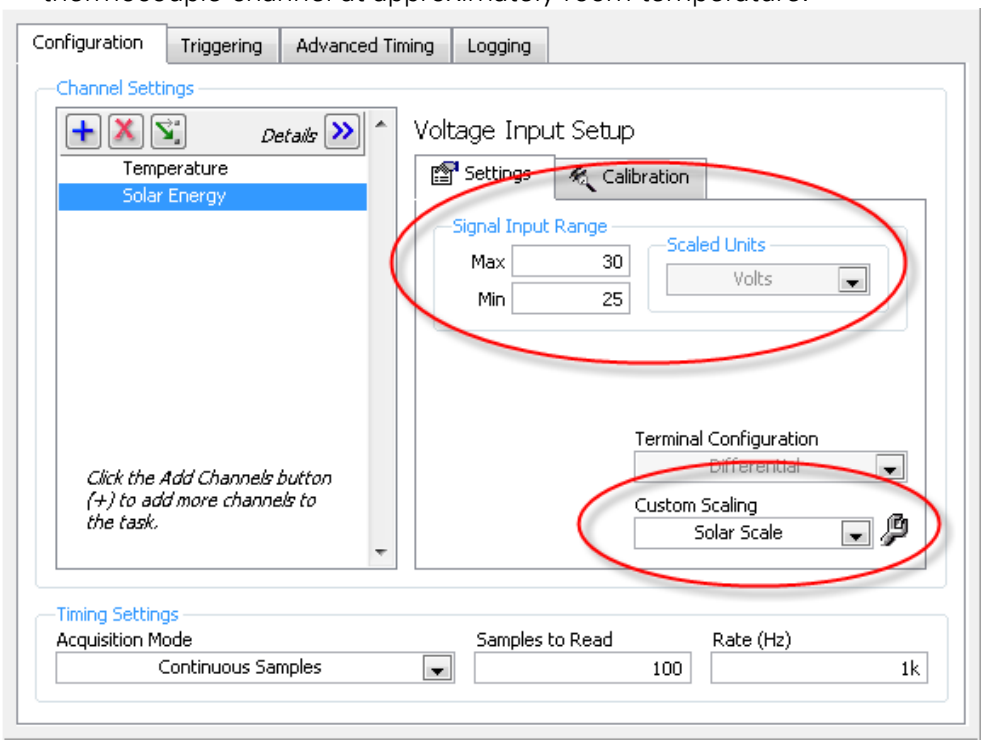


Figure 12. The Solar Energy channel uses its own scaling configuration but shares timing and triggering settings with all other channels in the same task.

8. Use the NI-DAQmx Task in the LabVIEW.

   a. In the **3 – Use a NI-DAQmx Task.vi** within LabVIEW, click-and-drag to highlight all VIs and input parameters to the left of the *DAQmx Start Task.vi* as shown in Figure 13.

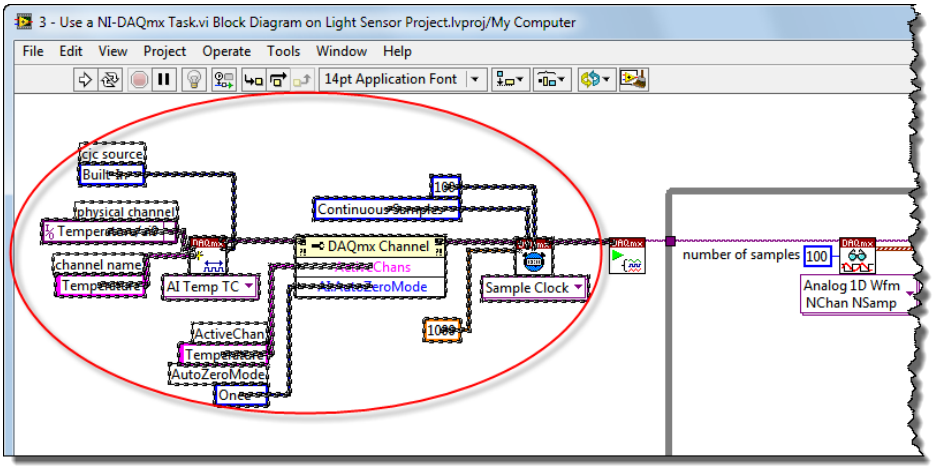


Figure 13. Delete all configuration VIs and input parameters to the left of the DAQmx Start Task.vi.

   b. Using your keyboard, press **Delete** to delete the code to the left of the DAQmx Start Task.vi.

   c. Press **<Ctrl+B>** to clear any broken wires resulting from the removal of the configuration code.

   d. Right-click in the white space on the block diagram to access the functions palette.

   e. Navigate to **Measurement I/O » NI-DAQmx** and select **DAQmx Task Name Constant**.

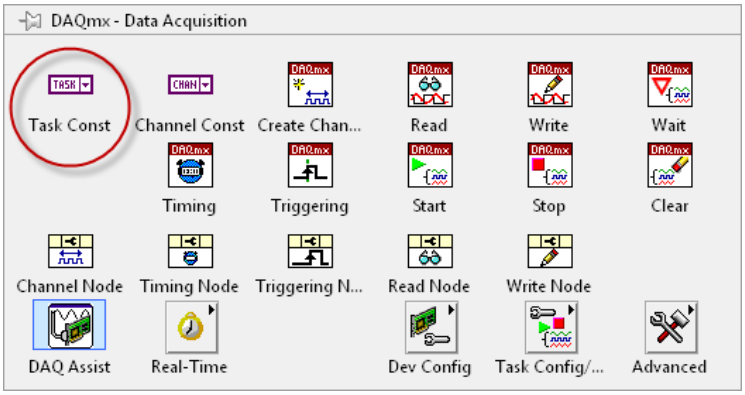


Figure 14. Select the DAQmx Task Name Constant from the DAQmx functions palette.

f. Left-click to place the DAQmx Task Name Constant to the left of the DAQmx Start Task VI.

g. Using the drop-down, select the **Measure Temperature and Light** task that you configured in Measurement & Automation Explorer.

h. Wire the task from the output of the **DAQmx Task Name Constant** to the input of the **Task / Channels In** input of the DAQmx Start Task.vi as shown in Figure 15.

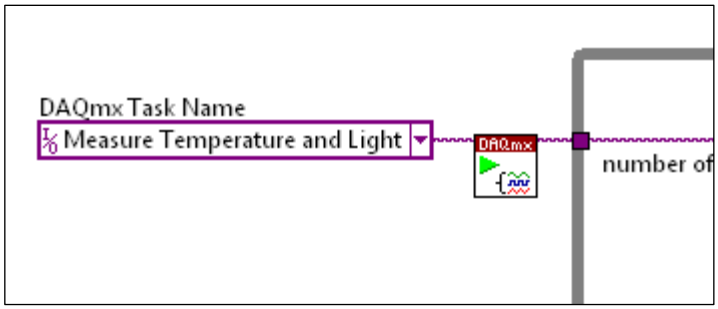


Figure 15. Replace the configuration VIs and input parameters with the DAQmx Task Name Constant.

i. Right-click on the wire exiting the **Result** output of the Comparison Express VI.

j. From the right-click context menu, select **Insert » Signal Manipulation Palette » Split Signals** as shown in Figure 16.

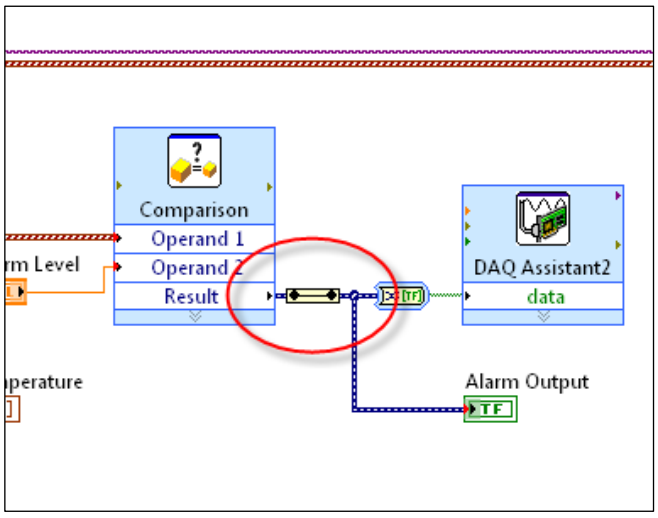


Figure 16. Insert the Split Signals node immediately after the Comparison Express VI.

Due to the tight integration between LabVIEW, hardware, and the NI-DAQmx driver, LabVIEW is aware of the creation or configuration of tasks or settings in

Measurement & Automation Explorer. Any tasks created in Measurement & Automation Explorer automatically populate within LabVIEW.

Because the Measure Temperature and Light task includes two channels, we must split the signal coming out of the Comparison VI so that only the results of the Temperature channel are output to the digital output DAQ Assistant.

9. Run the VI.
    a. Using the **Run** arrow (), run the VI.

    b. While the application is running, use your finger to warm the thermocouple and verify that the temperature rises appropriately.

    c. While the application is running, use your hand to shield the solar cell sensor from light and verify that the voltage falls appropriately. Alternatively, you can provide additional light by using an available cell phone and verifying that the voltage rises appropriately. The solar cell should not react as drastically as the thermocouple.

    d. Stop the VI by selecting the **Stop button** on the LabVIEW front panel.

10. Save the VI.

11. Keep the VI open for the next exercise.

Part D Save a Well-Documented File

Estimated time: 20 minutes

The project already writes to an ASCII file in the form of a LabVIEW Measurement (*.LVM) file. ASCII files are reasonable choices for small, one-shot applications, but the ASCII file format is not a viable data storage choice for applications that require scalability due to the large amount of overheard required using ASCII format.

This exercise will replace the ASCII format with a more flexible, scalable file format – the Technical Data Management Streaming (TDMS) file format – which will also yield future benefits for data management and mining.

1. Re-save the VI for use in Exercise 4.

    e. If it isn't already open, within the Light Sensor Project, double-click to select **Exercises » 3 – Use a NI-DAQmx Task.vi**.

    f. Select **File » Save As…**

    g. Choose **Substitute copy for original** and select **Continue…**

    h. Save the VI in the <Exercises> directory as **4 – Save a Well Documented File**.

2. Configure the Write to Measurement File format options.

    a. On the block diagram, double-click the *Write to Measurement File Express VI* to open its configuration dialog.

    b. In the File Format section, select **Binary (TDMS)**.

    c. Make a mental note of the directory in the Filename path directory.

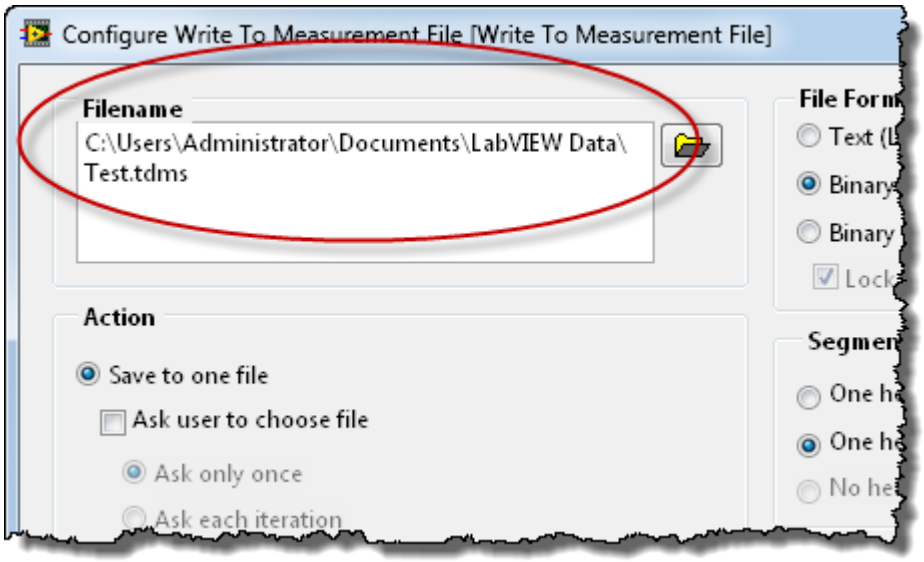


Figure 17. The new TDMS data files should be saved in the LabVIEW Data directory.

    d. In the *If a File Already Exists* section, select **Use Next Available Filename**.

    e. Provide an explanatory description of the file's purpose in the *File Description* field.

    f. Ensure that the dialog is configured similar to the settings in Figure 18.

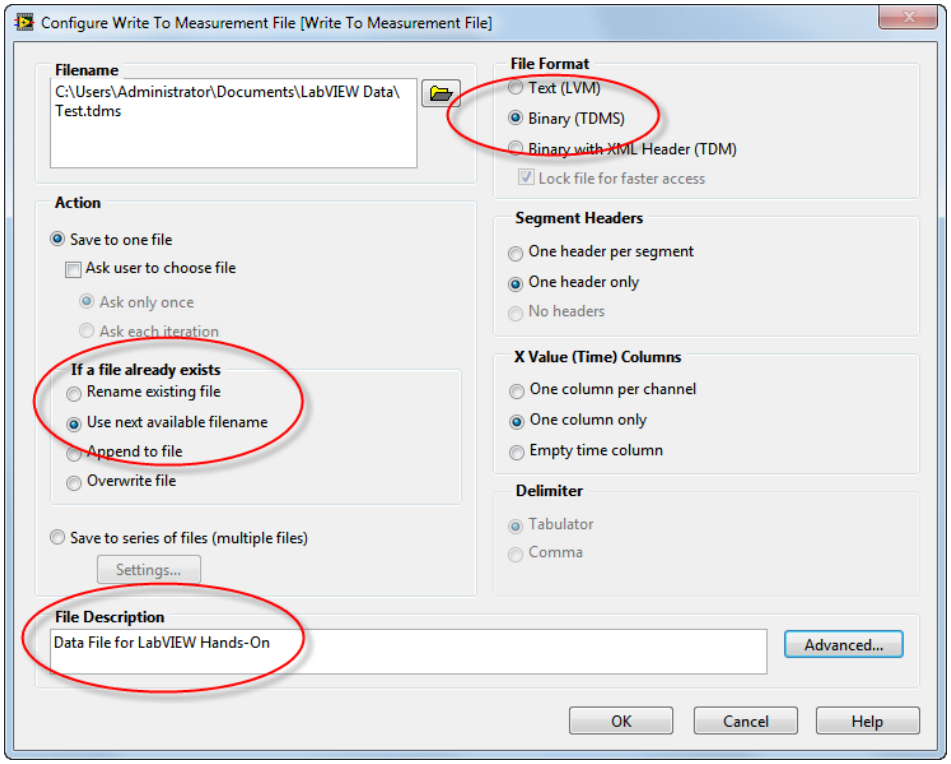


Figure 18. The Write to Measurement File Express VI provides maximum ease of use but, like all Express VIs, sacrifices some flexibility.

3. Write custom, descriptive properties to the structure of the TDMS file.

    a. In the Configure Write to Measurement File configuration dialog, select **Advanced…**.

    b. On the TDM Properties tab, select **Insert**.

    c. In the Hierarchy Level column, select **Channel Group**.

    d. In the Data Type column, select **STR** (string).

    e. In the Name column, replace the text Untitled_0 with the text **Author**.

    f. In the Value column for the Author property, type **your name** (i.e. Jane Smith).

    g. **Repeat steps 3B – 3F as desired** to document more custom properties. Feel free to experiment! With the TDMS file format, you can document an unlimited number of properties, and the more you add, the more flexibility you gain later when trying to search for or manage your data.

    Some optional properties to include are Location, Test Type, and Serial Number. You can populate these properties with made up values.

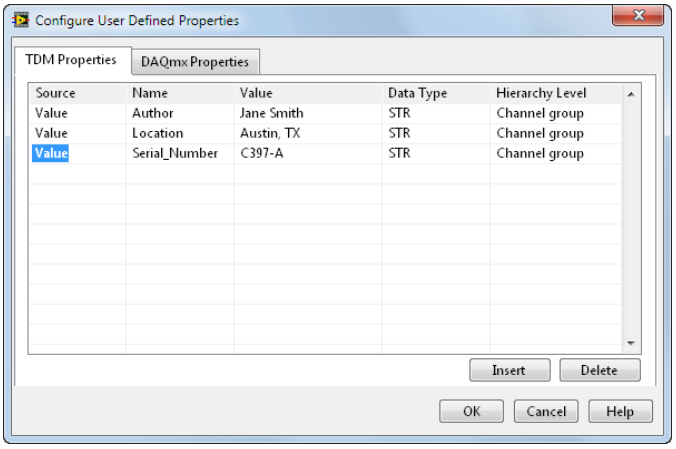


Figure 19. With the TDMS format, you can easily add unlimited descriptive information to the file. Feel free to experiment!

4. Add NI-DAQmx Properties to the TDMS file.
   a. In the Configure User Defined Properties dialog, select the **DAQmx Properties tab**.

   b. Check the box for **Analog Input**.

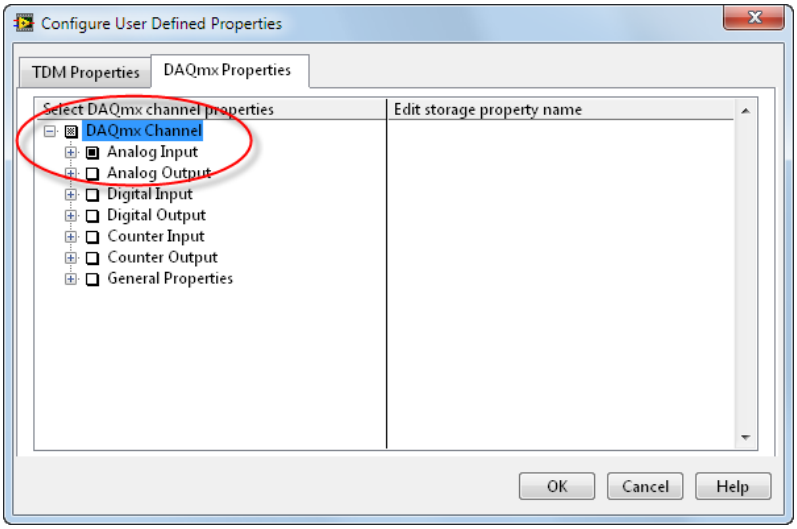


Figure 20. With the NI-DAQmx driver, you can automatically log all acquisition settings to TDMS properties for future reference.

   c. Select **OK** to close the Configure User Defined Properties dialog box.

   d. Select **OK** to close the Configure Write to Measurement File dialog box.

5. Connect the NI-DAQmx Task to the Write to Measurement File Express VI.

   a. Wire the purple **Task Out** output of the DAQmx Read.vi to the **DAQmx Task** input of the Write to Measurement File Express VI as shown in Figure 20.

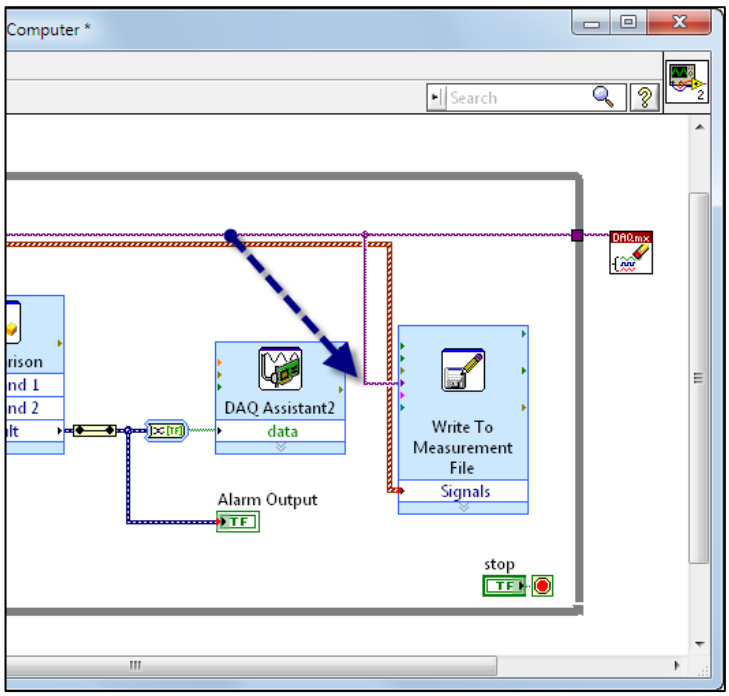


Figure 21. The purple DAQmx Task wire contains all data acquisition settings from the Measure Temperature and Light task.

By configuring the Write to Measurement File Express VI to write all Analog Input properties from the task, the NI-DAQmx driver automatically saves any Analog Input properties utilized in the measurement with the data file. These properties include descriptive settings such as custom scale information, measurement units, or thermocouple type – all very helpful information to have automatically documented!

6. Prompt the user for a test description.
    a. Right-click in the white space on the block diagram to access the functions palette.

    b. Navigate to **Programming » Dialog & User Interface** and select the **Prompt User for Input** Express VI.

    c. Left-click to place the Prompt User for Input Express VI outside of the while loop somewhere to the left of the while loop as shown in Figure 21.

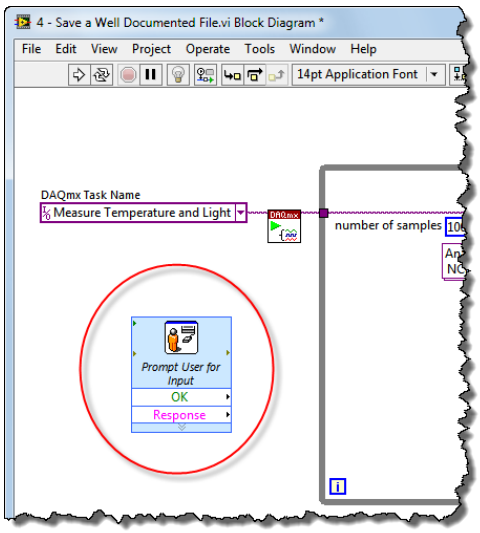


Figure 22. The Prompt User for Input Express VI is placed before the while loop since it is the first thing we want to happen when the application runs.

d. In the Configure Prompt User for Input dialog that appears, type *Please Provide a Test Name:* in the **Message to Display** field.

e. In the Inputs entry section, type **Response** in the Input Name column.

f. In the Inputs entry section, select **Text Entry Box** for the Input Data Type of the Response entry.

g. Confirm that the dialog is configured to match Figure 22 and then select **OK** to close the dialog.

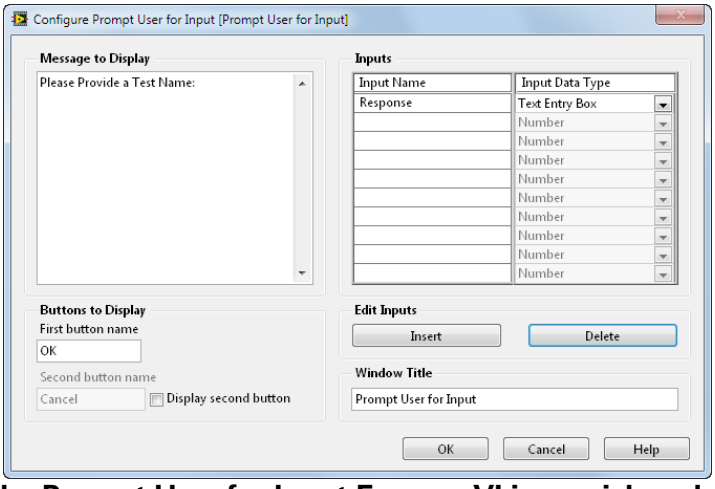


Figure 23. The Prompt User for Input Express VI is a quick and easy way to get feedback from the user.

h. Wire the pink **Response** output of the Prompt User for Input Express VI through the while loop to the pink **Comment** input of the Write to Measurement File Express VI as shown in Figure 23.

i. Wire the yellow **Error Out** output of the Prompt User for Input Express VI to the **Error In** input terminal of the DAQmx Start Task.vi as shown in Figure 23.

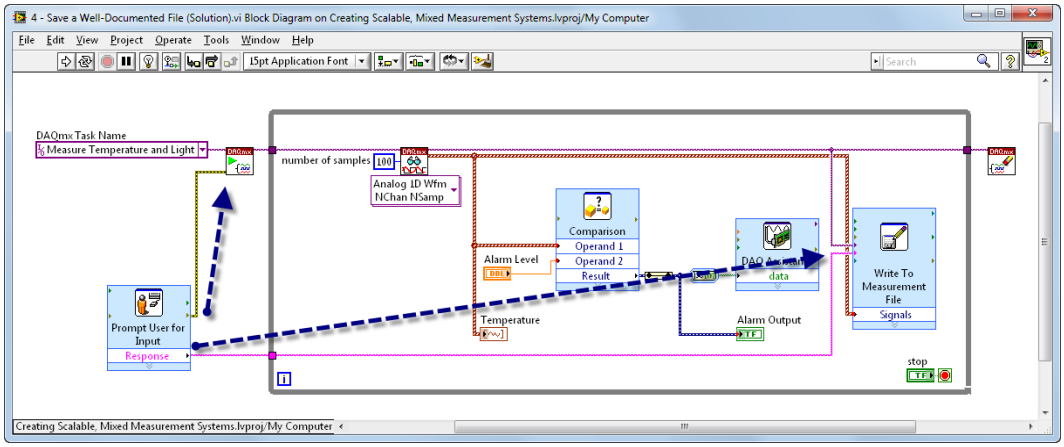


Figure 24. The Write to Measurement File takes input from multiple sources.

By wiring the Error Out output of the Prompt User for Input Express VI to the Error In input terminal of the DAQmx Start Task VI, we ensure that the acquisition is not started until the user is finished responding to our prompt. By enforcing an order of execution in this way, we are using a differentiating feature of LabVIEW called dataflow – we literally get to control the flow and timing of data.

7. Run the VI.
   a. Using the **Run** arrow (), run the VI.

   b. In the Prompt User for Input dialog, type any unique name for the test such as "**Test 001-A**."

   c. While the application is running, use your finger to warm the thermocouple; use your hand or cell phone to vary the amount of light reaching the solar cell.

   d. Stop the VI by selecting the **Stop button** on the LabVIEW front panel.

   e. **Repeat steps 7A – 7D as desired** to create a few varying data sets. Some options include:
      i. During one test, interact only with the thermocouple.
      ii. During one test, interact only with the solar cell.
      iii. During one test, interact with both sensors, starting at the same time.
      iv. During one test, interact with both sensors, starting at different times.

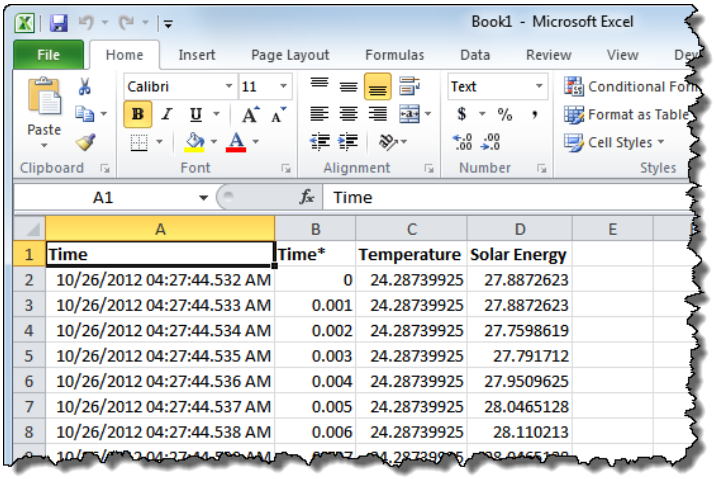
8. Save the VI.

9. Using Windows Explorer, navigate on disk to the location where the data was saved. If you forgot where the data was saved, you can double-click the Write to Measurement File Express VI on the block diagram and examine the Filename control.

13. Right-click on one of your data files in Windows Explorer and select **Open With » Excel Importer** to open it with Microsoft Excel. Review the header information on the first Excel sheet (tab) and notice all of your custom descriptive properties.

14. Review the properties that were added to the file by the DAQmx driver, such as the name of the custom scale and the measurement type.

15. Switch to the second Excel sheet (tab) in the file and notice the time, temperature, and solar energy data saved in the file.



	A	B	C	D	E
1	Time	Time*	Temperature	Solar Energy	
2	10/26/2012 04:27:44.532 AM	0	24.28739925	27.8872623	
3	10/26/2012 04:27:44.533 AM	0.001	24.28739925	27.8872623	
4	10/26/2012 04:27:44.534 AM	0.002	24.28739925	27.7598619	
5	10/26/2012 04:27:44.535 AM	0.003	24.28739925	27.791712	
6	10/26/2012 04:27:44.536 AM	0.004	24.28739925	27.9509625	
7	10/26/2012 04:27:44.537 AM	0.005	24.28739925	28.0465128	
8	10/26/2012 04:27:44.538 AM	0.006	24.28739925	28.110213	

Figure 25. The fully documented file contains metadata and measurement data.

16. Close the data file, if it is still open.

17. Close the LabVIEW VI, if it is still open.

18. Close the Light Sensor Project, if it is still open.

<End of Exercise>

Strain Gage Measurement with User LED Feedback

Goals

- In the first part of this exercise, you will write a program to acquire data from a strain gage in a quarter bridge configuration.
- For the second part, you will add code to indicate approximate strain to a user with LED indicators
- Key concepts include
  - Strain measurement
  - Using the NI-DAQmx API
  - Using SubVIs
  - Combining measurement and control tasks
  - Writing to file when an event occurs

Part A Measuring Data From a Strain Gage

Estimated time: 40 minutes

Acquiring strain data is one of the most common tasks in any structural measurement system. Strain is measured using a strain gage, which is a small resistive element that changes resistance when deflected. Because the resistance changes proportionally with deflection, you could excite the strain gage with a current and measure the voltage change to calculate the change in resistance. However, because the change in resistance is so small, a better solution is to use a Wheatstone bridge configuration to measure the small changes in resistance.

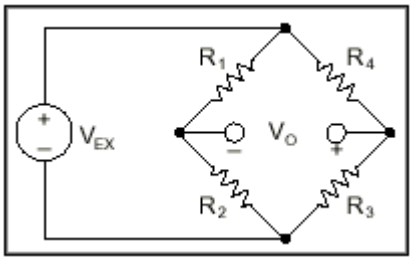


Figure 1. A Wheatstone Bridge enables precise measurement of small resistance changes.

In a Wheatstone Bridge, four resistors are used in the configuration seen in Figure 1 and excited using a voltage source V_{ex}. If all four resistors are equal, the bridge is said to be balanced and the voltage measured at V_o will be equal to zero. When using a Wheatstone bridge to measure strain, the strain gage will have a nominal resistance equal to the other elements of the bridge, in the case of this exercise, 350 Ohms. As the strain gage flexes, the resistance changes and, consequently, V_o. Knowing this, we can derive the proportional change (without getting into any details), arriving at the following equation:

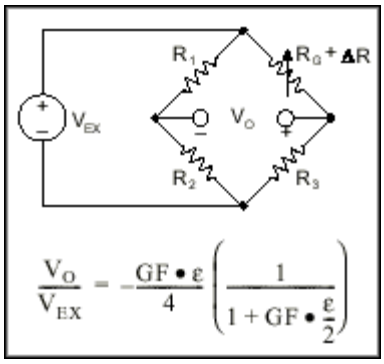


Figure 2. In a quarter bridge configuration, we can calculate strain based on the voltage change.

In this exercise, you will use a strain gage in the quarter bridge configuration, meaning that only one of the gages is active.

Strain gages are particularly delicate. Before you start this exercise, confirm that the strain gage is properly connected and you are able to acquire basic data from the channel connected to the strain gage.

1. On the NI CompactDAQ chassis, ensure that the green **Power** and amber **Ready** LEDs are lit to confirm that the chassis is connected over USB and powered on.

2. Examine the wiring into the NI 9236 module to confirm that the strain gage is properly connected.

3. If it isn't already open, launch Measurement & Automation Explorer by selecting **Start » All Programs » National Instruments » Measurement & Automation Explorer**.

4. Right-click on the NI 9236 and select **Test Panels…**.

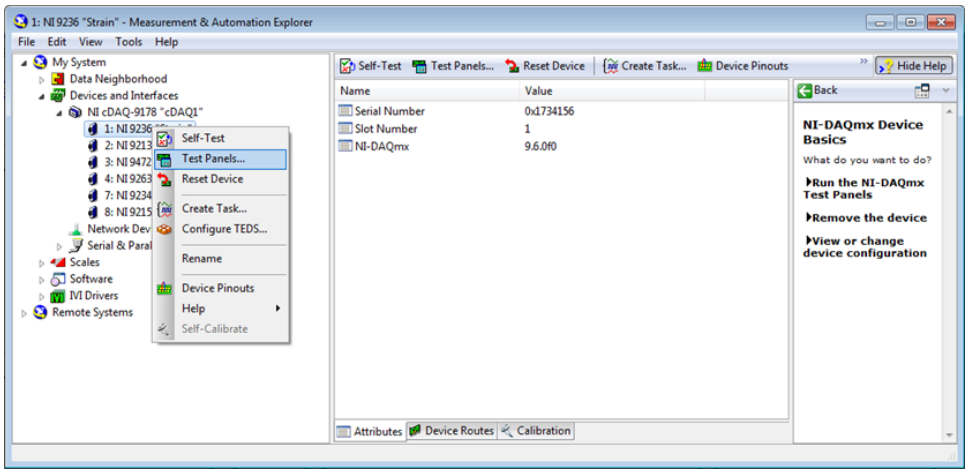


Figure 3. Select Test Panels to open a simple utility for checking signal connectivity.

5. When the test panel opens, change the **Samples to Read** input to 500 and the **Mode** input to Continuous.

6. Press **Start**. The test panel should look like Figure 3.

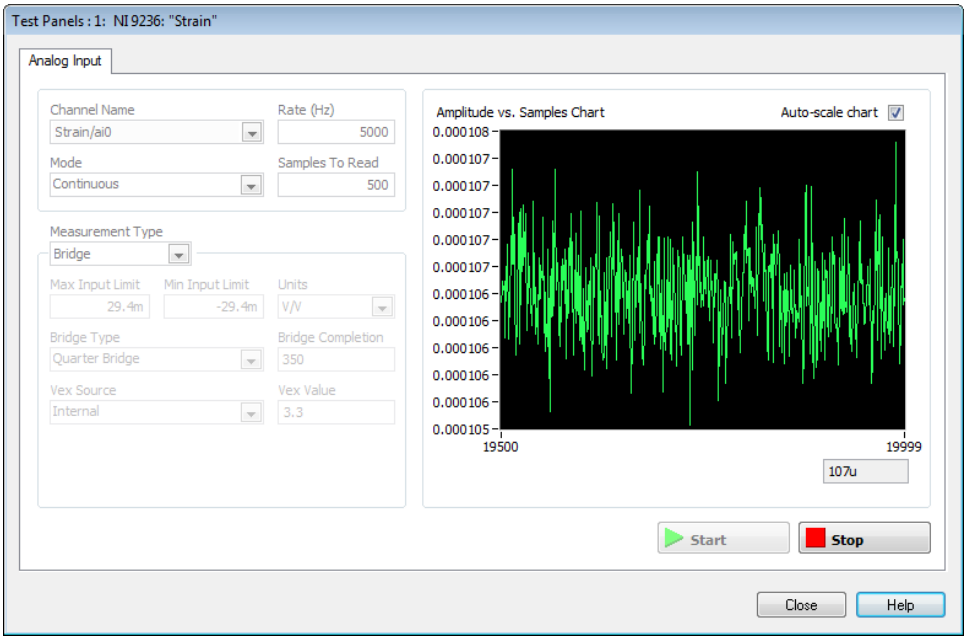


Figure 4. The test panel is used to confirm electrical connections.

7. Press down on the strain bar and ensure that the graph reacts to your input. If you detect no noticeable change, please alert your instructor at this time.

8. Press the **Stop** button and close the Test Panel.

   Every time you set up a new measurement or measurement system, it is a good idea to confirm that the wiring is correct and that all software is installed and working correctly. MAX provides the insight into your system and setup to help you eliminate mistakes early in your development process.

9. Once you have confirmed that the strain gage is working, minimize MAX and select **Start » All Programs » National Instruments » LabVIEW <Year> » LabVIEW** to open LabVIEW.

10. From the LabVIEW Getting Started window, select **Open Existing**.

11. Navigate to **C:\Seminars\LV HO\Exercises\5 – Strain**.

12. Double-click **Strain Gage Exploration.lvproj** to open the strain gage project. You should see the following screen:

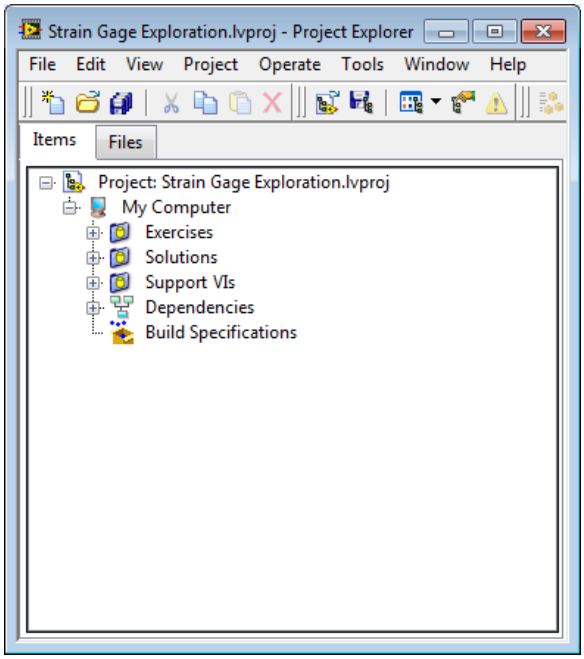


Figure 5. Use LabVIEW Projects to organize all your VIs.

With the LabVIEW Project you can easily organize and visualize all the files that are important to your application. In this particular project, the folders like Exercises, Solutions, and Support VIs have been set to auto-populate with the files within them.

13. Expand the folders to view the files within them. You should see the following VIs:

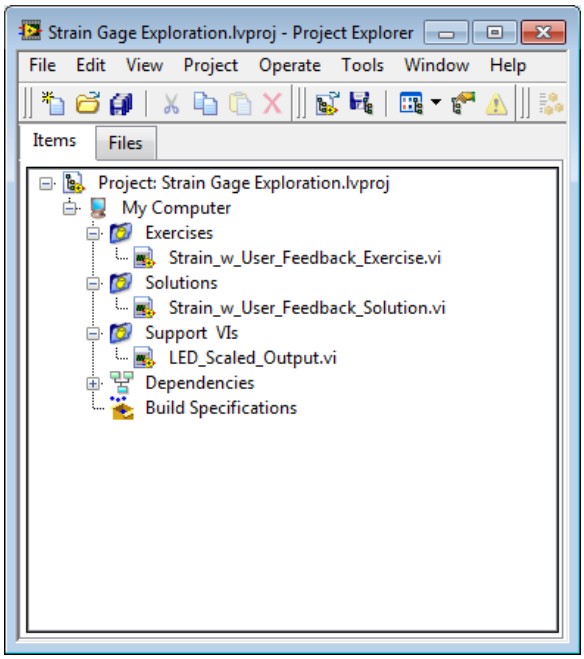


Figure 6. The VIs used in your project can be found in the auto-populating folders.

14. Start by opening the solution. In the Project Explorer, expand **My Computer » Solutions** and double-click **Strain_w_User_Feedback_Solution.vi**.

15. Click the **Run** arrow ( ) and then **delicately** push down on the strain bar. Notice that the LEDs signal the amount of strain experienced on the strain bar.

16. Close the VI. If prompted, do not save changes.

17. To start this exercise, use the Project Explorer to navigate to **My Computer » Exercises** and double-click **Strain_w_User_Feedback_Exercise.vi**. The front panel has already been created for you.

    When starting a new program, beginning with a Front Panel design is a good way to organize your thoughts around the inputs and outputs that you expect. Once you understand how you want a user to interact with the code, then you can begin writing the VI to support that functionality.

    Throughout this exercise, you will complete this VI to acquire data from the strain gage and light a series of LEDs to communicate the strain level to a user.

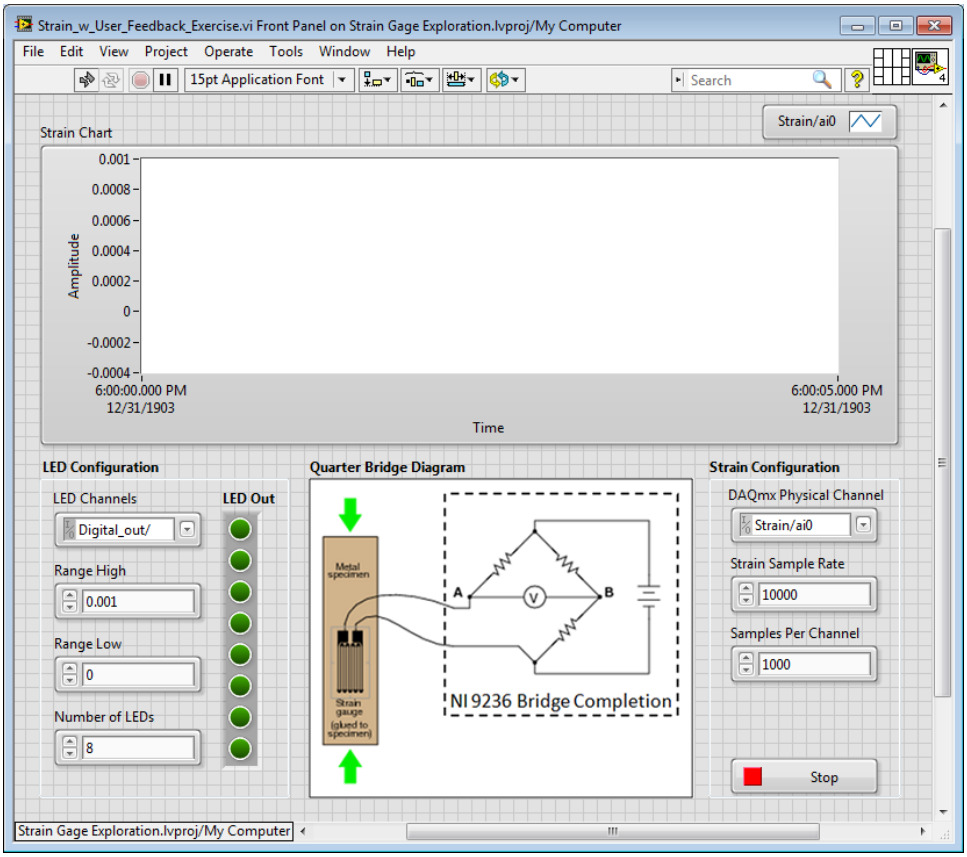


Figure 7. The Front Panel is the user interface for your program and contains all the controls and indicators to interact with your code.

Notice that the Run arrow in the top right corner is broken ( ). This is because

the code contains errors, or in this case, is not in a running state. LabVIEW continuously compiles in the editing environment, so you will always know if your code is able to run.

Pressing the run arrow when it is broken will show a list of errors. Try pressing the run arrow to view some of the errors with our current code. If you double-click on one of the errors, LabVIEW will even highlight the section of code containing the error.

If your code compiles with no errors, the run arrow will no longer be broken ( ).

18. Close the Error Dialog and select **Window » Show Block Diagram**. The keyboard shortcut for this is **<Ctrl+E>**. Use this shortcut to toggle back and forth between the Front Panel and Block Diagram. When the Block Diagram opens, you will see the following code:

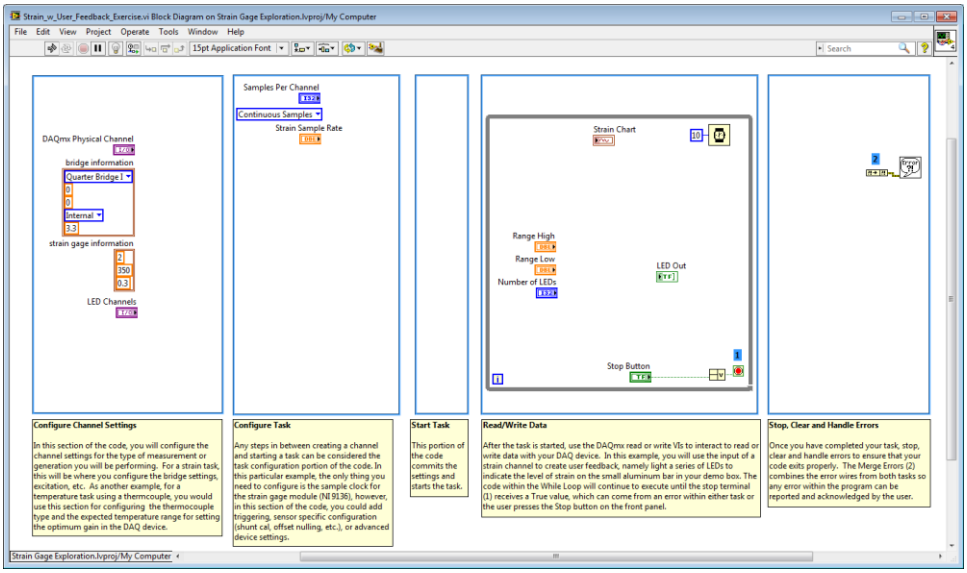


Figure 8. The Block Diagram does the work of your program. This is where you will write the code to control the Front Panel.

Because the code has been started, this Block Diagram includes some basic elements already. Each element on the Front Panel has a corresponding terminal on the Block Diagram. To find the corresponding Front Panel element, you can double-click the element on the block diagram to highlight it.

Try double-clicking on the **Strain Chart** icon. Then double-click on the same item on the Front Panel. This is a convenient way to find items on either the Front Panel or Block Diagram.

The light yellow boxes are annotations. You can create notes on the block diagram simply by double-clicking in any blank space. It is always a good idea to properly document your code so that you (if you come back a week later) or someone else can understand what is going on.

19. Now that you are familiar with the elements of the program, you can start building the code to acquire strain data. To start, make sure that the Block Diagram is visible.

20. For this exercise, a large part of the coding will come from the DAQmx palette. To access this palette right-click on any white space on the Block Diagram and navigate to **Measurement I/O » NI-DAQmx**, like in Figure 9.

21. Click on the **pin** to keep this palette visible (red arrow in Figure 9).

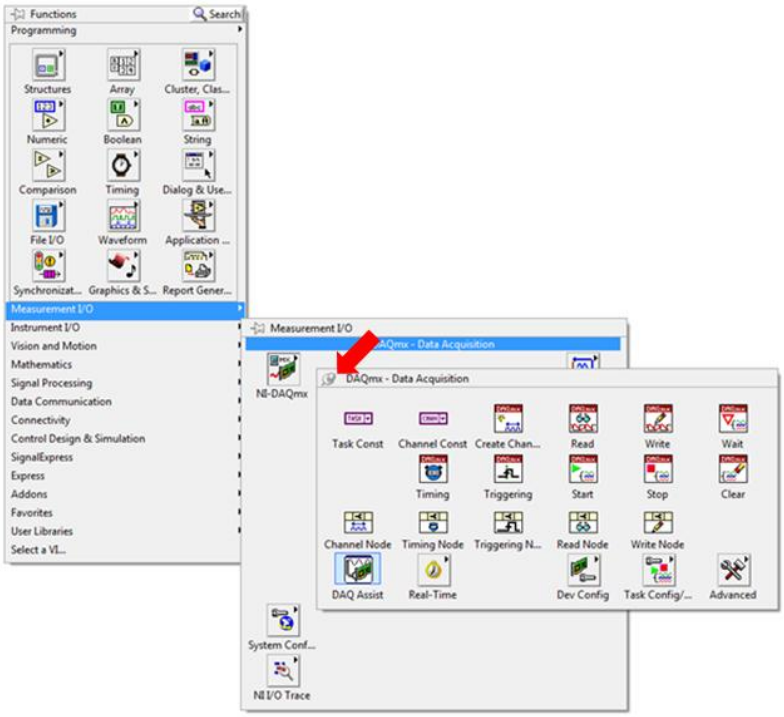


Figure 9. The DAQmx palette includes all the required VIs for communicating with your NI CompactDAQ chassis.

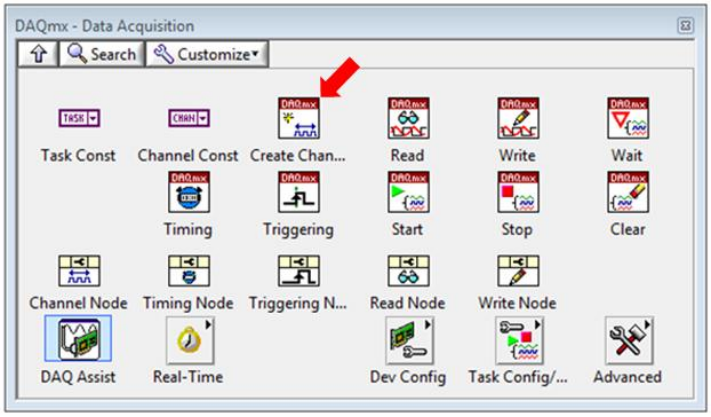


Figure 10. Pinning a palette enables you to click elsewhere in your code with the palette still visible.

22. To start, drag the **NI-DAQmx Create Channel** (red arrow in Figure 10) and place it in the position shown in Figure 11.

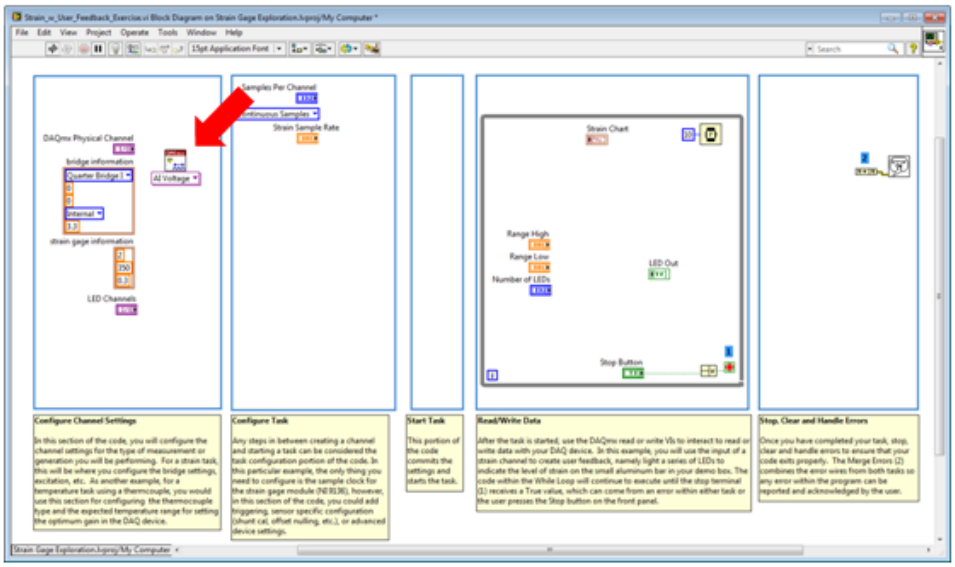


Figure 11. The DAQmx Create Channel VI establishes communication with your DAQ device.

23. By default the VI is configured to create a simple analog input voltage. To configure this for strain, click on the drop down arrow and select **Analog Input » Strain » Strain Gage**.

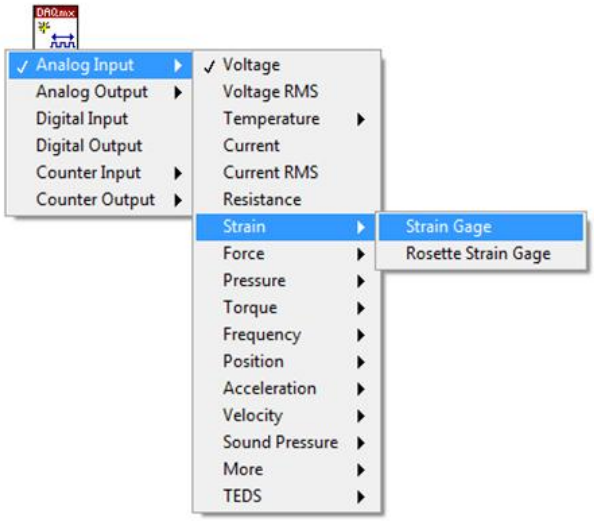


Figure 12. The DAQmx Create Channel VI is called polymorphic because it changes to adapt to the selected measurement type.

24. To find out more about this VI, bring up the Context Help by pressing **<Ctrl+H>**.

    The Context Help gives you a brief description of anything you hover over with your cursor. Hover over the *DAQmx Create Channel VI* to see the inputs and outputs of the VI.

For a strain channel, we need to provide the channels you are measuring and the strain gage information. To provide these inputs, wire up the VI as shown in Figure 13 by clicking on the *output* of each block diagram node and then clicking again on the target input terminal of the DAQmx Create Channel VI.

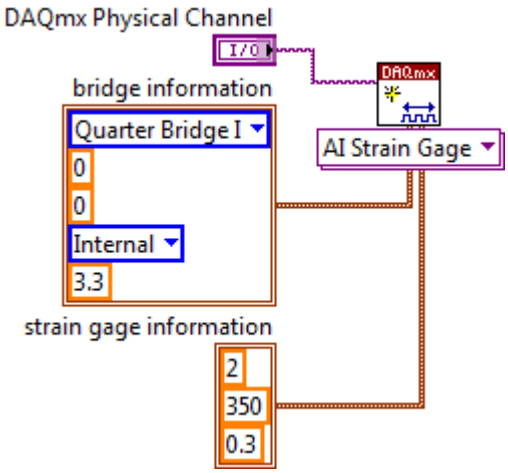


Figure 13. Wire up the inputs to the DAQmx Create Channel VI.

25. Drag the following VIs from the DAQmx palette and place them as shown in Figure 14.
    a) DAQmx Timing
    b) DAQmx Start Task
    c) DAQmx Read
    d) DAQmx Stop Task
    e) DAQmx Clear Task

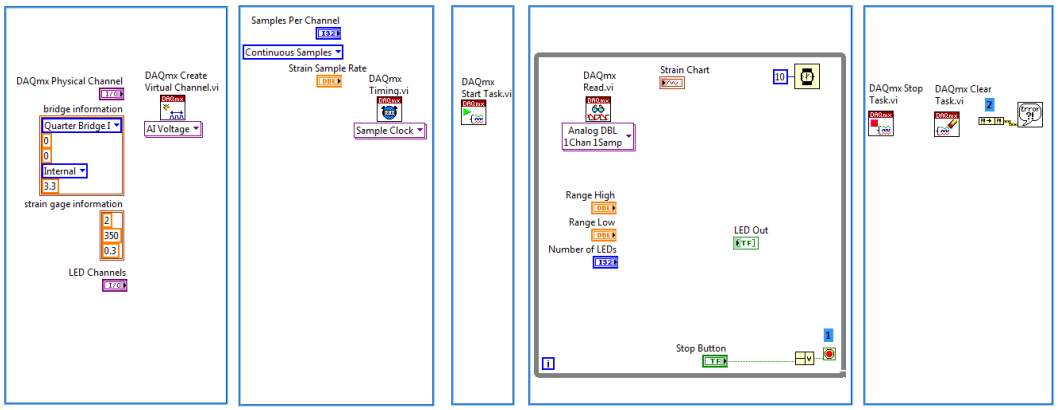


Figure 14. The flow of DAQmx code almost always follows the same pattern.

If you recall from the example programs you examined in the earlier exercises, the DAQmx pattern almost always follows the same flow. A channel is created, settings like timing and triggering are configured, the task is started, the channel is read then the task is stopped and the channel is cleared.

26. The *DAQmx Timing VI* configures the sample rate, sample mode, and clock source

for your task. You are going to be sampling continuously, using the on-board clock source. Wire the DAQmx Timing VI as follows:

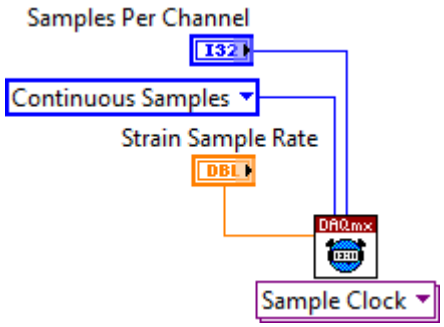


Figure 15. The DAQmx Timing VI configures your clock parameters.

27. The *DAQmx Start Task VI* transitions the code to the running state. Once your code is past this VI, you are ready to read or write data. To learn more about this VI, make sure the Context Help is still visible **<Ctrl+H>** and hover over the DAQmx Start VI.

28. After the task is started, the *DAQmx Read VI* pulls data off the DAQ device buffer. Like the DAQmx Create Channel VI, the DAQmx Read VI is polymorphic. To configure this VI to read a data from the strain gage, click the drop-down arrow and select **Analog » Single Channel » Multiple Samples » Waveform**.

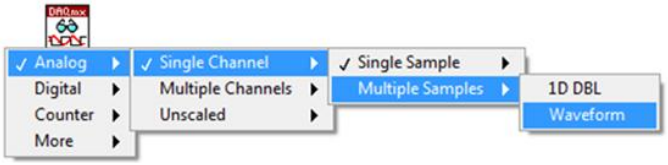


Figure 16. Select a single channel of waveform measurement to read from the supplied strain gage.

29. Wire the output of the *DAQmx Read VI* to the Strain Chart. This will enable a user to visualize the data on the front panel.

30. The *DAQmx Stop VI* and *DAQmx Clear VI* ensure that your program properly closes communication and frees up the hardware before stopping the code. Again, to learn more about these VIs, hover over them with the Context Help visible.

31. Wire the task and error wires through the code as shown in Figure 17. This will pass the task values and any errors that may arise through the code.

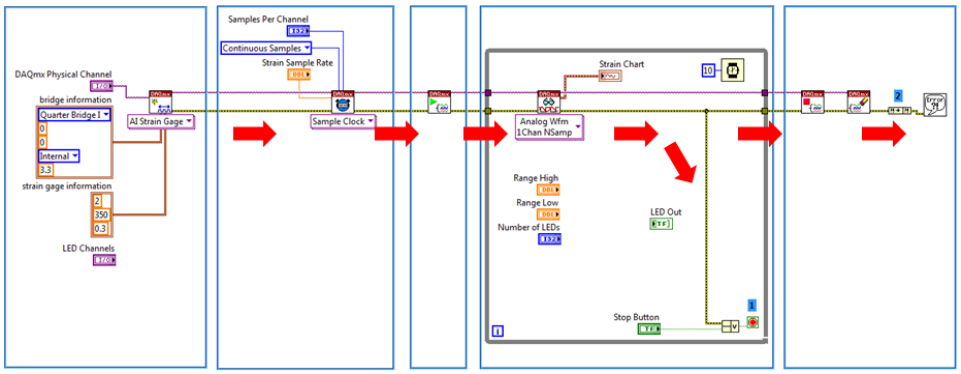


Figure 17. To finish the strain acquisition code, wire the task and error wires through the code.

Because LabVIEW uses dataflow to enforce order of execution, you can visualize the data traveling across each of the wires, from one VI to the next. Each VI will only execute once all the wired inputs have been accounted for.

Wiring the error wire thought the code ensures that any errors in the code will pass through the code and be reported to the user.

32. Wire the error wire from the DAQmx Read VI to the input of the Compound Arithmetic VI. This VI will OR the inputs to stopping the While Loop from executing. The code you just wrote will stop if an error occurs or a user stops presses the stop button.

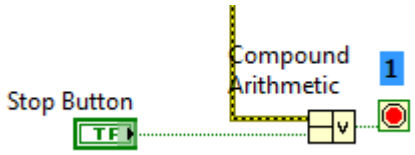


Figure 18. The Compound Arithmetic ORs the inputs, stopping the While Loop if the code has an error or the user presses stop.

33. Once you wire the error wire to the Compound Arithmetic, your code should be ready to run. Check to see that the Run Arrow is intact ( ).

34. Press **<Ctrl+E>** to toggle to the Front Panel.

35. Press the drop-down arrow on the *DAQmx Physical Channel* to select the correct channel. In this exercise, the strain gage should be attached to channel 0 of the NI 9236. If you correctly renamed your modules in an earlier exercise, select **Strain/ai0**.

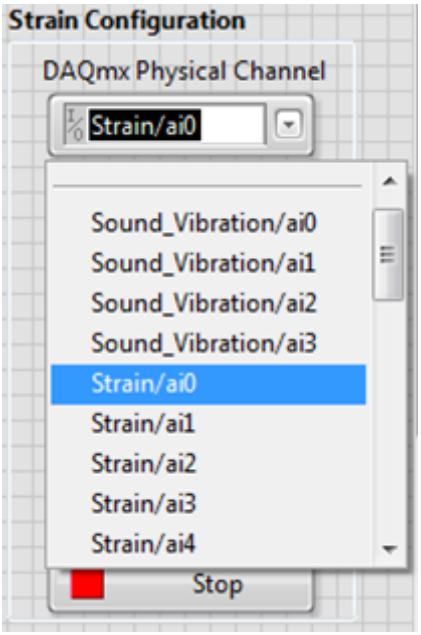


Figure 19. Select the channel that your strain gage is attached to.

36. Set the *Strain Sample Rate* to 10000 Hz and the *Samples Per Channel* to 1000, like in Figure 20.

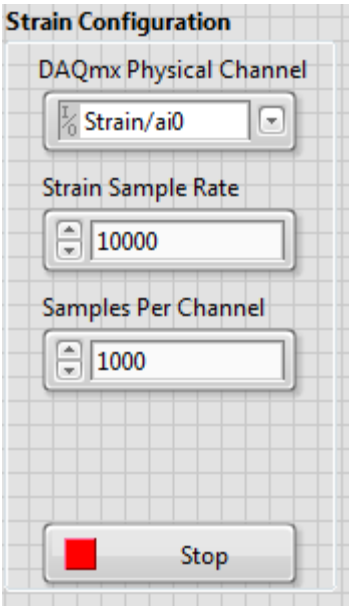


Figure 20. A good rule of thumb is to set the number of samples to 1/10th of the sample rate.

The sample rate dictates how fast your DAQ device samples the selected channel. The Samples Per Channel dictates how much data is pulled from the DAQ buffer each time the loop runs. In the example above, DAQmx Read will wait until there are 1000 samples on the buffer before it pulls them from the DAQ device. This translates to an update every 1/10th of a second. If you change the Samples Per Channel to 10000, your code will update once every second. If you get buffer

overflow errors, it usually means that your loop is not executing fast enough to pull data off the DAQ device before the buffer is full. Try increasing the number of Samples Per Channel if this is the case. In some cases, you may need to split your code into parallel loops. For more information about this type of architecture, search *Producer Consumer* on ni.com.

37. Press the Run arrow. Notice that when you **gently** press on the strain gage, you can see the inflections on the waveform chart.

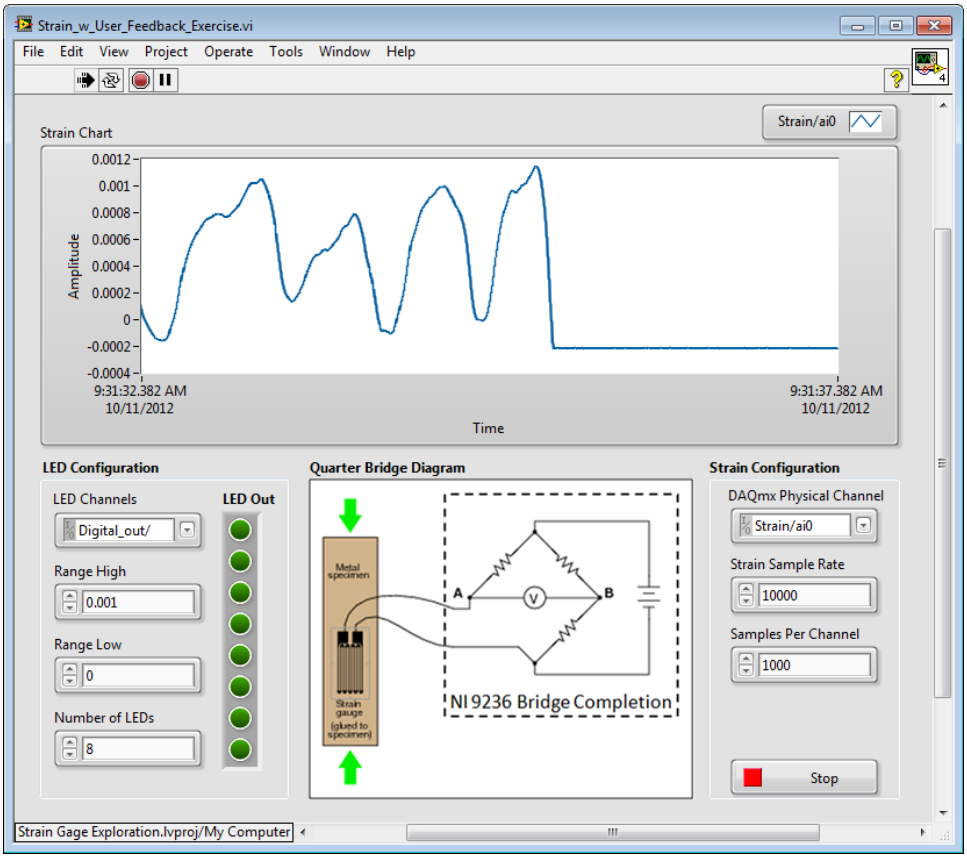


Figure 21. You have created a program to visualize strain gage data. This data can be analyzed, saved to file or used to control an output.

38. Press **Stop**. Save this VI, as you will use it in the next part of the exercise.

39. To finish out this portion of the exercise, open the LabVIEW Example Finder by selecting **Help » Find Examples**.

40. Navigate to **Hardware Input and Output » DAQmx » Analog Input » Strain – Continuous Input.vi**.

41. Change the *Physical Channel* to **Strain/ai0** (or the channel your strain gage is connected to), the *Strain Configuration* to **Quarter Bridge I** and the *Voltage Excitation Value* to **3.3**. See Figure 22 for reference.

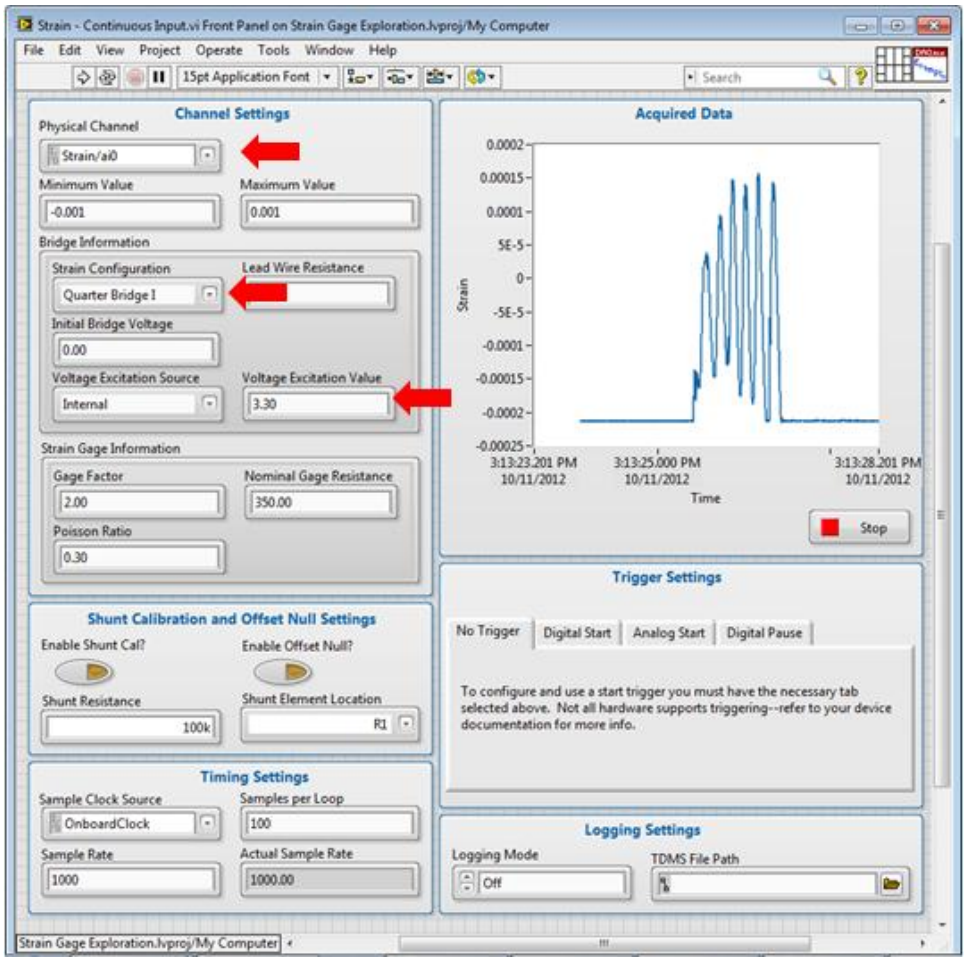


Figure 22. The DAQmx examples are built to give you a solid starting point for your programs. Use them as a starting point.

42. Press the run button. The graph should react to you **gently** pressing on the strain bar just like in your code. Press **Stop**.

43. Press **<Ctrl+E>** to open the block diagram. This code includes many more settings specific to strain gage measurements than the code you have just built, but it still follows the same exact DAQmx flow. Additionally, this code includes logging and triggering functionality. These could easily be added you your code by adding the appropriate VIs.

44. Close this example VI. When prompted, do not save your changes.

Part B Controlling the Colored LEDs for User Feedback

Estimated time: 30 minutes

For this part of the exercise, you will add to the code built in Part A.

Oftentimes, users need immediate feedback from a measurement system and a computer monitor may not be practical for applications like embedded measurement

systems, industrial monitoring or process control. A common way to alert an operator to the condition of the test is to use signal LEDs. For this next part of the exercise you will use your code to alert a user to the strain level on the aluminum bar.

Before you start this exercise, confirm that the LEDs are properly connected and you are able control them in Measurement & Automation Explorer.

1. Begin by opening MAX (**Start » All Programs » National Instruments » Measurement & Automation Explorer**).

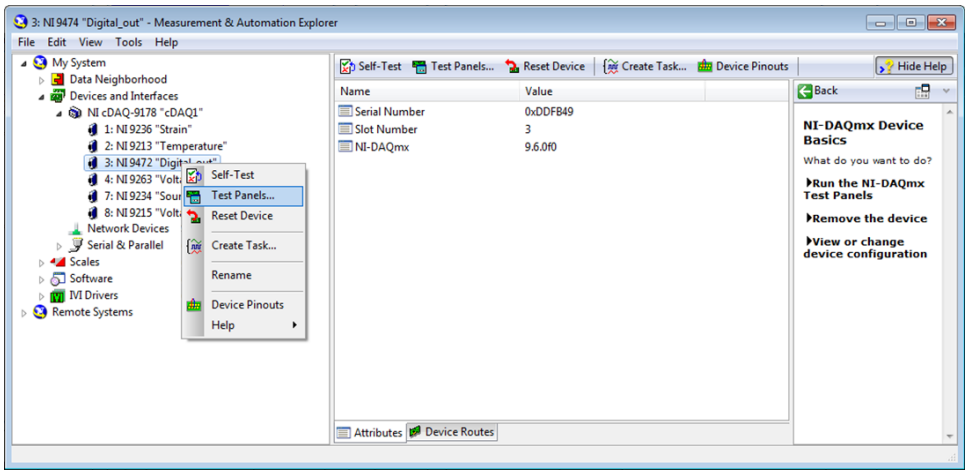


Figure 22. Starting with Test Panels ensures that your sensors are working and you will not have hardware problems.

2. Right click on the NI 9472 and select **Test Panels**. The following panel will open:

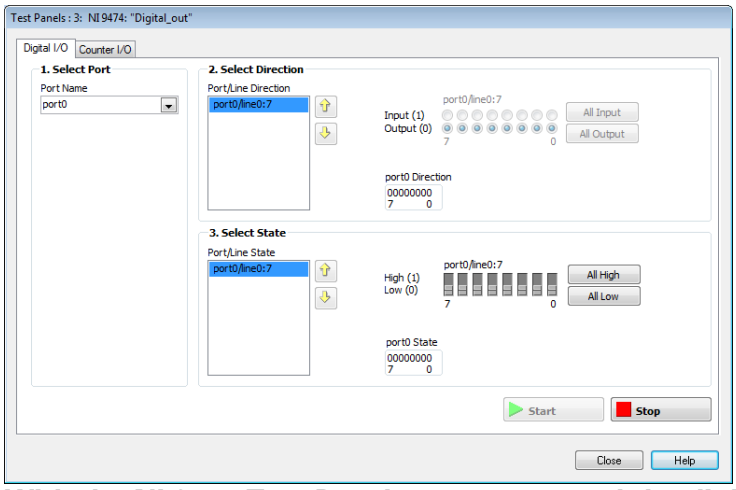


Figure 23. With the NI 9472 Test Panel, you can control the digital output channels to ensure they are operating properly.

3. Press **Start**.

4. Select the **All High** button as a quick test of all the digital outputs. You should see all the LEDs on your demo box light up. Notice that the LEDs on the NI 9472 also

light up. You can control each LED individually, but for testing operability, it is not essential.

5. Press **Stop** and close the Test Panel.

6. Open your copy of **Strain_w_User_Feedback_Exercise.vi**.

7. Press **<Ctrl+E>** to open the Block Diagram.

   In this part of the exercise, you will add code to control the LEDs proportionally to the strain experienced by the aluminum bar. To add this code, you will perform many of the same steps used in the strain acquisition, but this time, you are writing data, rather than reading. You should notice some overlap in the steps performed.

8. Like in Part A, a large part of the coding will come from the DAQmx palette. To access this palette right click on any white space on the Block Diagram and navigate to **Measurement I/O » NI-DAQmx.**

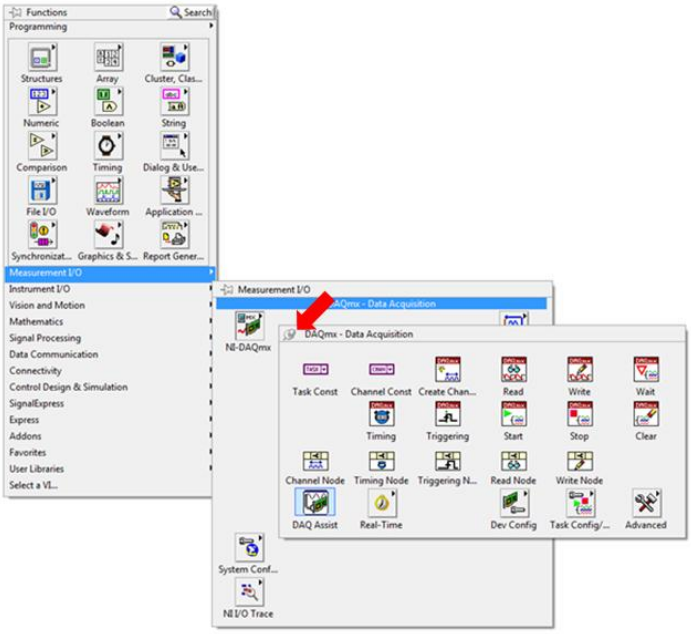


Figure 24. The DAQmx palette includes all the required VIs for communicating with your NI CompactDAQ chassis.

9. Click on the pin to keep this palette visible (red arrow in Figure 24).

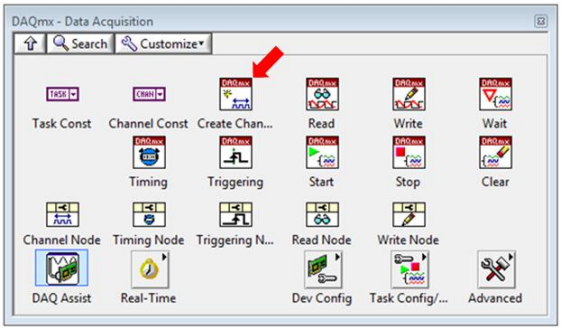


Figure 25. Pinning a palette enables you to click elsewhere in your code with the palette still visible.

10. To start, drag the **NI-DAQmx Create Channel** (red arrow in Figure 25) and place it in the position shown in Figure 26.

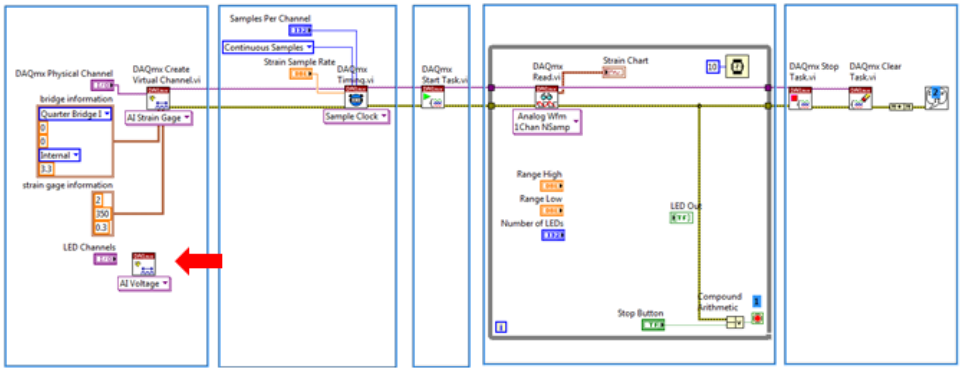


Figure 26. The digital output code will run parallel to the strain task.

11. Use the drop-down menu under the VI to change the **DAQmx Create Channel VI** to Digital Output.

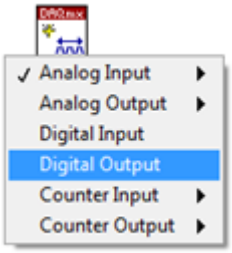


Figure 27. Choose Digital Output for controlling the signal LEDs.

12. Wire the LED Channel control to the Lines input of the VI. Press **<Ctrl+H>** and hover over the VI if you need a reminder where this input is.

13. Drag the following VIs from the DAQmx palette and place them as shown in Figure 28.
    a) DAQmx Start Task
    b) DAQmx Write
    c) DAQmx Stop Task
    d) DAQmx Clear Task

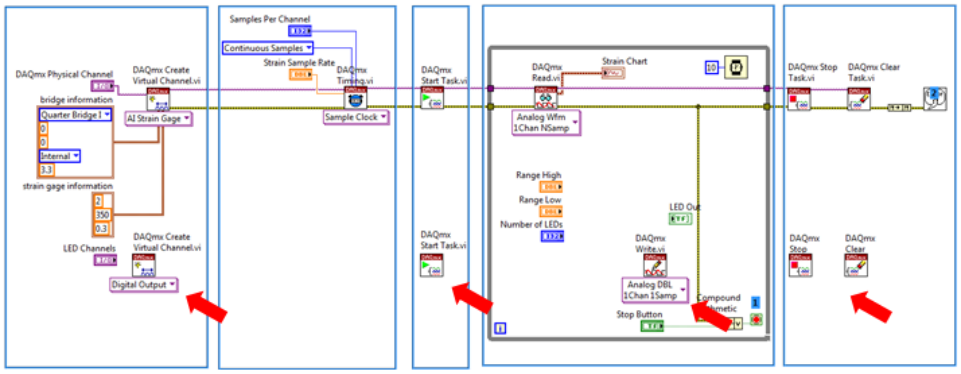


Figure 28. These VIs should look familiar – nearly all DAQ programs will follow this pattern.

14. Change the new **DAQmx Write VI** to **Digital » Single Channel » Single Sample » 1D Boolean (N lines)**.

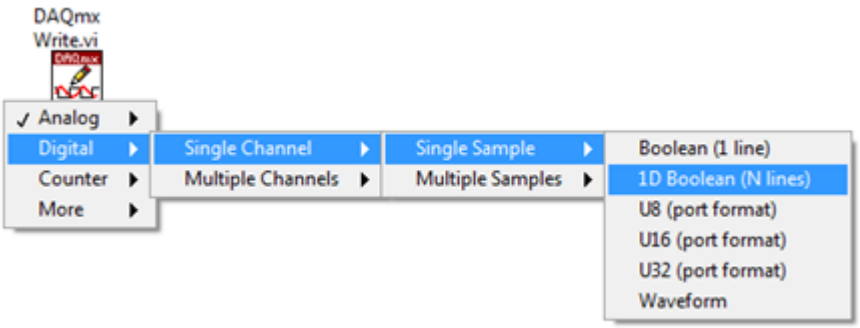


Figure 29. You can write an array to the digital lines on the NI 9472 using the DAQmx Write VI.

15. Wire the task and error wires through each of the DAQmx VIs like in Figure 30.

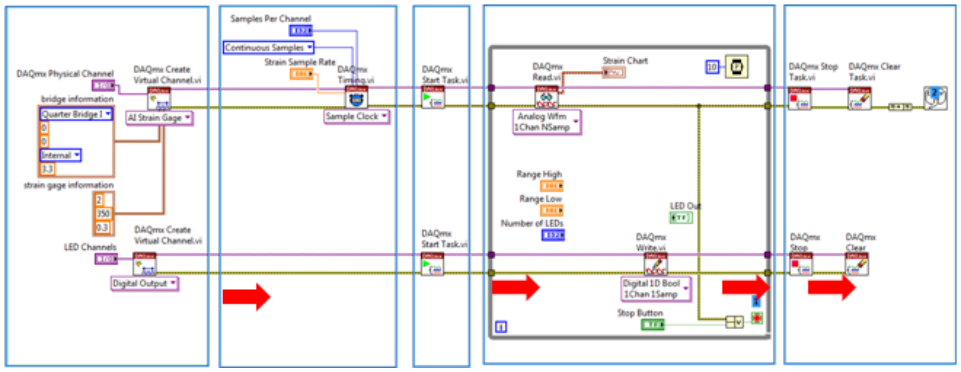


Figure 30. Wiring the task and error wires through dictates the flow of your code.

16. To properly handle the errors, you will need to ensure that the while loop stops if your digital code has an error. To do this, start by expanding the Compound Arithmetic. Hover over the VI and drag the handle up to expose another input.

Figure 31. Expanding the Compound Arithmetic ORs all the inputs.

17. Wire the error wire from the output of the DAQmx Write to the new Compound Arithmetic input.

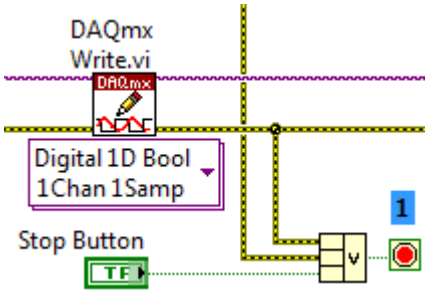


Figure 32. The Compound OR now accounts for all the errors inside the While Loop.

18. At the end of your code, you will need to combine all the error wires, alerting the user of any errors in the code. To do this, expand the Merge Errors function to expose a new input.

Figure 33. Combine all the error wires in your code with the Combine Errors VI.

19. Wire the error wire from the output of the bottom DAQmx Clear Task VI into the new input of the Merge Errors function. Your code should look like Figure 34.

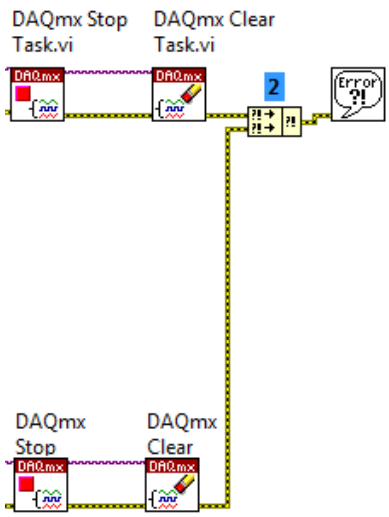


Figure 34. All your error wires are combines so the Simple Error Handler can display the errors to the user.

20. Now that you have the code for writing the data to the digital outputs, you will need to create the code to translate the analog data from the strain gage to a digital level. Luckily, that code has already been created for you as a subVI. To add the subVI, right click on some white space on the Block Diagram and click on **Select a VI…**

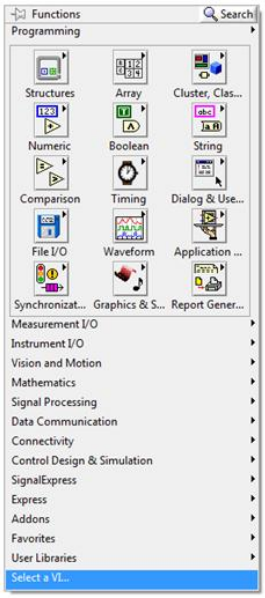


Figure 35. Press Select a VI… to find a subVI that isn't visible on your palettes.

21. Navigate to **5 – Strain\Support VIs** and select **LED_Scaled_Output.vi**.

22. Place the VI as shown in Figure 36.

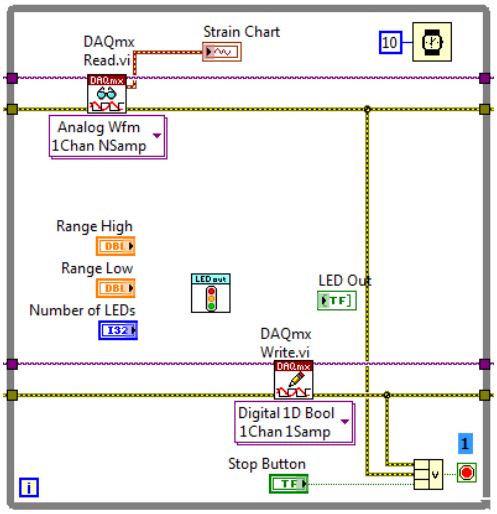


Figure 36. The LED_Scaled_Output VI is a subVI to scale the strain input to an LED output.

An alternate way to place the subVI on your Block Diagram is to navigate to it in the Project Explorer and drag it onto the Block Diagram, like in Figure 37.

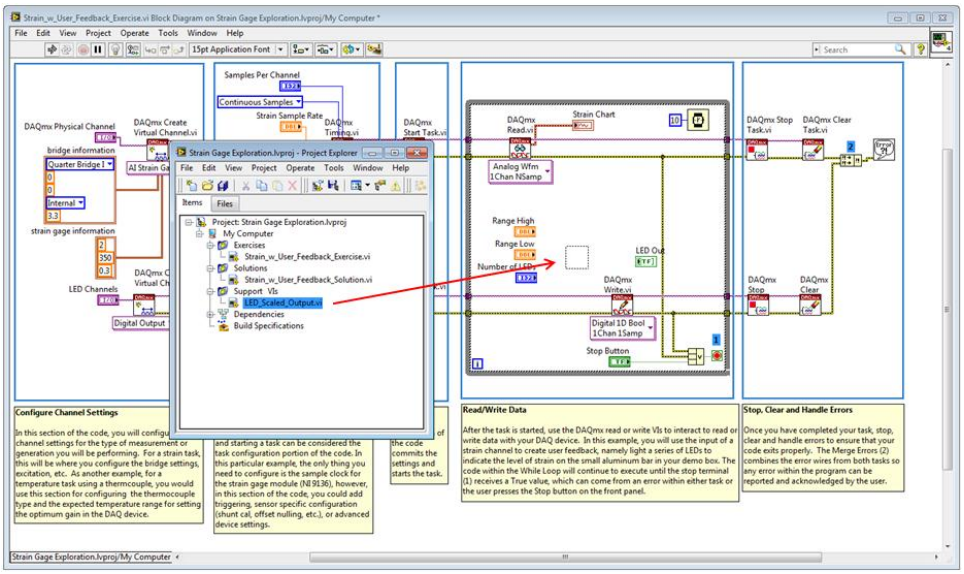


Figure 37. You can drag any VIs from your project onto the Block Diagram.

23. Organizing your code into subVIs a way to make your code more manageable and scalable. To examine the contents of the subVI, just double-click the icon to open the subVI's front panel. Take a minute to examine the contents of the subVI.

24. Close the subVI.

25. Wire the call to the subVI like in Figure 38.

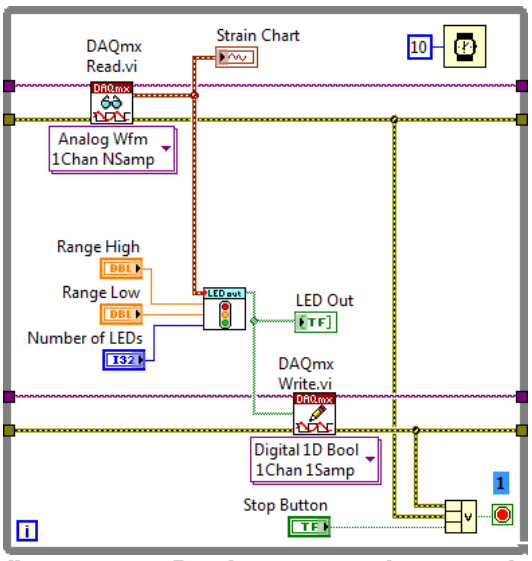


Figure 38. This subVI outputs a Boolean array that can then be output by the DAQmx Write.

26. Your code is ready to run. Switch to the Front Panel and press the Run button. You should notice that the LEDs light proportionally to the amount of force applied to the strain gage.

    If the scaling seems off. Press on the bar and take note of the highest and lowest values on the graph. Stop the VI, adjust the Range High and Range Low values to match the range of the strain. Restart the VI and you should see the LEDs reacting to pressure on the strain bar.

27. If you watch the front panel and the LEDs, they should be reversed in how they are displayed. The physical LEDs and the front panel indicator can be matched in code easily. To swap the order of the front panel indicator, start by deleting the wire between the LED subVI and the LED indicator on the block diagram. Click once on the wire shown in Figure 39 and press delete.

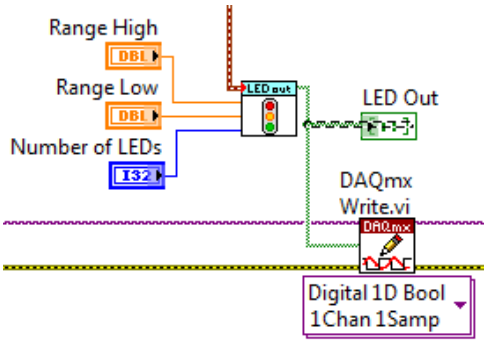


Figure 39. Delete the wire between the LED subVI and the LED indicator.

28. To switch the order of a 1D array, use the Reverse 1D Array function. Right click on white space on the block diagram. Select **Programming » Array » Reverse 1D Array**.

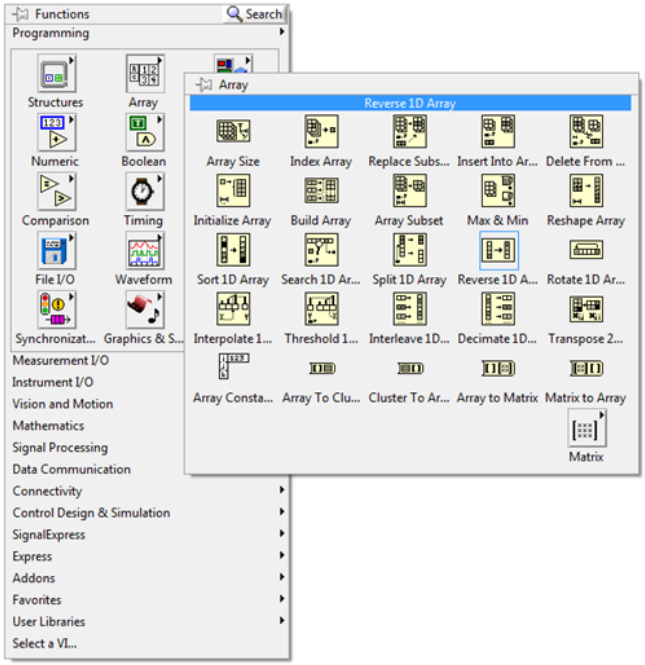


Figure 40. The Array palette includes many useful VIs for array manipulation.

29. Place the VI in between the LED out subVI and the LED out indicator and wire it up like in Figure 41.

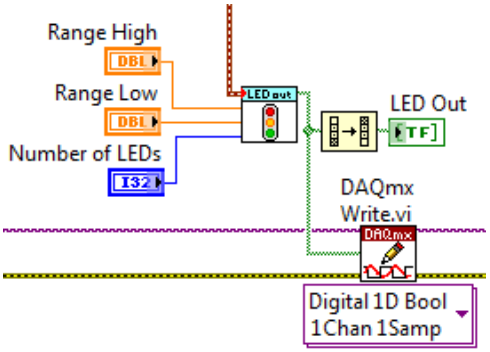


Figure 41. Use Reverse 1D Array to swap the order that the Boolean array is displayed.

<End of Exercise>

Audio Equalizer

Exercise 1: Acquire Audio Signal

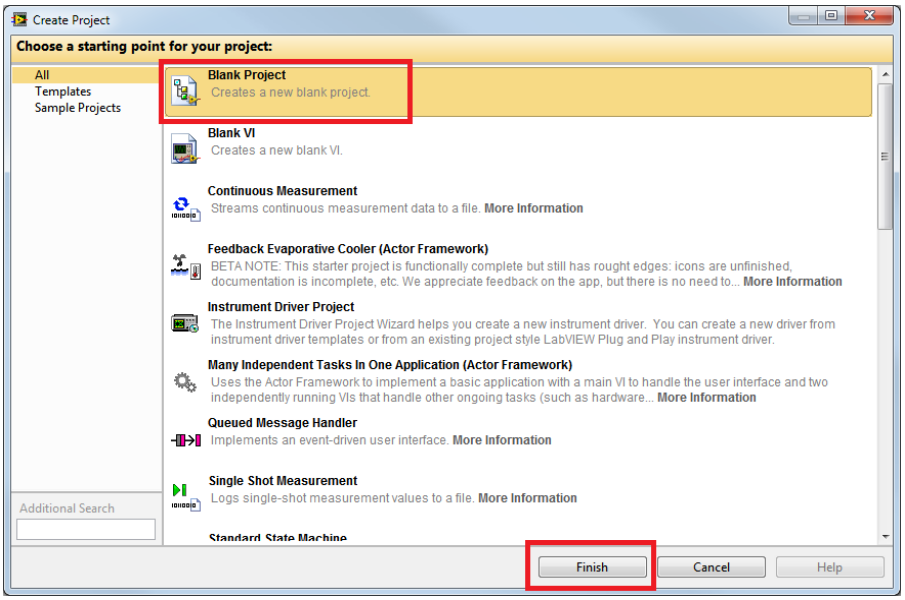
Note: This exercise assumes that you have already installed all required software and configured all hardware as outlined in earlier sections. If you have not, please refer back to the appropriate section to do so.

Goals

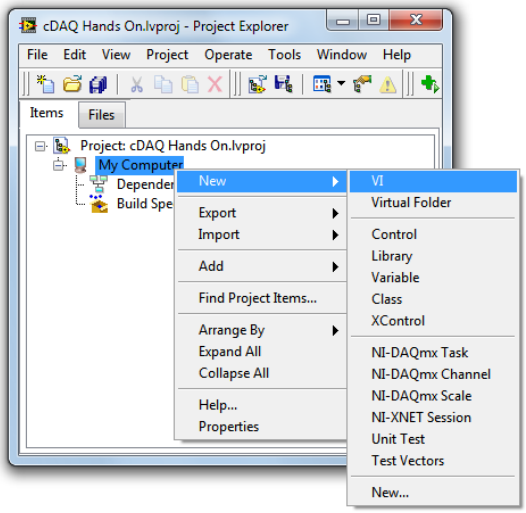
- The first goal is to continuously acquire 40000 data point arrays from the audio source connected to the NI 9215 (or similar Analog Input) module
- The second goal is to add data logging to the DAQmx task

Part A

1. Launch LabVIEW by navigating to **Start » All Programs » National Instruments » LabVIEW <Year> » LabVIEW.**
2. Create a new LabVIEW Project by selecting the **Create Project** button from the Getting Started Window.
3. Select **Blank Project** and then select **Finish** to create a project file to hold all files and references for the application.



4. In the newly opened LabVIEW Project Explorer, navigate to **File » Save As…** and save your project as **cDAQ Hands On.lvproj** within the **C:\Seminars\LV HO\Exercises\6 – Audio Equalizer** folder.
5. Within the LabVIEW project window, right-click *My Computer* and select **New » VI** to create a new LabVIEW Virtual Instrument (VI) to acquire data.

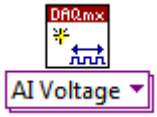


6. Select **File » Save** to save the newly created VI. Name the VI "Exercise 1a" and save the VI within the same folder as your LabVIEW Project.

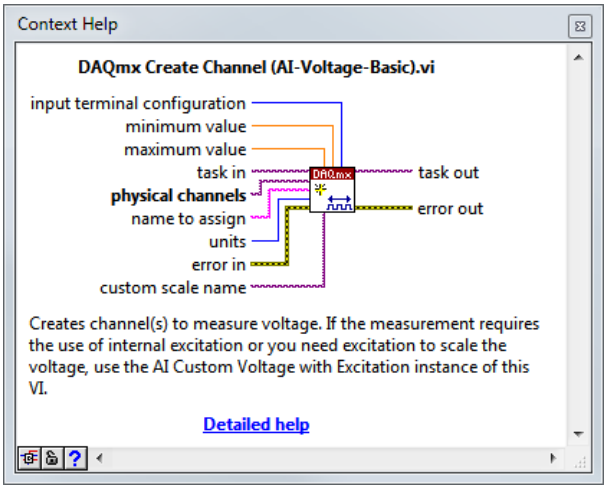
7. Navigate to the Block Diagram window of *Exercise 1a.vi*. If you are on the Front Panel, press **<Ctrl+E>** to toggle between Block Diagram and Front Panel.

8. Right-click on empty white space in the window to bring up the Functions Palette and navigate to **Functions » Measurement I/O » NI-DAQmx**. This is the DAQmx Functions Palette.
    a. Now select the first function, **DAQmx Create Virtual Channel**. Left-click to select the function, and then left-click to place it onto the Block Diagram. This function opens a reference to and reserves a DAQ hardware device.

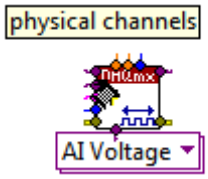
    b. Once placed, left-click the white polymorphic selection box and navigate to **Analog Input » Voltage**. Although we are acquiring a sound signal, we will still use a voltage because this is what the MP3 player outputs.



    c. Turn on Context Help by pressing **<Ctrl+H>** and then hover over the *Create Channel VI* to see all of the inputs and outputs.



d. Hover over the upper-left corner of the Create Channel VI and notice that the cursor changes to the wiring tool automatically and the tip-strip displays for the terminal over which your mouse is located.

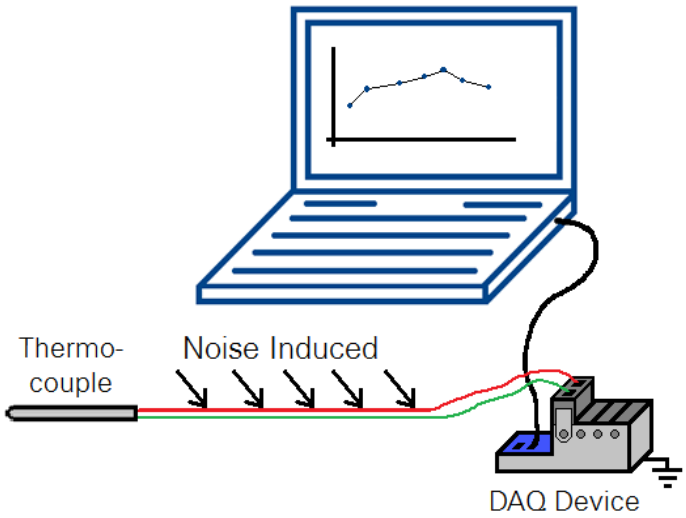


e. Right-click the terminal for **Physical Channels** – the *second* terminal from the top – and select **Create » Constant.** This constant will only be visible on the Block Diagram, and there is no corresponding control or indicator on the Front Panel.

f. Left-click the arrow on the Physical Channels constant and browse to the channel of interest, **Voltage_in/ai1**.

g. Right-click the **Input terminal configuration** terminal and select **Create » Constant**. Set this constant value to *Differential.*

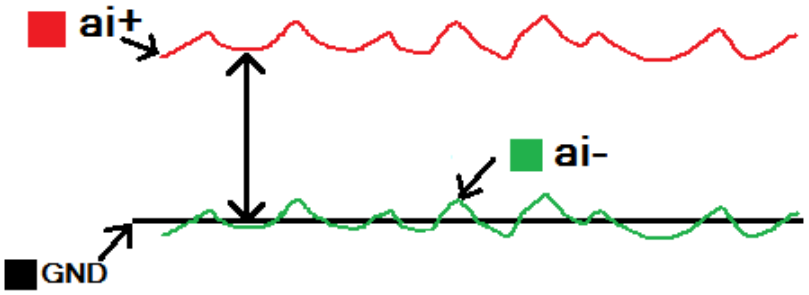
h. The next image displays is a common sensor setup diagram. In this example, a thermocouple is connected to an NI CompactDAQ device.



Noise and other signals can enter into the signal through the wires between the sensor and the DAQ device, as wires act like antennas. Shielding and other techniques can help to reduce this affect.

The terminal input configuration is the easiest way to eliminate a lot of the effect of noise. Single-ended measurements (RSE and NRSE) only take the positive sensor signal (ai+) and then they reference this to the system or chassis ground (GND). The chassis ground typically does not have any noise on it, and is represented by the solid black line below. The signal line does typically have noise on it.

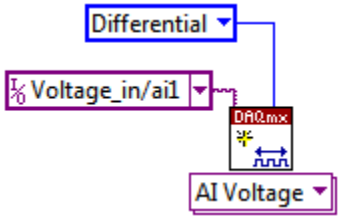
With Single-ended measurements, the noise will still be seen because the acquired data will be the difference between the noisy signal line, and the clean GND line.



Differential configuration means that the ADC will take the difference between the ai1+ and ai1- pins on the DAQ device, which is the most accurate way to measure from a sensor as it eliminates noise that is common on both the positive and negative wires.

Always use this configuration if available for sensor acquisition. The resulting

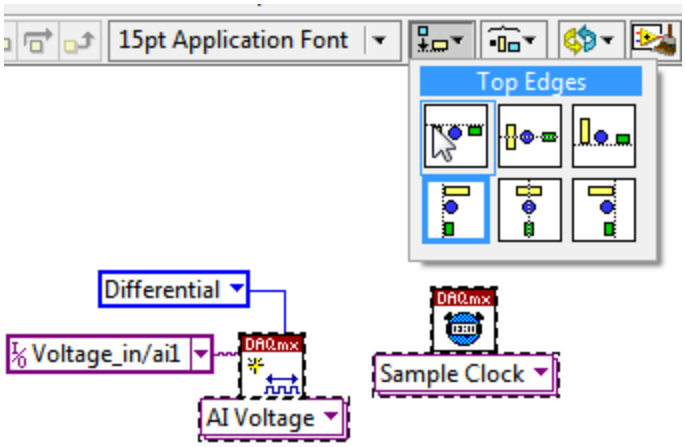
Block Diagram should appear as follows.



9. Right-click the *Create Channel VI* and navigate to **DAQmx – Data Acquisition Palette » DAQmx Timing**. This is a shortcut to quickly place another function from the same palette, NI-DAQmx.

    a. Click to place the *DAQmx Timing VI* to the right of the Create Channel VI.

    b. To align any functions on the Block Diagram, select by holding <Shift> and left-clicking the functions. Then select the **Align Objects** button on the toolbar and choose **Align Top Edges**.



    c. Now wire the **Task Out** and **Error Out** of the Create Channel VI to the **Task In** and **Error In** of the Timing VI, respectively.

    d. If you diagram is cluttered or messy, press the **Clean Up Diagram** button () in the toolbar or simply press **<Ctrl+U>** to have LabVIEW automatically arrange the Block Diagram.

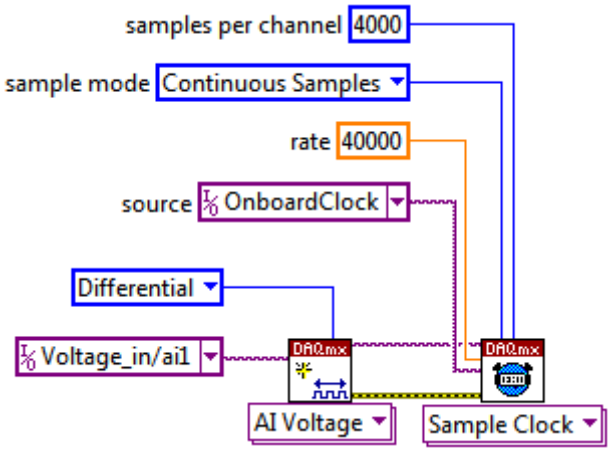
    e. Create a constant for the **Sample mode** input terminal and select **Continuous Samples**, which means that you will continuously acquire data until the task is cleared.

    f. Create a constant for the **Rate** input terminal and enter **40k** for the value. LabVIEW can interpret metric abbreviations for units such a 'k' for thousand, and 'm' for milli.

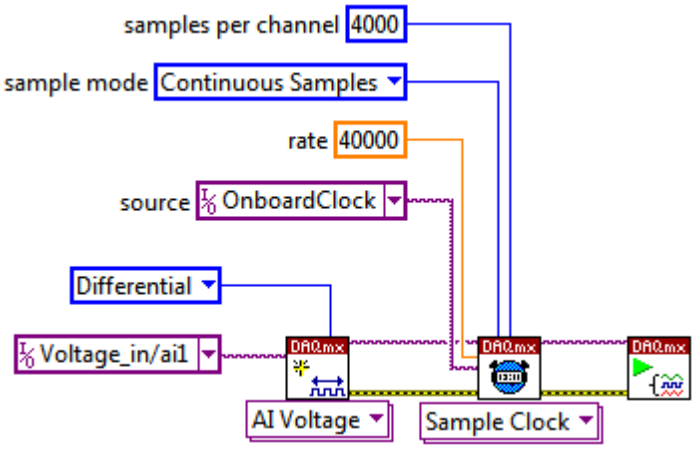
g. Create a constant for the **Source** input and select **OnboardClock**. This means that the task will use the Analog Input internal timing engine on the 9215 module to acquire data at a rate of 40k Samples per second.

h. Create a constant for **Samples per channel** input terminal and set the value to **4k**. This means you will acquire 4k Samples every time the *DAQmx Read VI* is called corresponding to one second of data. This will configure the buffer for the transfer of data from the DAQ device to the System Memory.

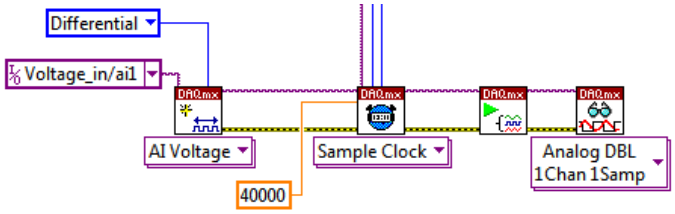
i. A good benchmark is to acquire 1/10th of the number of samples as the sample rate. This is not required, but you can do this programmatically using a Divide function in LabVIEW. This function is located in the **Programming » Numeric** palette.



10. Place a NI-DAQmx Start Task VI to the right of the NI-DAQmx Timing VI and wire the error and task wires together from each VI. This function will begin the acquisition of data from the device and begin to store data in the on-board FIFO memory of the device.

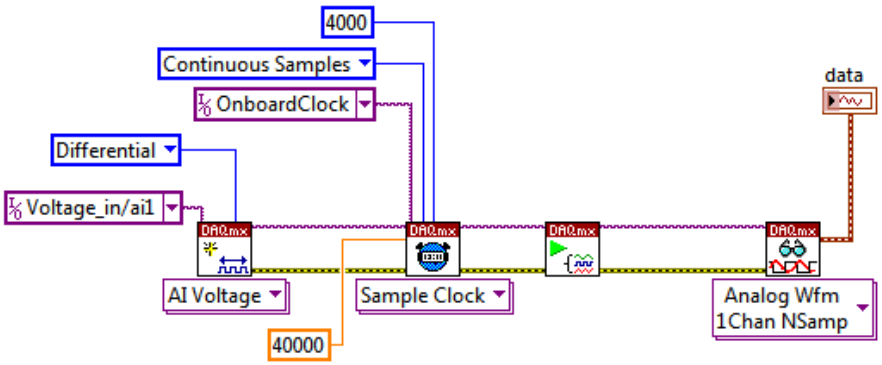


11. Place a NI-DAQmx Read VI to the right of the Start Task VI and wire the task and error terminals together from each VI.



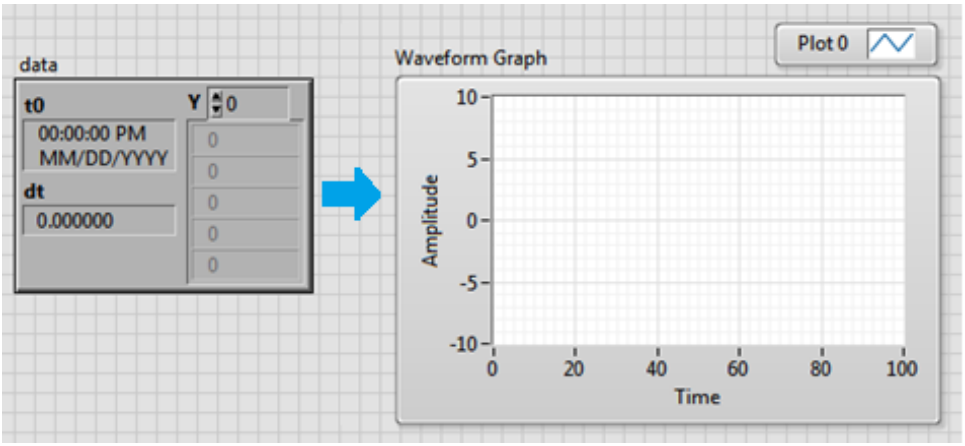
a. The VI's polymorphic instance is by default set to **Analog » Single Channel » Single Sample » DBL**. Because we will be reading 4k samples at a time, we will need to change this to **Analog » Single Channel » Multiple Samples » Waveform**.

b. Right-click the data output terminal of the DAQmx Read VI and create an indicator.



c. Locate the indicator on the Front Panel. This is a waveform indicator, which is the proper indicator for the data type, but it would be better to visualize the results on a graph.

d. Right-click the waveform indicator on the Front Panel and choose **Replace » Silver » Graph » Waveform Graph**. A waveform graph will display an entire waveform of data, and will redraw the graph each time it receives new waveform data.



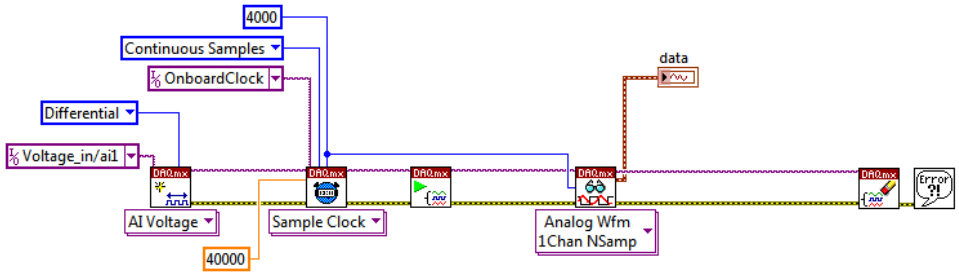
e. On the block diagram, wire the **Samples per Channel** constant of the DAQmx Timing VI to the **Number of Samples per Channel** input of the DAQmx Read VI.

12. Return to the Block Diagram and place a NI-DAQmx Clear Task VI to the right of the Read VI and wire the Error and Task wires. This will clear and dispose of all references to the hardware.

13. Place a Simple Error Handler to the right of the Clear Task VI. This is located in the **Programming » Dialog and User Interface** palette, but you can also use the search button of the Functions Palette.

    a. Wire the **Error Out** terminal to the **Error In** terminal of the Simple Error Handler VI.

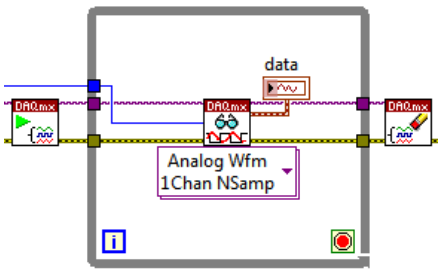
    b. Because you have wired all of the error terminals together, the errors will be passed from one VI to the next, and then handled at the end. Once handled, the error message will be displayed, if there is an error detected.



14. To acquire continuously, you will need to use an execution structure to repeat code continuously until a condition is met; this is called a While Loop.

    a. On the Block Diagram, open the Functions Palette and navigate to **Programming » Structures » While Loop**. This will turn your mouse cursor into a while loop cursor.

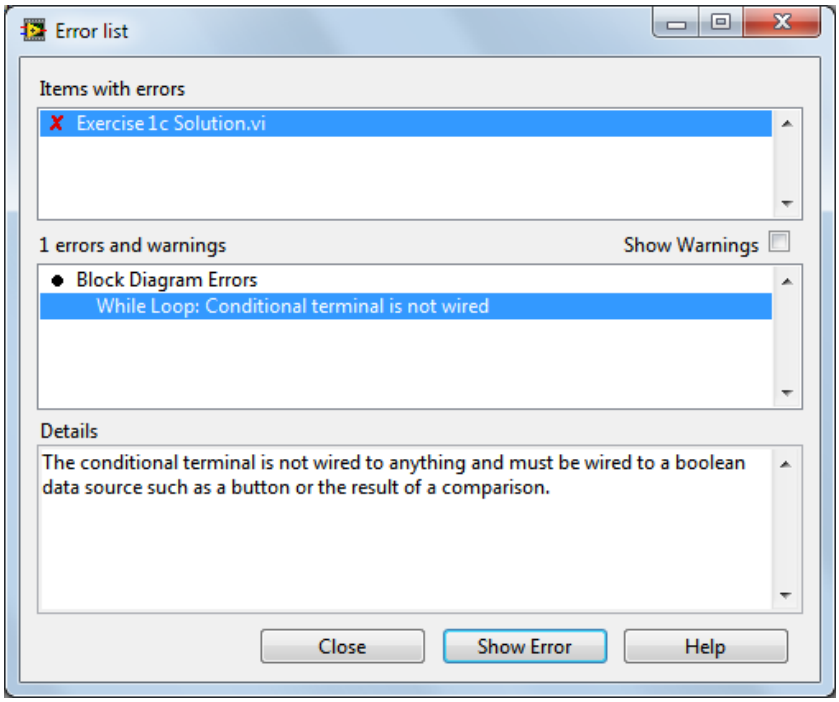
    b. Left-click-and-drag around everything that needs to run continuously in the VI, which is the DAQmx Read Function. Release the mouse button to complete the drawing of the loop.



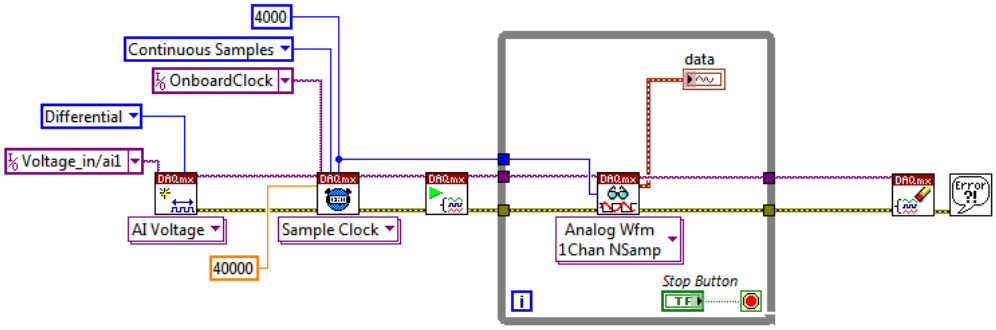
    c. Notice the solid boxes that are created on the edges of the While Loop

border; these are called tunnels and are used to pass data from outside the loop into the loop before it begins, and then from inside the loop outside the loop after the condition is met and the loop is exited.

d. Try to run the VI by pressing the run arrow – notice that the run arrow is broken (), but go ahead and click on it. The VI will not run, but it will **list the errors** that need to be resolved before you are able to run. The **Error list** window will appear, and you can click on **Show Error** and LabVIEW will take you to the error source.

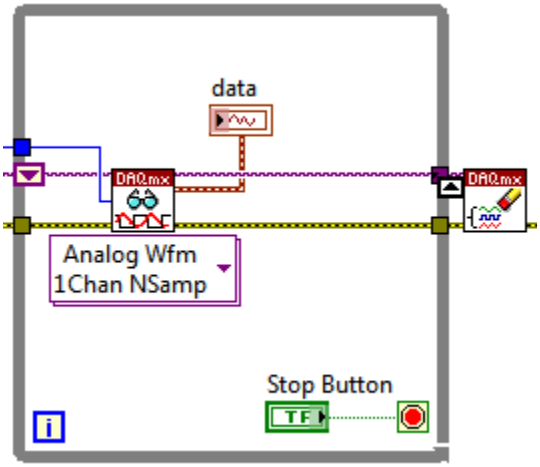


15. You now need to wire a Boolean (True/False) control into the **Conditional Terminal** () of the loop (which looks like a stop sign). Right-click the input and select **Create » Control** to create an interactive control – a stop button on the Front Panel.



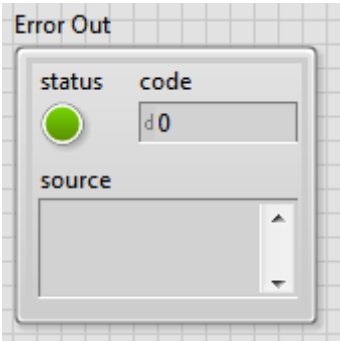
16. To ensure that the DAQmx Read VI receives the Task and Error values are passed from one iteration to the next, you will need to use shift registers to pass data.

a. Right-click the output tunnel of the purple DAQmx Task wire and select Replace with Shift Register. This will change the output tunnel to a shift register, and then allow you to select the proper input terminal. Left-click the input tunnel for the DAQmx Task wire.



b. Repeat for error wire tunnels.

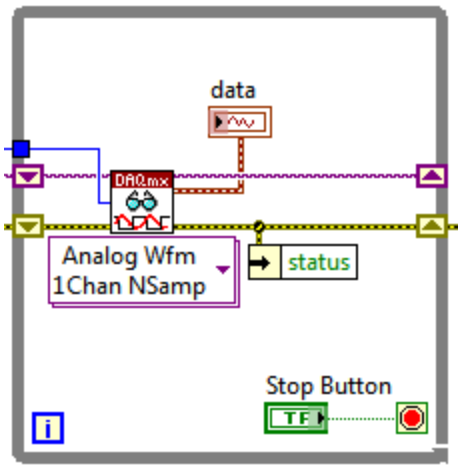
17. To ensure that the VI exits the loop in the event that an error is generated within the loop, it is necessary to unbundle the error cluster which contains an error status Boolean, an error code numeric, and an error source string.



a. Select the While Loop on the Block Diagram and expand it vertically downward, and then move the conditional terminal and stop button to the bottom again to make room for more functions within the loop.

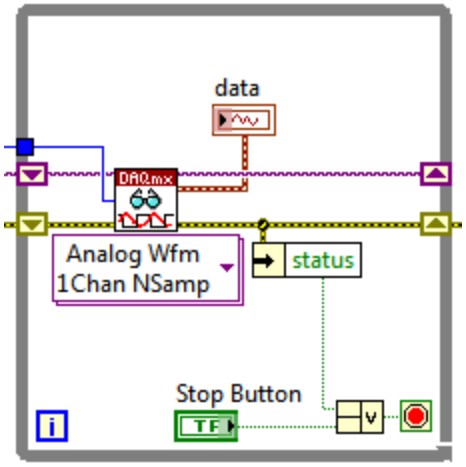
b. On the Functions Palette, navigate to **Programming » Cluster, Class, & Variant » Unbundle by Name** and place this function in the empty space you just created.

c. Wire from the Error output wire of the DAQmx Read to the input of the Unbundle by Name function. This will automatically populate the names of all of the elements of the Error cluster, and you can left-click to select the element that you wish to unbundle. Select **Status**.



d. Now place a Compound Arithmetic function with the Boolean functionality of OR with multiple inputs. This is located in the Functions Palette under **Programming » Boolean**. This will create a multi-input OR function that will report a Boolean True if any one of the inputs is True, which stops the While Loop execution. **Note**: you can use a basic Boolean OR function for this example, but the compound is used as it will enable you to grow the application in a later exercise.

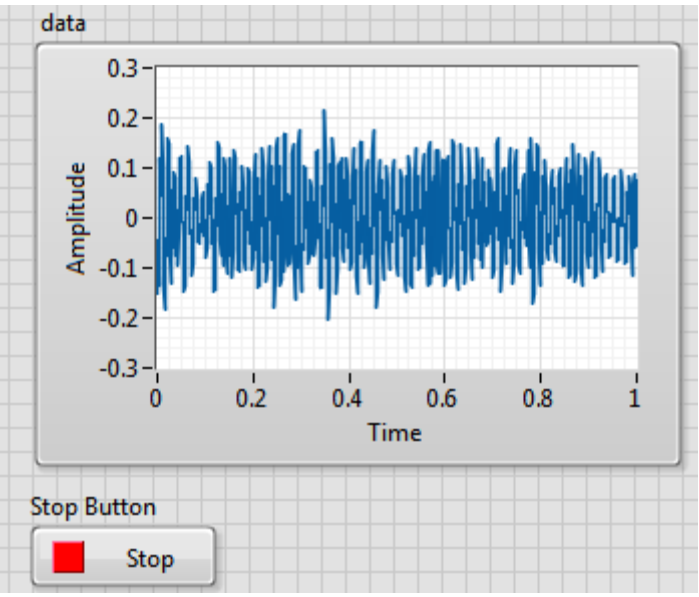
e. Wire the output of the Compound OR function into the Conditional Terminal of the While loop.



18. Save your VI by pressing **<Ctrl+S>**.

19. Navigate to the Front Panel and press the run button or simply press **<Ctrl+R>** to run the VI. Ensure that your audio source is playing a song – for example, from your cell phone, or from the audio output jack of the laptop you are working on.

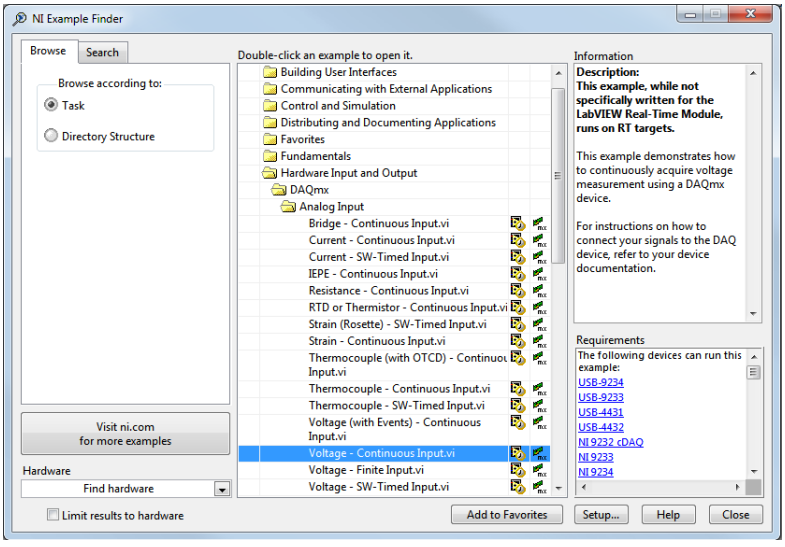
    a. You should see a sound signal in the waveform graph indicator similar to the image below.



Part B The following section will explain how to log data using the TDMS format. This format allows for easy organization of data, high streaming rates to disk, and other great qualities for engineering data storage. NI offers a program dedicated to opening and manipulating data in TDMS format, NI DIAdem. If you have more than 30K samples in your log, you are approaching Microsoft Excel's limitations and DIAdem is the right tool for you.

1. To log the acquired data to the previous example, only one (1) function needs to be added to the chain of DAQmx code. Fortunately, there is already an example built by the DAQmx R&D team at National Instruments that has this function added.

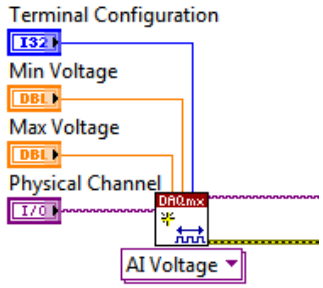
    a. Within the LabVIEW environment, navigate to **Help » Find Examples….** Once the NI Example Finder is loaded, browse to **Hardware Input and Output » DAQmx » Analog Input » Voltage – Continuous Input.vi**. Double-click to open this example.



b. Explore the Front Panel. This was made entirely using LabVIEW Front Panel controls, indicators, and decorations.

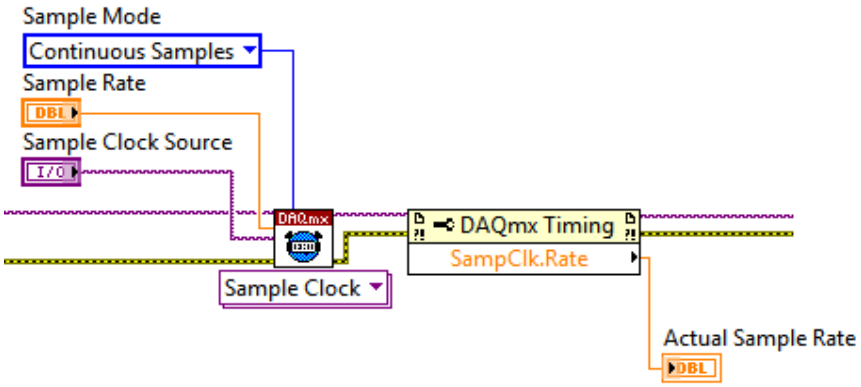
2. Press **<Ctrl+E>** to open the Block Diagram. Observe the code. If you still have the solution to Exercise 1a open, compare your Block Diagram code to this code. There are a few differences in this Block Diagram in comparison to your Block Diagram.

   a. For the Create Channel VI, all of the inputs are set to be controls rather than indicators. This allows the operator to easily change things directly from the front panel, instead of having to go to the block diagram. Also, the Max and Min Voltage input terminals have controls. This allows you to specify the range of the input signal. There are limits for this range, and you should refer to the specifications document for these limits.



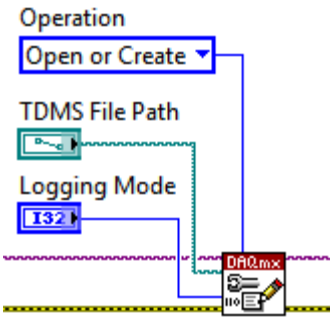
Channel Settings

   b. For the Timing VI, again, controls are used instead of constants. There is also an advanced function used, called a *Property Node*. Turn on context help **<Ctrl+H>** and mouse over the property node to learn more. This property node is used to read out the 'actual' rate of the sample clock. Some timers can only sample at certain rates; if you do not specify one of those rates, it will be coerced into that rate.



Timing Settings

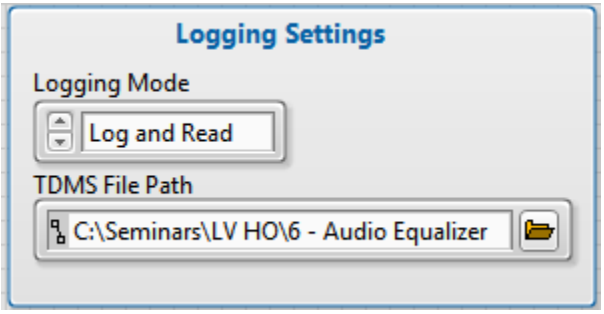
c. Here is a brand new VI, DAQmx Configure Logging (TDMS).vi. This enables you to automatically stream (log) the data acquired to your DAQ device directly to disk using the Technical Data Management (TDM) format. If you are using a sensor such as an accelerometer or thermocouple that has custom scaling, it will log in the scaled engineering units.



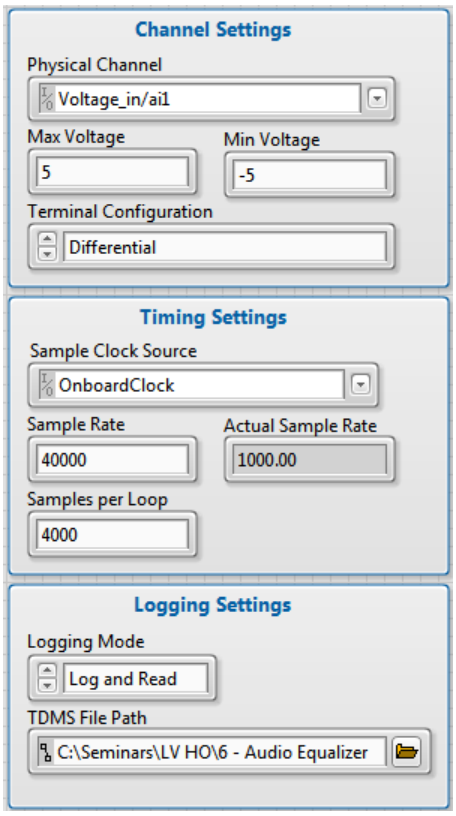
d. Explore the remaining functions of the VI to notice a few more changes such as the Read and Boolean OR functions.

3. To explore the functionality of the data logging, return to the Front Panel. Locate the Logging Settings section on the Front Panel.

a. Mouse over the right folder section of the TDMS File Path and left-click to browse to the proper file path. This allows you to browse to the file location, but you must also enter the name of the file to be created before selecting **OK** in the Open Dialog Window. Enter "Exercise 1b Data" for the file name and place the file within the project folder and then proceed.



b. Change the controls on the Front Panel so that they match the image below:



c. Run the VI for a few seconds, and then press the **Stop** button on the Front Panel to end the acquisition. Assuming that no errors were encountered, the data should now be logged to file.

4. If you have Excel installed, you should be able to locate and double-click the file to open it directly in Excel. If not, you can try to first open Excel, and then choose to open the file from the Excel environment.

   a. After you open the file in Excel, the root tab is the default open tab. This tab contains high level information about the file, but does not actually have the data points.

   b. To see the data, navigate the _unnamedTask<1> tab. This tab has every single data point that was acquired, and the top row has the physical channel name.

	A	B	C
1	**Voltage_in/ai1**		
2	0.030820487		
3	0.030820487		
4	0.030820487		
5	0.031136481		
6	0.031452475		
7	0.030820487		
8	0.030820487		
9	0.030820487		
10	0.030820487		
11	0.030820487		
12	0.031136481		

c. You can choose to graph and manipulate the data however you please from this point.

5. Close the DAQmx Example VI, but keep your Exercise VI open for the next step.

<Exercise Continued on Next Page>

Exercise 2: Filtering and FFT

Goals

- Add a Fast Fourier Transform (FFT) to the audio signal to view the signal in the frequency spectrum
- Add a low-pass filter to remove the high-frequencies from the audio signal
- Add a band-pass and high-pass filter to split the signal into Bass, Mid-tone, and Treble, attenuate signals, and mix back together
- Create a subVI to contain all of the signal processing

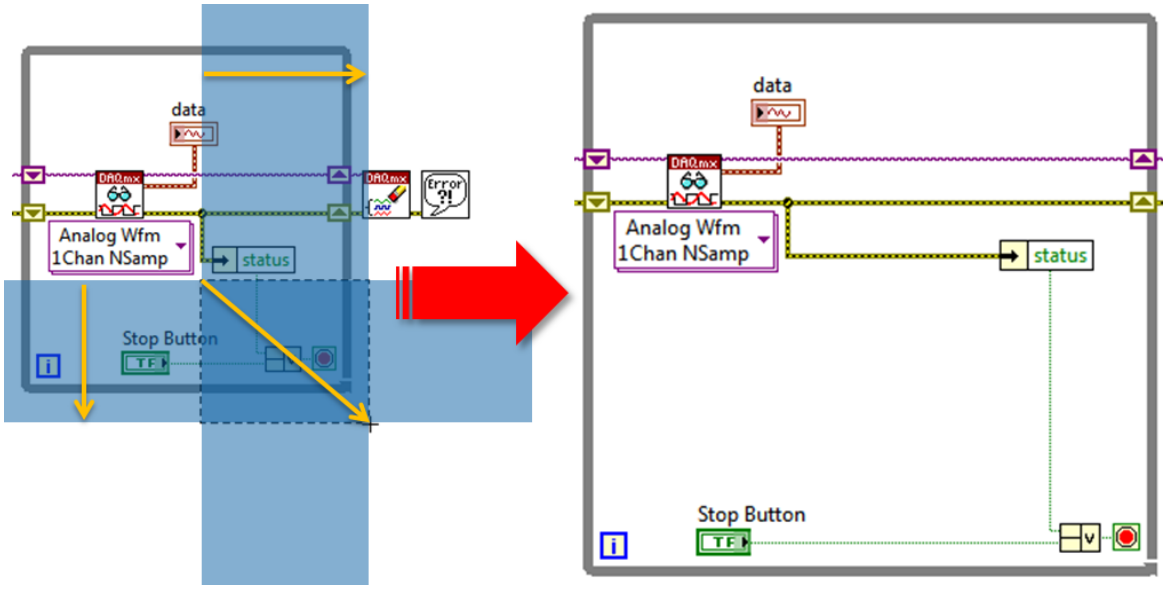
Part A

1. Before placing the signal processing functions, first create empty space within the while loop for these functions.

    a. While holding <Ctrl>, **left-click-and-drag** to place blank white space on the block diagram within the while loop

    b. Try this a few times to get used to how it works. If you do not like the result, press **<Ctrl+Z>** to undo your last action.

    c. Here is an example on how this functionality works.



2. Open the Functions Palette and navigate to **Signal Processing » Waveform Measurements » Power Spectrum and PSD** and place this VI to the right and below the DAQmx Read VI.

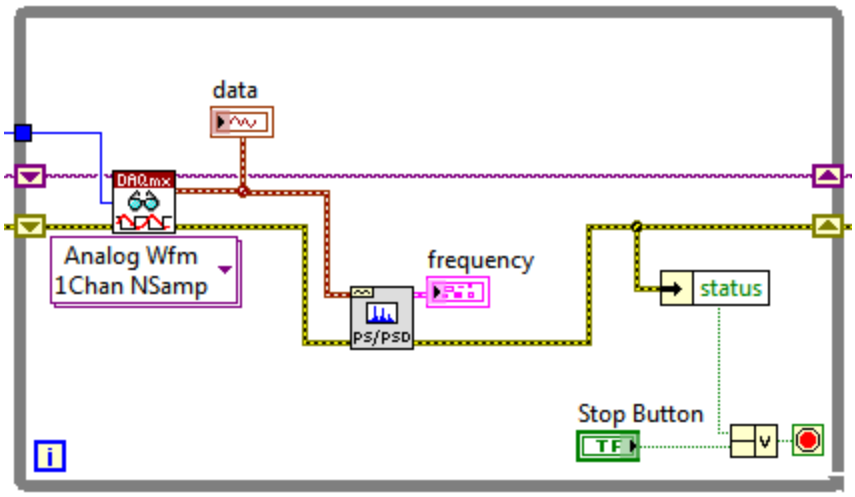
    a. Wire the DAQmx Read **data** output into the **Time signal** input of the Power Spectrum.

    b. This function will compute a fast Fourier transformation and output a Power Spectrum, which displays the power of each signal for each frequency band.

    c. Create a waveform Graph on the Front Panel and wire the **Power Spectrum**

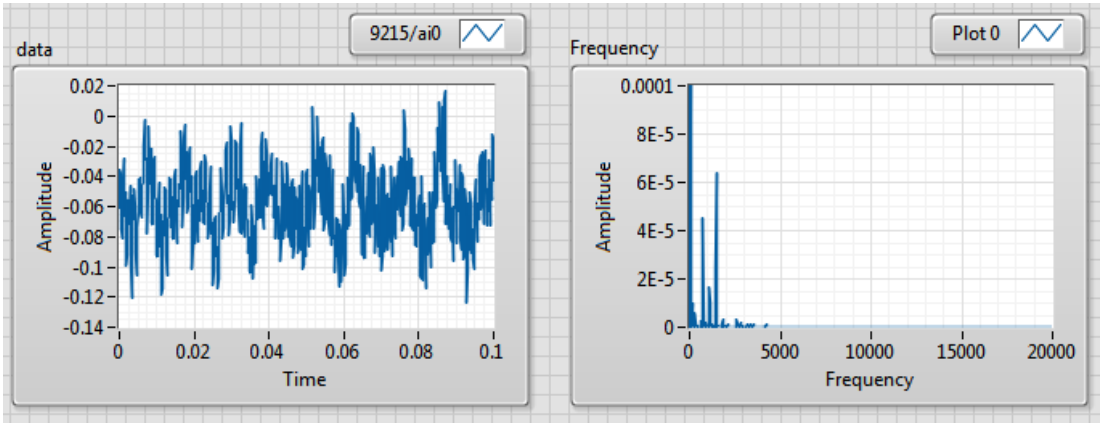
/ PSD output into the input of this graph.

d. Wire the Error out of the DAQmx Read into the Error In of the II Filter, and then connect the Error Output into the Shift Register and the Unbundle by Name function.



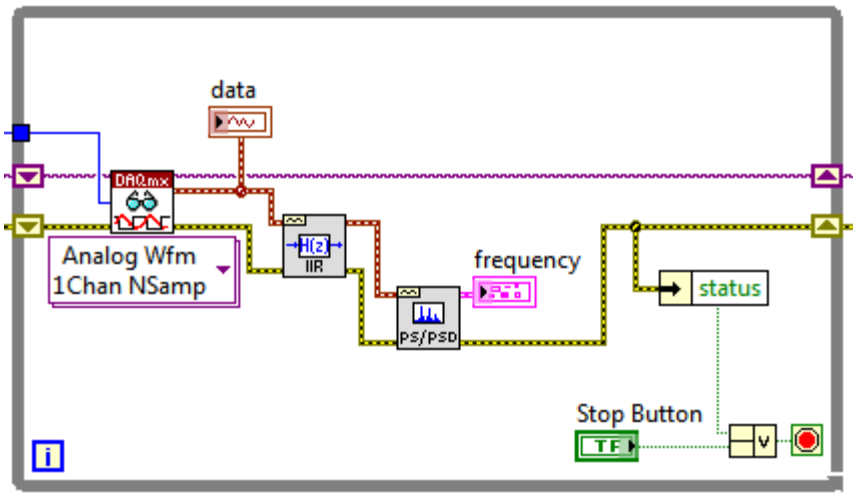
3. Save the VI.

4. Run the VI to observe the Frequency spectrum of the audio signal

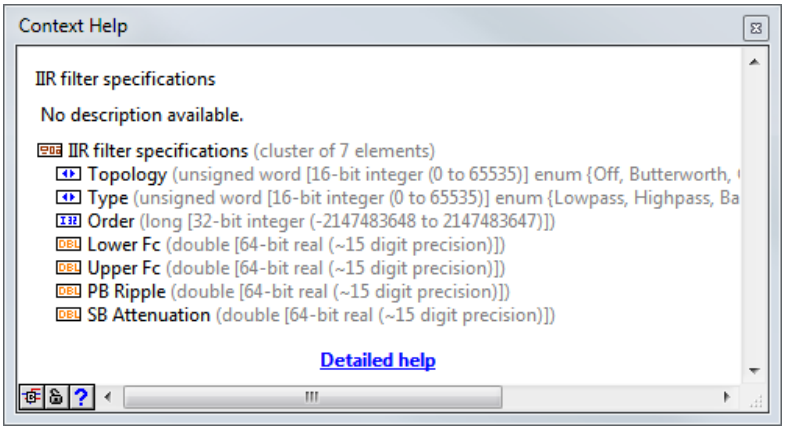


Part B

1. Open the Functions Palette and navigate to **Signal Processing » Waveform Conditioning » Digital IIR Filter** and place the VI in-between the DAQmx Read and Power Spectrum VI so that the signal is filtered before it is transformed to the frequency spectrum. Wire accordingly for error and signal wires. Expand the while loop if you need more space.

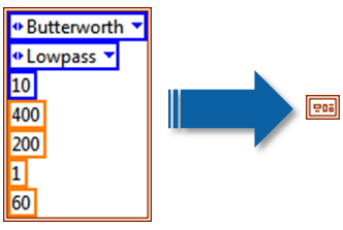


a. Create a constant for the **IIR filter specifications** input to the filter. This is a cluster of data that contains many items. Mouse over each with Context Help on to find out what each represents.



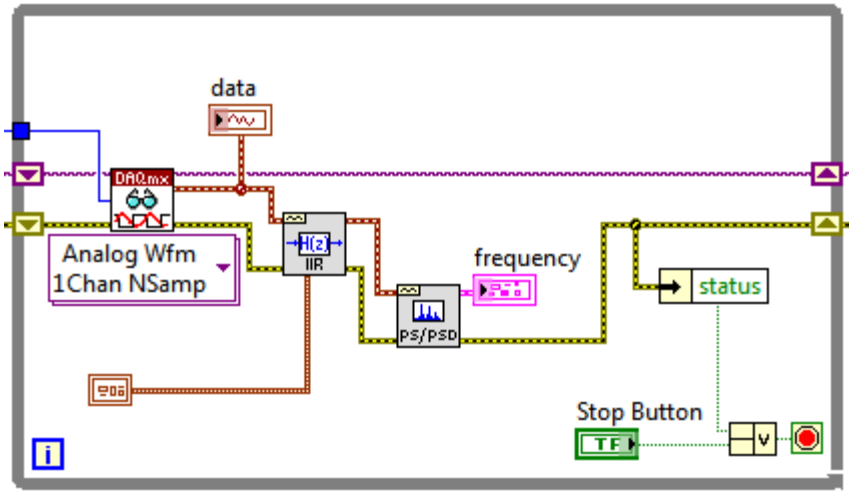
b. Make the following entries into the constants
   i. Topology = Butterworth
   ii. Type = Lowpass
   iii. Order = 10
   iv. Lower Fc = 600

c. Right-click the border to the cluster and select **View Cluster as Icon** to minimize its space on the block diagram.



d. Move the cluster constant below and to the left of the filter. Your Block

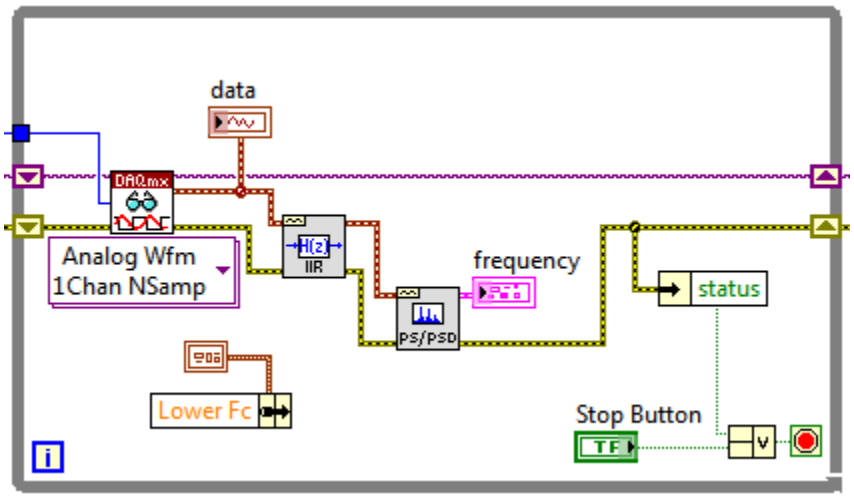
Diagram should appear as follows.



2. To visualize the effect of the filter and to find out the ideal cutoff frequency for the filter, it would be better to be able to control the frequency cutoff while the program is running. With LabVIEW, this is possible!

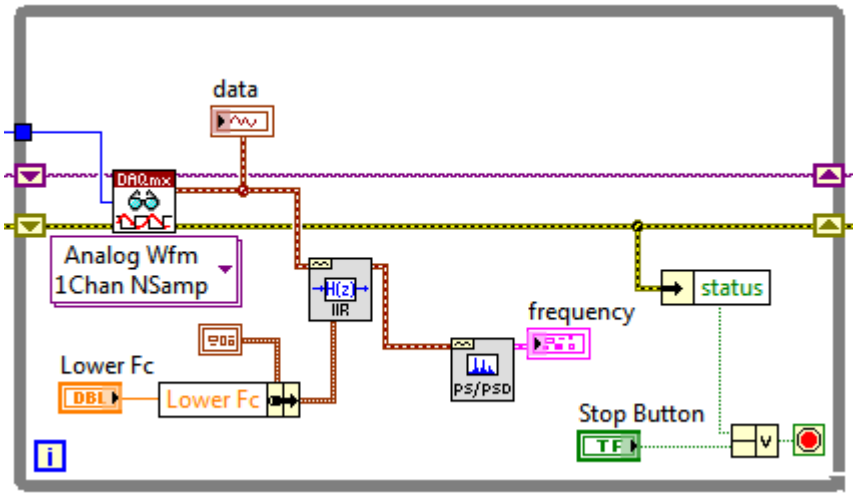
    a. You will need to modify the **Lower Fc** component of the **IIR Filter Specifications**. Place a *Bundle by Name* function below and to the left of the IIR filter function.

    b. Wire the cluster output into the **input cluster** input on the top of the *Bundle by Name* function.



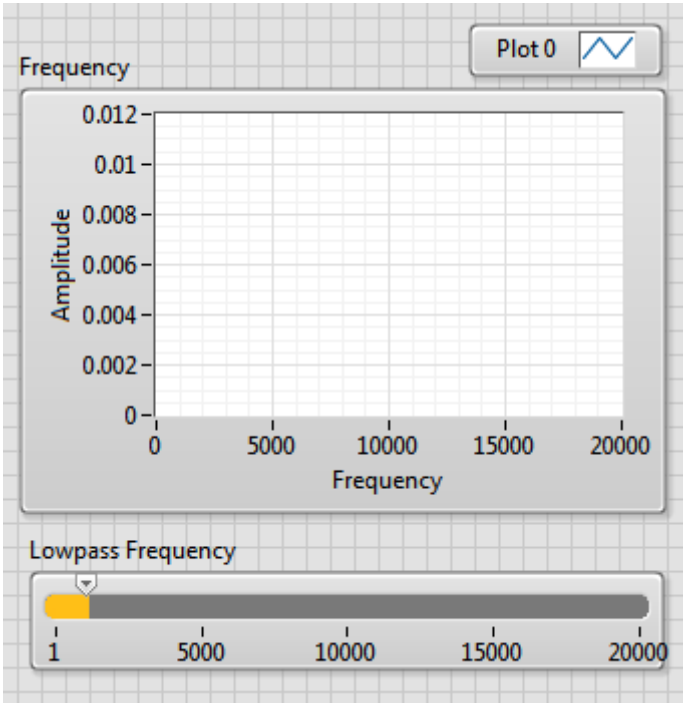
    c. Wire the output of the Bundle by Name function into the IIR Filter Specifications input of the IIR Filter function

    d. Create a control for the **Lower Fc** input of the Bundle by Name function



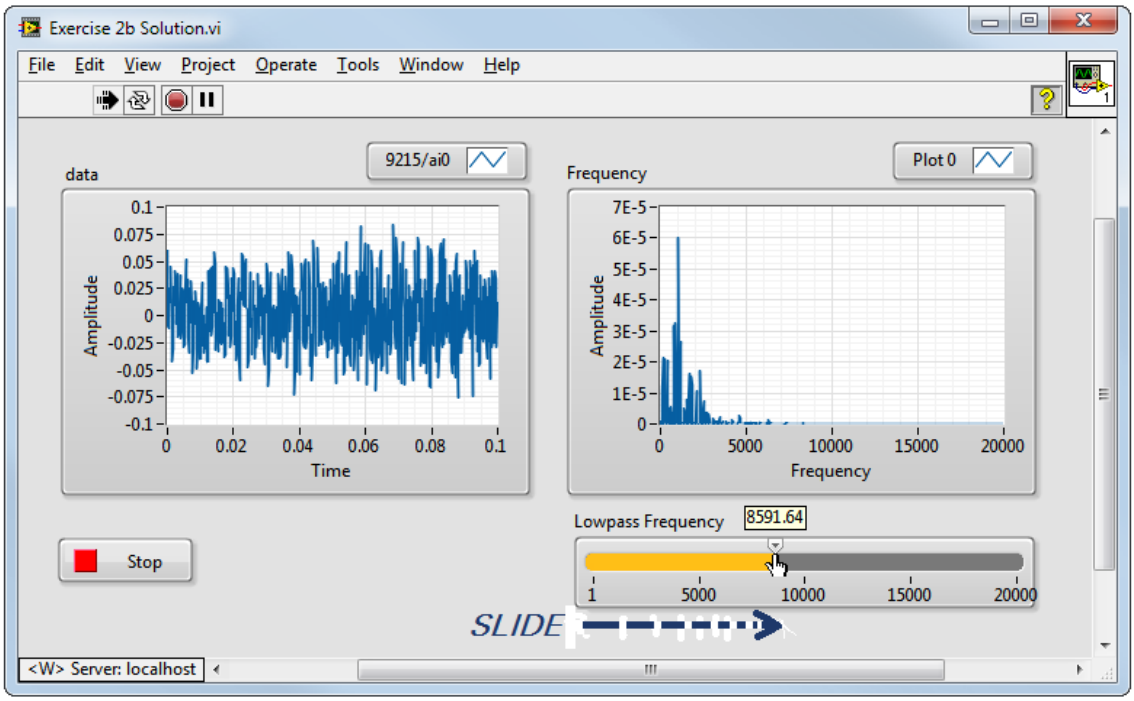
e. Navigate to the Front Panel to locate the Numeric Control and replace the control with a **Silver » Horizontal Numeric Pointer Slide**. Double-click the minimum value and change it to **1** and double-click the maximum value and change it to **20k**.

f. The minimum frequency must always be greater than zero, and the maximum frequency must always be less than or equal to the Nyquist Frequency, which is the ½ of the Sampling Frequency, 40kHz in this example.



3. Save the VI.

4. Run the VI and modify the cutoff frequency while it is running. Can you see the effect on the frequency?



Part C

1. You are now aware of the effect of the filter. For this reason, you no longer need to have the control on the Front Panel.

    a. Delete the *Lowpass Frequency* control and remove the *Bundle by Name* function from the Block Diagram.

    b. Change the Lower Fc value to **600** within the cluster

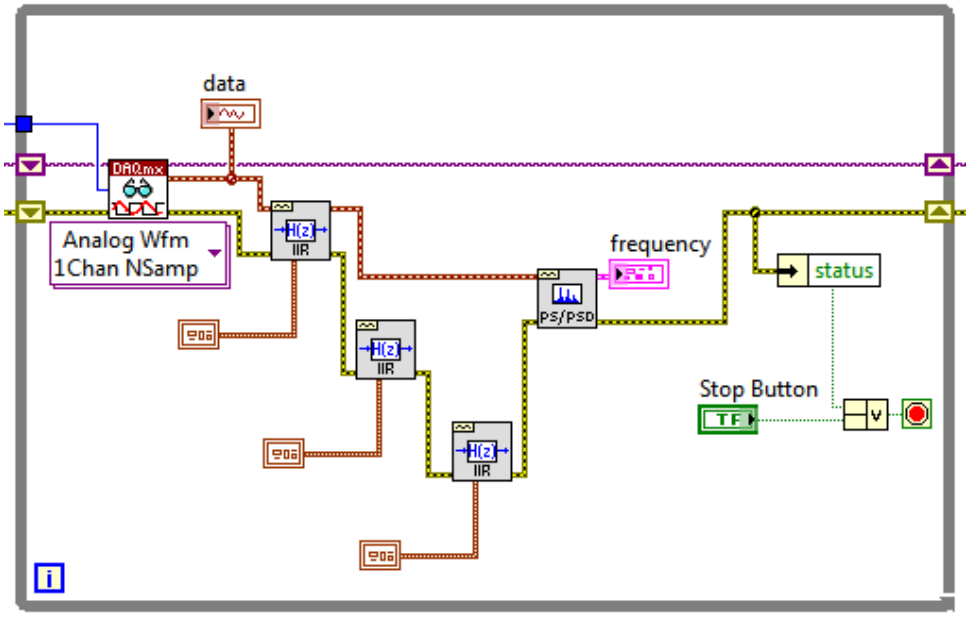
    c. Wire the cluster output back into the **IIR Filter Specifications** input of the IIR filter function.

2. Add in 2 more Digital IIR filters, one bandpass filter for the mid-range frequencies, and one high-pass filter for treble frequencies. You will need to add more space to the While Loop to place two more filters.

    a. Select the entire filter setup by clicking and dragging to select, and then hold <Ctrl> and **click-drag-and-drop** to copy the functions for the additional filters to save time. You can also select multiple items by holding <Shift> while selecting items.

    b. Wire the error out of the Lowpass into the error in of the Bandpass Filter. Wire the error out of the Bandpass into the error in of the highpass. Wire the error out of the highpass into the error in of the Power Spectrum

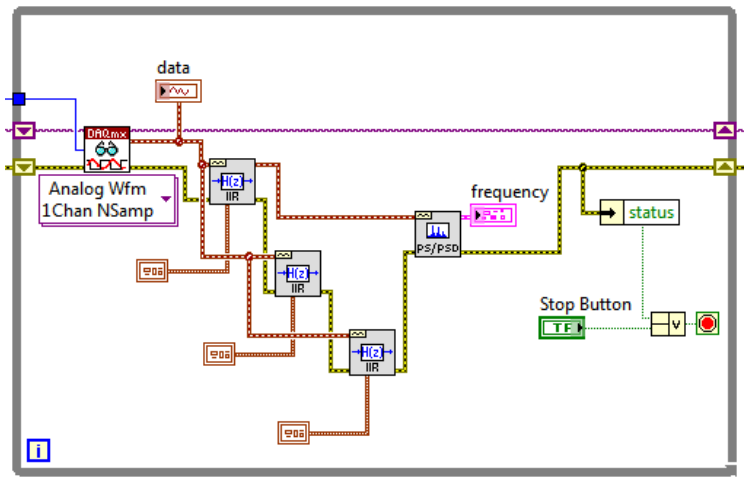
function. You will need to adjust the Block Diagram spacing again.



c. The bandpass Lower Fc must always be greater than zero and less than the Upper Fc of the bandpass filter. Open the cluster by double clicking on it and then set Lower Fc to **800** and Upper Fc to **3000**. Change the type to **Bandpass**.

d. Contrary to intuition, the high-pass filter uses the Lower Fc for the frequency limit, not the Upper Fc terminal. Set this to **5000**. Change the type to **Highpass**.

e. The filters are now cascaded in series for the error wires, which is acceptable. However, we do not want to run the filtering in series; the filtering must be in parallel to allow each filter to view the original sound signal. Branch off of the data output from the DAQmx Read VI and wire this to the signal in terminal of each filter.



3. With the above setup, only the lowpass filtered signal will be displayed. However, the goal of an audio equalizer is to mix the signals back together after filtering them out and attenuating them.

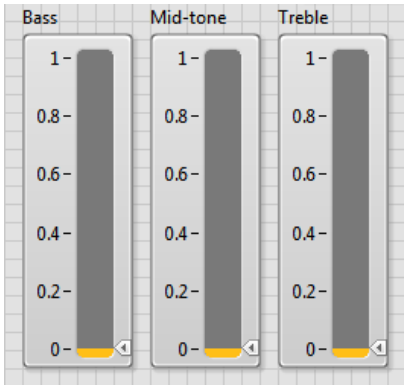
    a. To attenuate the signals, you need to operate on the array of numeric data within the waveform rather than the entire waveform. Use a **Get Waveform Components** for each of the three waveform clusters to unbundle the **Y** array of data.

    b. This is located in **Programming » Waveform** on the Functions Palette. Place 3 of these down and wire the waveform output from each filter into the function.

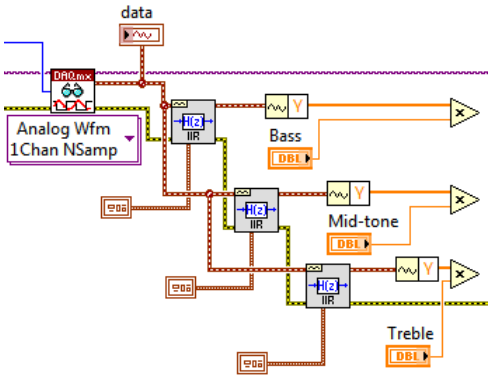
    c. Place a multiply function for each array from the **Programming » Numeric** Functions Palette and connect each one of the three arrays to the top input of the function.

    d. On the Front panel, create 3 Silver Numeric Vertical Pointer controls, one for Bass, one for Mid-tone, and one for Treble, titled respectively.

    e. Set the input range at **0** to **1.0** for each



    f. Place each control inside of the while loop and connect to the respective Multiply functions.



4. To add the arrays back together, use a Compound Arithmetic function as you did to

add the Boolean functions together for the Conditional terminal.

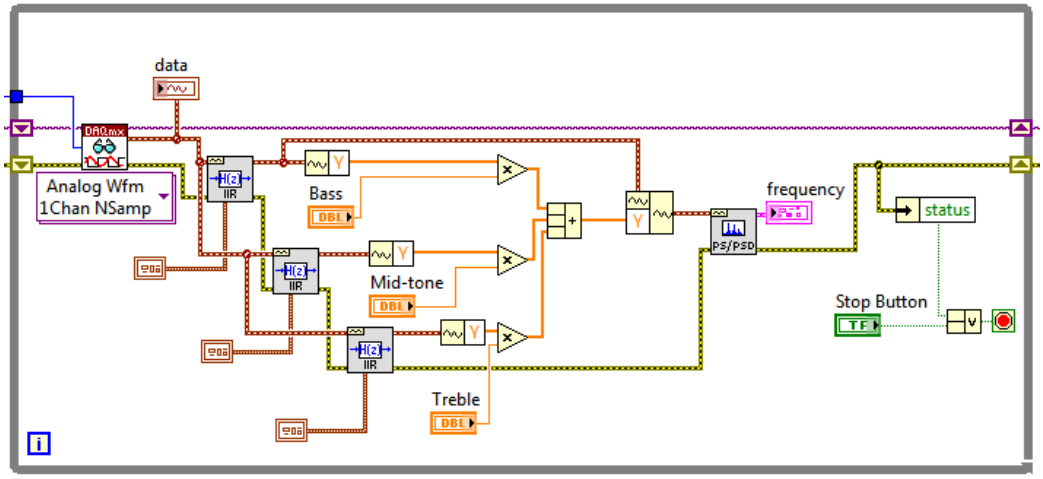
a. You can copy and paste from the Compound OR function and change the mode, or place it from the **Programming » Numeric** Functions Palette.

b. Expand for three inputs, and wire the arrays into it accordingly.

c. To build the arrays back into a waveform, use a *Build Waveform* function and branch off of the previous waveform to use as a reference for the **input cluster** input on the top of the function.

d. Wire the output of the Compound Arithmetic into the **Y** terminal of the Build Waveform.

e. Wire the resulting waveform into the **Power Spectrum** VI. The Block Diagram code within the While Loop should appear as follows.



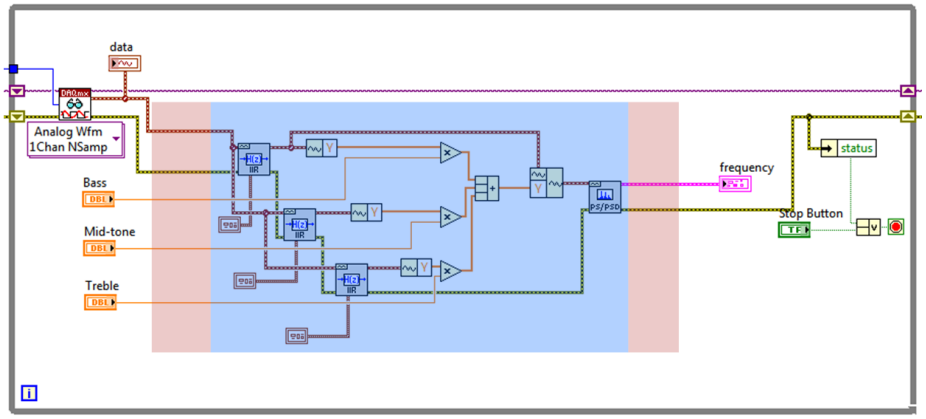
5. Save the VI.

6. Run the VI and set the level of attenuation of the signals on the Front Panel to visualize the effect of the filters.

Part D

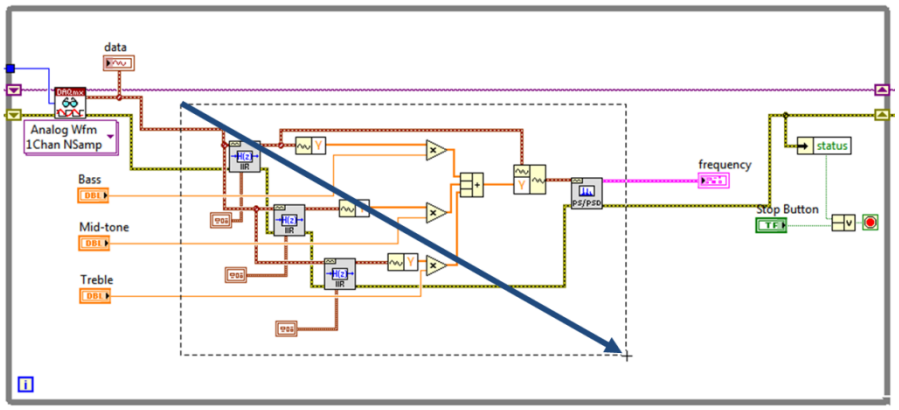
1. At this point, the Block Diagram is getting pretty messy, and it would be in your best interest to try to clean it up. In LabVIEW it is possible to make a subVI, which is a function that you create and use within a larger VI. In this example, we will make a subVI for all of the signal processing.

2. To prepare the Block Diagram for creating a subVI, you need to align the Controls (Inputs) to the left side and the indicators (outputs) to the right. Everything that you wish to conceal within the subVI should be in the center. Here is an example.



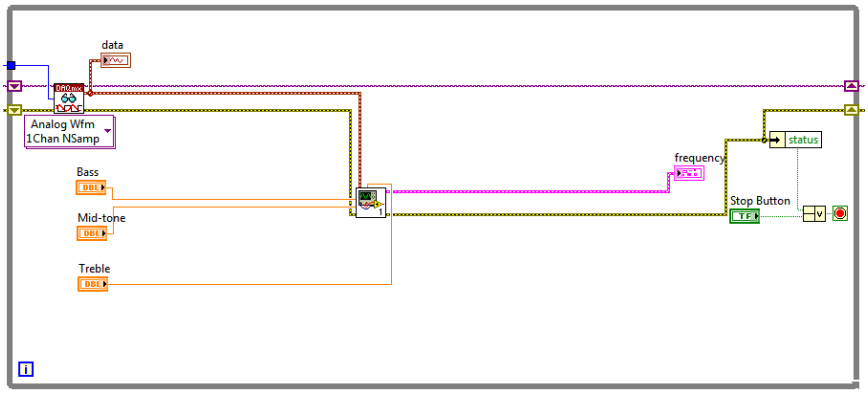
The blue section will be contained within the subVI, and the wires that are enveloped in the red section will be inputs and outputs to the subVI function.

3. To create the subVI, left-click and drag a selection around the entire blue section from the prior image.



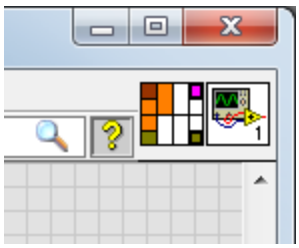
   a. After you have selected the items, navigate to **Edit » Create subVI**.

   b. This will put all the enclosed items within a new subVI and automatically create controls and indicators



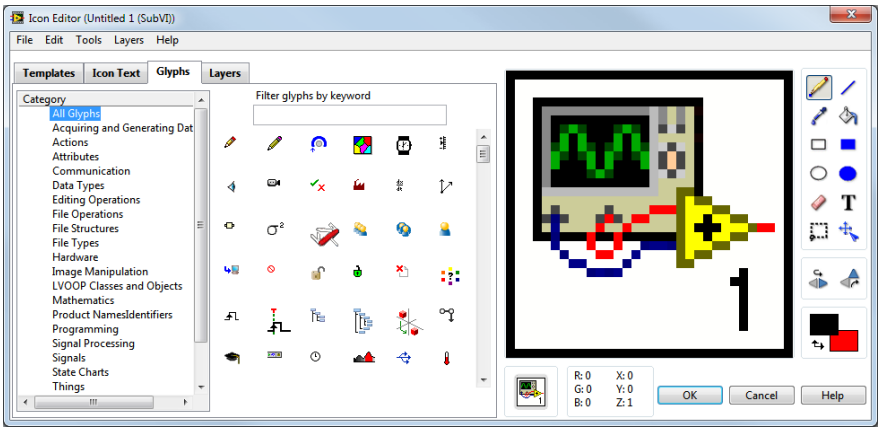
4. You can now proceed by editing the icon for the subVI and changing the wiring if required. Double-click on the subVI icon to begin this process. This will bring up the Front Panel of the subVI.

    a. In the upper right hand corner of the subVI Front Panel, you will find the icon and terminal editor of the subVI.

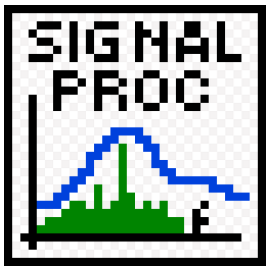


    b. Double-click the default icon to open the icon editor.

    c. Browse around the environment and edit the icon as you desire. It is best to add in text and choose an icon to indicate the functionality of the subVI. You can drag-and-drop glyphs from the library and also copy and paste in images from the internet.



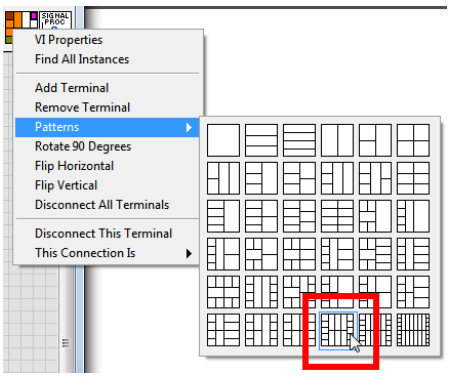
    d. Here is an example of an icon that you could use, but feel free to create your own. Select **OK** once you are satisfied.



    e. Now let's change the terminal pattern and wiring. It is best to keep the inputs on the left and outputs on the right. The top and bottom are typically

used for specifications and optional inputs. This means that we need to have 5 inputs on the left and move the Treble control terminal to the left-hand side.

f. Right-click the terminal editor and select **Patterns »** and then select the pattern with 5 inputs, 5 outputs, and 3 top and 3 bottom terminals.



g. You now need to change the terminals that are mapped to controls. Right-click the lower orange input which corresponds to mid-tone and select **Disconnect This Terminal**.

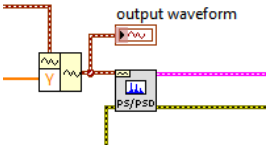
h. No you will need to reconnect the Mid-tone control to the middle input terminal. Left-click the middle input. Notice that it changes the mouse pointer to the wiring tool.

i. You will now need to left-click on the control that you wish to be mapped to this terminal, Mid-tone.

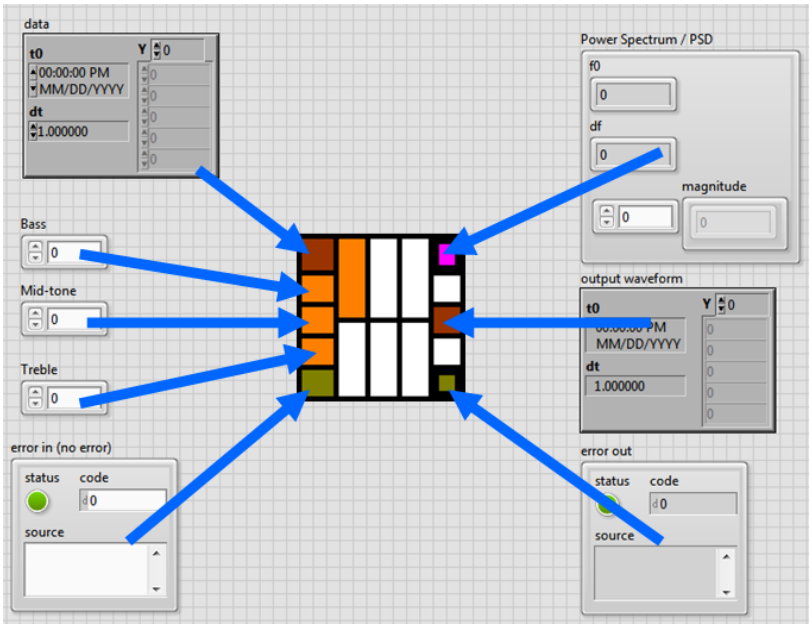
j. You can left-click any terminal that is already assigned to a control and it will highlight the associate control.

k. Repeat this process to map the Treble control to the input terminal just above the error input and just below the mid-tone input.

l. Return to the Block Diagram and right-click on the waveform wire that is input to the Power Spectrum function and create an indicator. Map this indicator to the middle output terminal.



m. Here is the how all of the controls and indicators should be mapped.



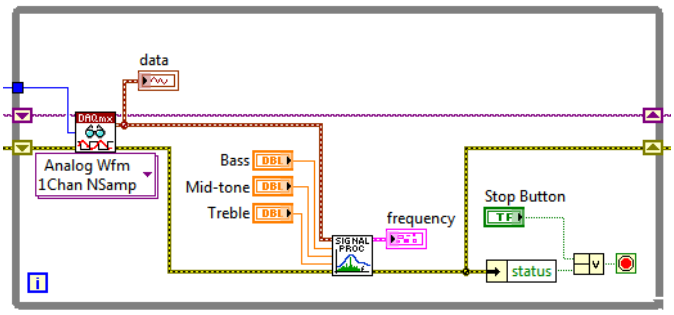
n. Save the VI as **Signal Processing subVI** within the folder on your desktop.

o. Close the subVI.

5. Open the Block Diagram of your main code and notice that the subVI is now greyed-out. This is because you need to re-link to the subVI after changing the input terminals.

a. Right-click the subVI and select **Relink to subVI.**

b. Verify that all connections are now correct. Clean up the Block Diagram and condense the size of the while loop. Here is the resulting Block Diagram after cleaning it up.



6. Save the VI. Run the VI to verify that it still functions correctly.

<Exercise Continued on Next Page>

Exercise 3: Output Processed Signal

Goals

- Create an Analog Output Task to play a simple tone
- Modify code to continually generate and output a new waveform
- Add this code into the Analog Input code and then output the filtered Audio Signal
- Modify code to allow for two analog inputs and two analog outputs because it is a stereo signal with Right and Left audio signals

Part A

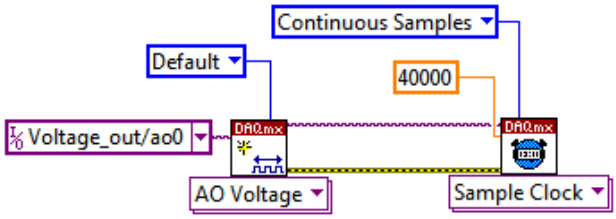
1. Create a new VI.

2. On the block diagram of your new VI, create an Analog Output Voltage DAQmx Virtual Channel by placing a DAQmx Create Virtual Channel and then selecting the type to be **Analog Output » Voltage.**

    a. Set the input to be **Voltage_out/ao0**
    b. Set the input terminal configuration to be **default**

3. Create a DAQmx Timing VI

    a. Set the rate to be **40k**
    b. Set the sample mode to be **Continuous**



4. By default, Analog Voltage output tasks are setup to have a waveform pre-loaded into the output buffer before starting, and then continuously regenerate the waveform until the DAQmx Task is cleared. So you now need to create a waveform.

    a. Navigate to the Functions Palette and browse to **Programming » Waveform » Analog Waveform » Waveform Generation » Basic Function Generator** and place it above the Sample Clock Timing VI.

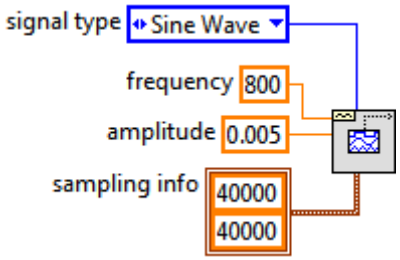
    b. Browse the inputs using Context Help **<Ctrl+H>**

    c. Place a constant for **frequency**, **amplitude**, **signal type**, and **sampling info**.

    d. Enter **800** for frequency, **0.005** for amplitude, and **Sine Wave** for signal type. For the sampling info, enter **40k** for the Sample Rate (Fs) and **40k** for Number of Samples (#s). You can mouse over the items in a cluster to view their label in context help, but in this case it does not matter as they are the same value.

    e. This function will now generate a one second long 800 Hz tone that is

sampled at 40kHz.



5. Create a DAQmx Write VI and place to the right of the Sample Clock VI

    a. Connect the Task and Error wires accordingly form the Sample Clock VI

    b. On top of the VI there is a terminal, **Auto-start**, which will start the task automatically after the buffer is loaded. Create a constant for this input and select **F** to prevent it from auto-starting.

    c. Wire the **signal out** from the Basic Function Generator VI to the **data** input of the DAQmx Write VI

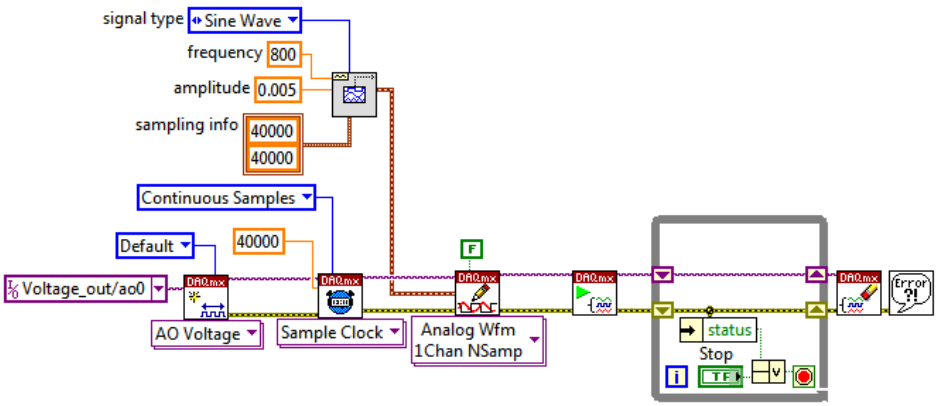
6. Place a DAQmx Start Task after the DAQmx Write

7. Create a small while loop to the right of the Start Task VI and pass the task and error wire through to make tunnels.

    a. Create a Stop Button control, and also unbundle the error wire to stop the loop if an error is detected.
    b. Replace the tunnels with shift registers

8. Clear the task and place a simple error handler which is located under **Programming » Dialog & User Interface**.

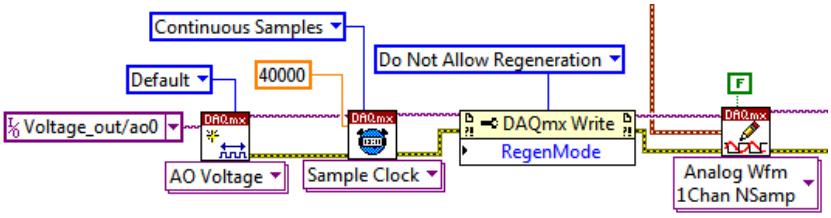
9. This VI will now output an 800Hz tone continuously until the Stop button is pressed. Run the VI to test it out.



Part B

1. To update the output waveform while running, you will need to change the default behavior of allowing regeneration.

    a. Right-click the DAQmx Write VI and navigate to **DAQmx – Data Acquisition Palette » DAQmx Write Property Node** and place this property node in-between the Sample Clock and DAQmx Write VIs
    b. Connect the Task and Error wires accordingly
    c. For the property node, select **Regeneration Mode** for the property of interest and create a constant. Select **Do Not Allow Regeneration**.

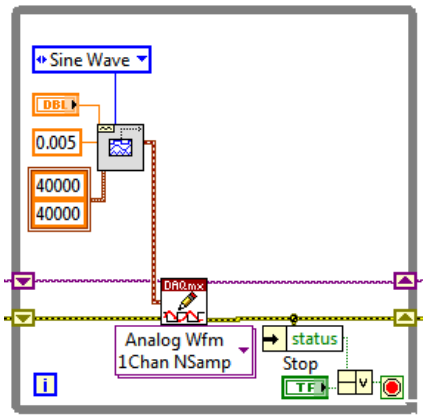


2. Increase the size of the While Loop to allow for a copy of the Basic Function Generator VI and all constants.

    a. Copy and paste within the while loop the Function Generator.
    b. Change the Frequency constant to a Control.
    c. Navigate to the Front Panel and change the numeric control with a Silver numeric horizontal pointer slide control.
    d. Set the limits for the control at **1** to **20k**.

3. Copy and Paste the DAQmx Write into the While Loop and wire accordingly to the Task and Error wires.

    a. Set auto-start to **F** if it is not already
    b. Wire the **Signal out** of the Function Generator inside the loop to the **data** in of the DAQmx Read VI.

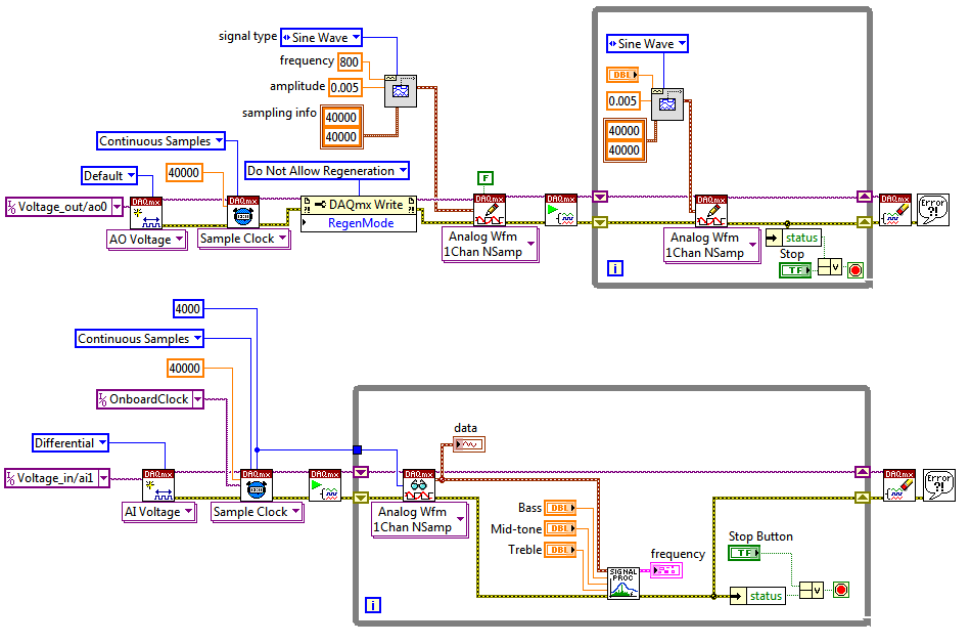


4. Run the VI and modify the frequency during run time while listening to the output waveform.

Part C

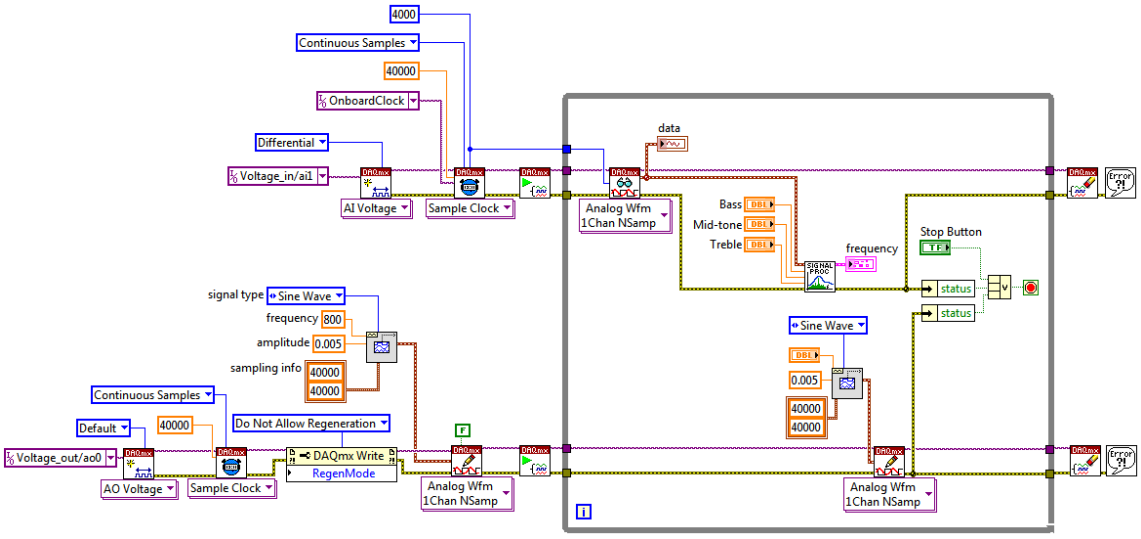
1. To use the analog output code to play back the processed audio input, you will need to add your new analog output code to the existing VI that you created to read sound in.

    a. Copy the entire Block Diagram Code of your analog output VI and paste into the Block Diagram of the Analog output as seen in the figure below. You can also select the code from one Block Diagram and drag-and-drop onto the other.



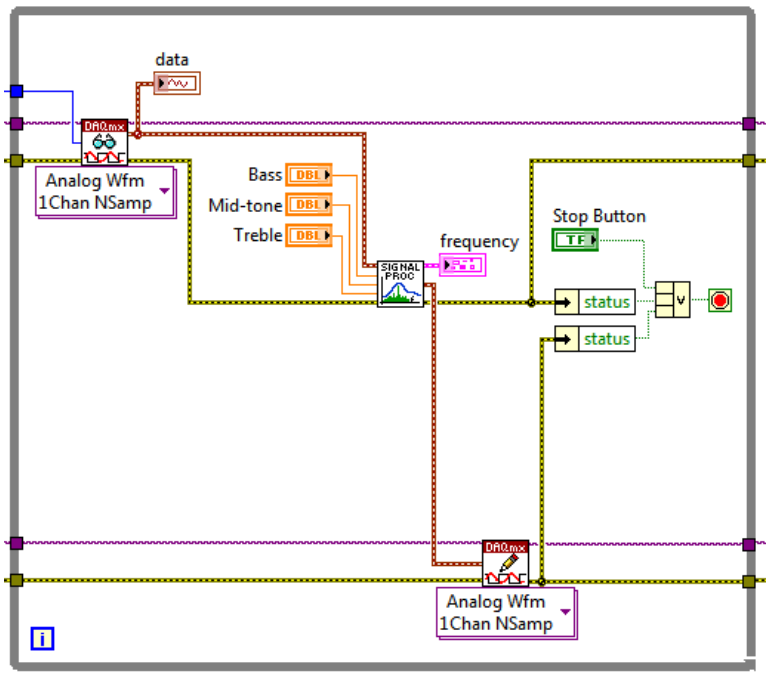
2. The goal is to now merge the two while loops together. Because the acquisition and generation are linked together and data needs to be shared from one to the other, they must be in the same while loop.

    a. You have two options: remove one loop and add code into the other, or remove both and draw a new while loop.
    b. The result is the same, and it is only a matter of preference. You will also need to remove one Stop button as you no longer need both.
    c. Your block diagram should appear as follows.



3. Remove the Function Generator VI and all of its connected constants and controls inside of the While Loop.

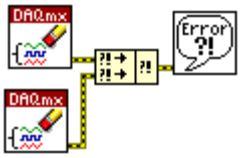
4. Wire the **Output Waveform** terminal of the Signal Processing subVI to the **data** input terminal of the DAQmx Write VI.



5. Change the amplitude on the Function Generator before the while loop to **zero (0)** so that it is a 1 second delay with no sound as opposed to a 1 second tone at the beginning of the initial output.

6. Delete one of the Simple Error Handler VIs as it is a best practice to only have one error handler per VI.

a. Place a Merge Errors function to merge the two error wires together by navigating to **Programming » Dialog & User Interface**
b. Connect the Error output from each DAQmx Clear Task into the inputs of the merge error function, and wire the output of this function to the input of the Simple Error Handler.



7. Save the VI.

8. Run the VI with your MP3 source playing. Toggle the attenuation and filter frequencies. Can you hear the effect?

<End of Exercise>

Measuring Vibration from an Accelerometer

Goals

- In the first part of this exercise, you will write a program to control the speed of a fan and acquire data from accelerometers attached to the fan
- For the second part, you will add code analyze the vibration signal in both LabVIEW and MathScript
- Key concepts include –
    - Accelerometer measurements
    - Using the NI-DAQmx API
    - Using SubVIs
    - Combining measurement and control tasks
    - Processing data with LabVIEW and MathScript

Part A **Measuring Vibration From an Accelerometer**

Estimated time: 60 minutes

Many sensors for measuring acceleration and pressure are based on the principle of piezoelectric generation. The piezoelectric effect denotes the ability of ceramic or quartz crystals to generate electric potential upon experiencing compressive stresses. These mechanical stresses are triggered by forces such as acceleration, strain, or pressure.

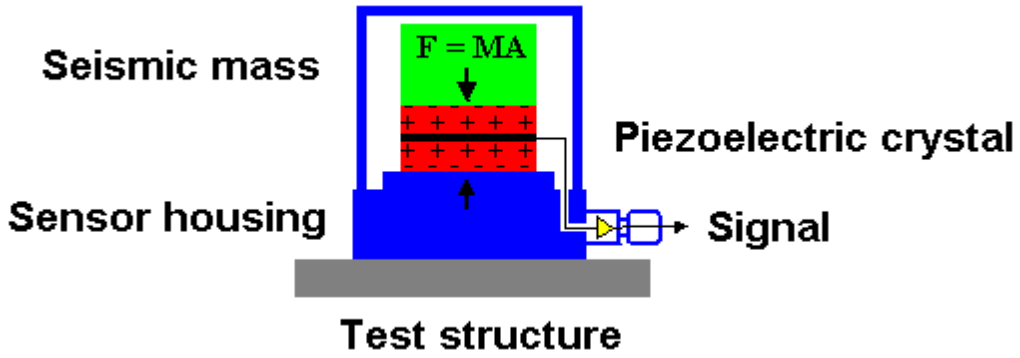


Figure 1. An accelerometer uses a seismic mass to create a small electrical signal based on piezoelectric generation.

In the case of microphones, acoustic pressure waves cause a diaphragm, or thin membrane, to vibrate and transfer stresses into the surrounding piezoelectric crystals. Accelerometers, on the other hand, contain a seismic mass that directly applies forces to the surrounding crystals in response to shock and vibrations. The voltage generated is proportional to the internal stresses in the crystals.

A particular class of piezoelectric sensors, known by the term integrated electronic piezoelectric (IEPE), incorporates an amplifier in its design next to the piezoelectric crystals. Because the charge produced by a piezoelectric transducer is very small, the electrical signal produced by the sensor is susceptible to noise, and you must use sensitive electronics to amplify and

condition the signal and reduce the output impedance. IEPE therefore makes the logical step of integrating the sensitive electronics as close as possible to the transducer to ensure better noise immunity and more convenient packaging. A typical IEPE sensor is powered by an external constant current source and modulates its output voltage with respect to the varying charge on the piezoelectric crystal. The IEPE sensor uses only one or two wires for both sensor excitation (current) and signal output (voltage). The NI 9234 C Series module in your NI CompactDAQ chassis provides this constant current source to IEPE accelerometers and microphones.

In this exercise, you will measure the vibration of a fan using the accelerometers on the Sound and Vibration Signal Simulator. The NI 9234 features four simultaneously sampled channels and can measure signals from both IEPE and non-IEPE accelerometers and microphones.

Before you start this exercise, confirm that the accelerometer is properly connected and you are able to acquire basic data from the channel connected to the accelerometer.

1. Ensure that the green **Power** LED and amber **Ready** LED are lit to confirm that the chassis is connected over USB and powered on.

2. Examine the wiring into the NI 9234 to confirm that the BNC cables are connected to the X Acceleration, Y Acceleration and Tach Out on the *Sound and Vibration Signal Simulator*.

3. Open Measurement and Automation Explorer by selecting **Start » All Programs» National Instruments » Measurement and Automation Explorer** or double-click the icon on your desktop.

4. Right click on the NI 9234 and select **Test Panels**.

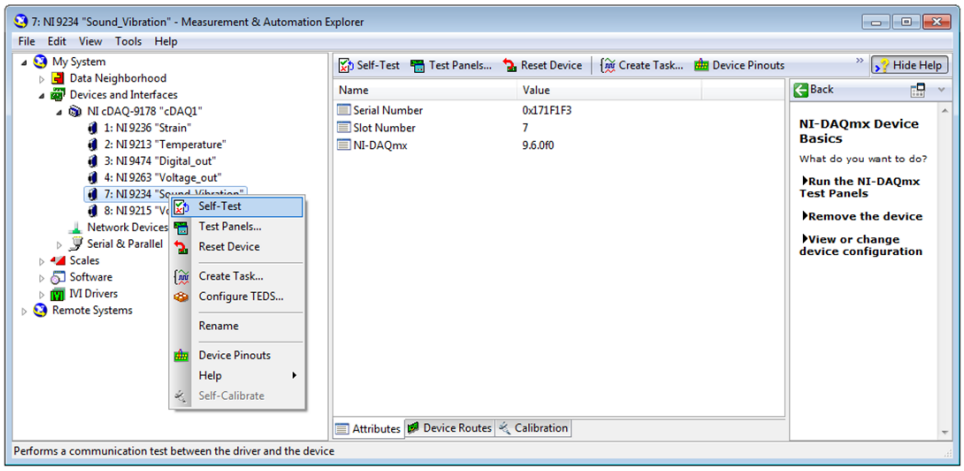


Figure 2. Select Test Panels to open a simple utility for checking signal connectivity.

5. When the test panel opens, change the **Samples to Read** to 2560, change the **Mode** to Continuous, select the button next to **IEPE Enable**, and change the **Coupling** type to

AC, as shown in Figure 3.

6. Press the Start Button.

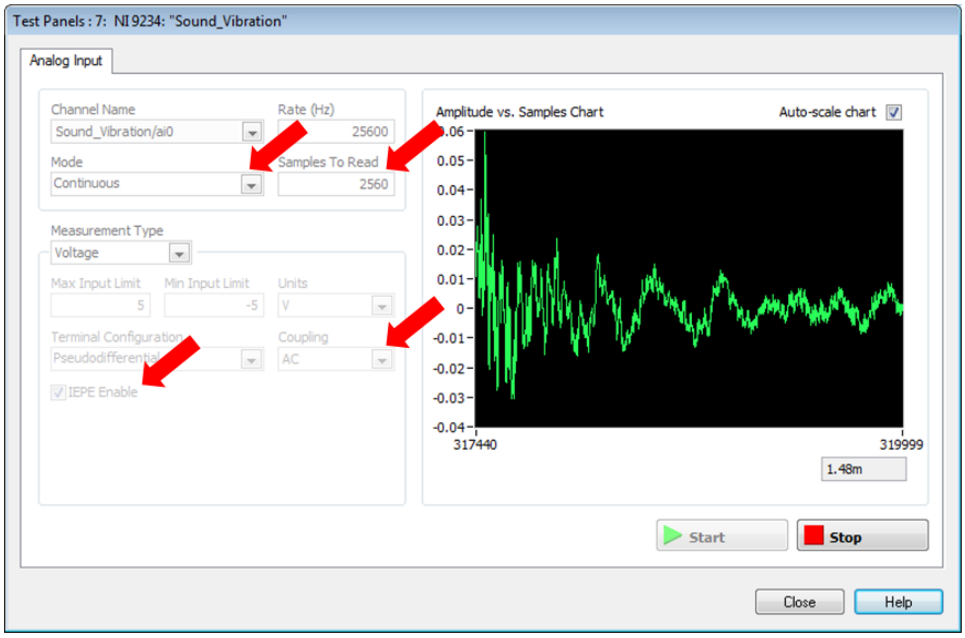


Figure 3. The test panel is used to confirm electrical connections.

7. Gently tap the side of the box to ensure that your sensor reacts to your input.

8. Press the **Stop** button and close the Test Panel.

Every time you set up a new measurement or measurement system, it is a good idea to confirm that the wiring is correct and that all software is installed and working correctly. MAX provides the insight into your system and setup to help you eliminate mistakes early in your development process.

9. After confirming that your sensors are working, minimize MAX and select **Start » All Programs » National Instruments » LabVIEW <Year>** to open LabVIEW.

10. Select **File » Open** and navigate to the hands-on directory. The starting point for this exercise is located at Hands-On Exercises\Vibration\Vibration Project.

11. Double click **Vibration Exploration.lvproj** to open the accelerometer measurement project. You should see the following screen:

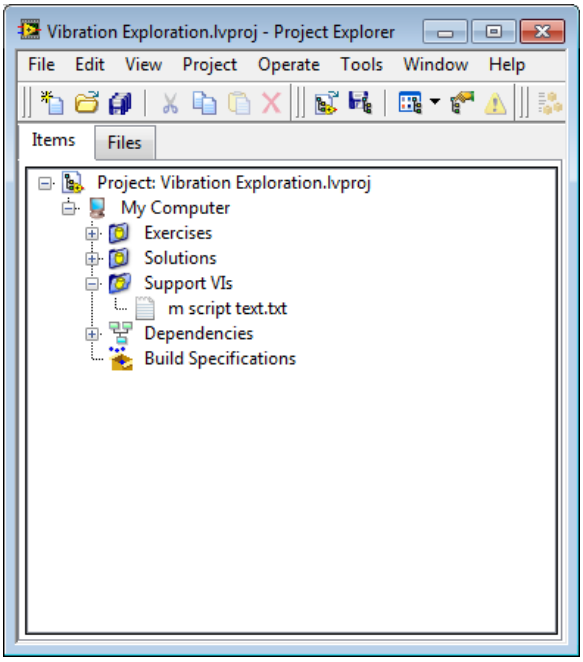


Figure 4. Use the LabVIEW Project to organize all your VIs.

With the LabVIEW Project you can easily organize and visualize all the files that are important to your application. In this particular project, the folders like Exercises, Solutions, and Supporting VIs, have been set to auto-populate with the files within them.

12. Expand the folders to view the file within them. You should see the following VIs:

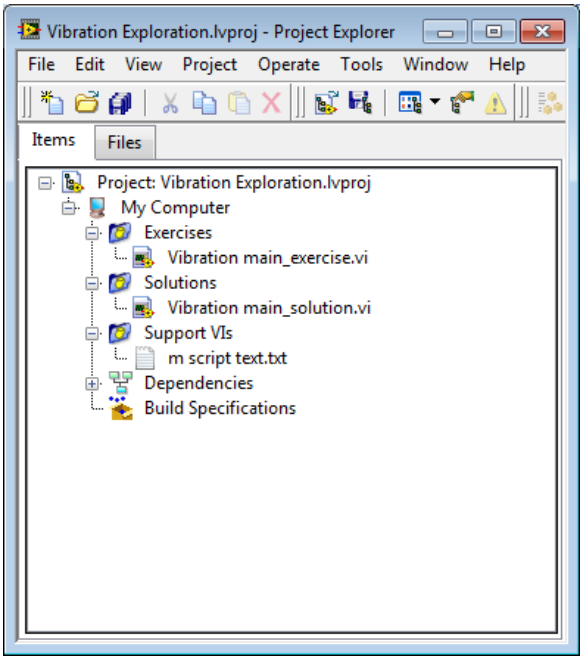


Figure 5. The VIs used in your project can be found in the auto-populating folders.

13. Start by opening the solution – expand **My Computer » Solutions** and double click

Vibration main_solution.vi.

14. Through the rest of this exercise you will build the VI in Figure 6. Ensure that the switch on the Sound and Vibration Signal Simulator is flipped to BNC. Press the run button.

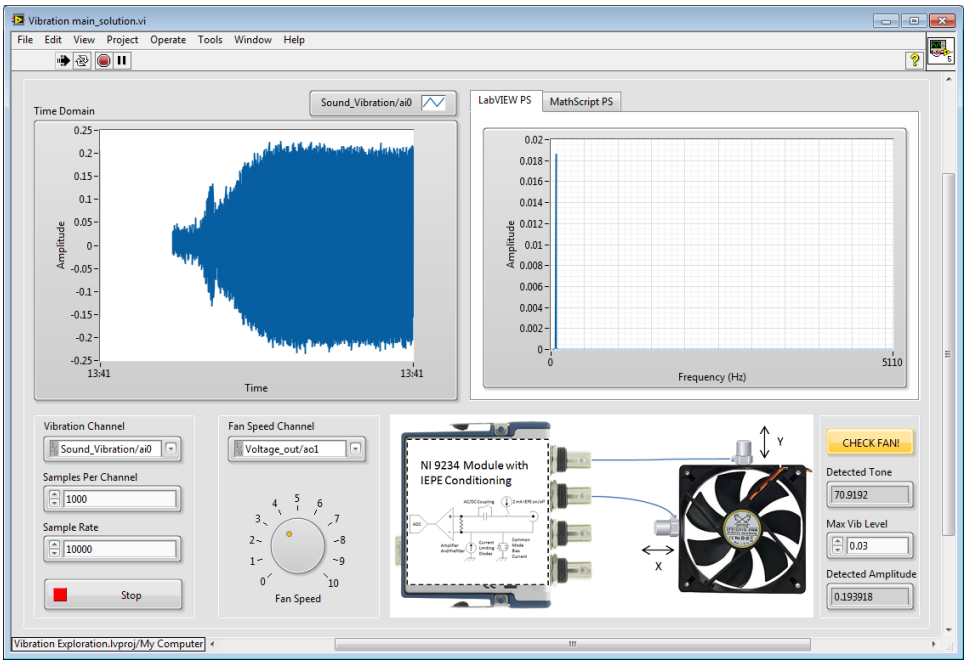


Figure 6. This VI will acquire a time-domain vibration signal and calculate the frequency content using both LabVIEW VIs and MathScript.

15. Using the knob in the center of the screen, adjust the speed of the fan and notice the reaction in both the time domain and frequency domain. Frequency domain analysis is crucial to many types of monitoring applications.

16. Click the MathScript PS tab and notice that the frequency domain information is exactly the same.

17. Press **Stop** and close the solution without saving.

18. To start the exercise, double-click **Vibration main_exercise.vi**. The Front Panel has already been built for you.

When starting a new program, beginning with the Front Panel design is a good way to organize your thoughts around the inputs and outputs that you expect. Once you understand how you want a user to interact with the code, then you can begin writing the VI to support that functionality.

Throughout this exercise, you will complete this VI to control the speed of a fan and acquire data from an accelerometer in the Sound and Vibration Signal Simulator. Additionally, you will use both LabVIEW and MathScript to calculate the frequency components of the acquired signal.

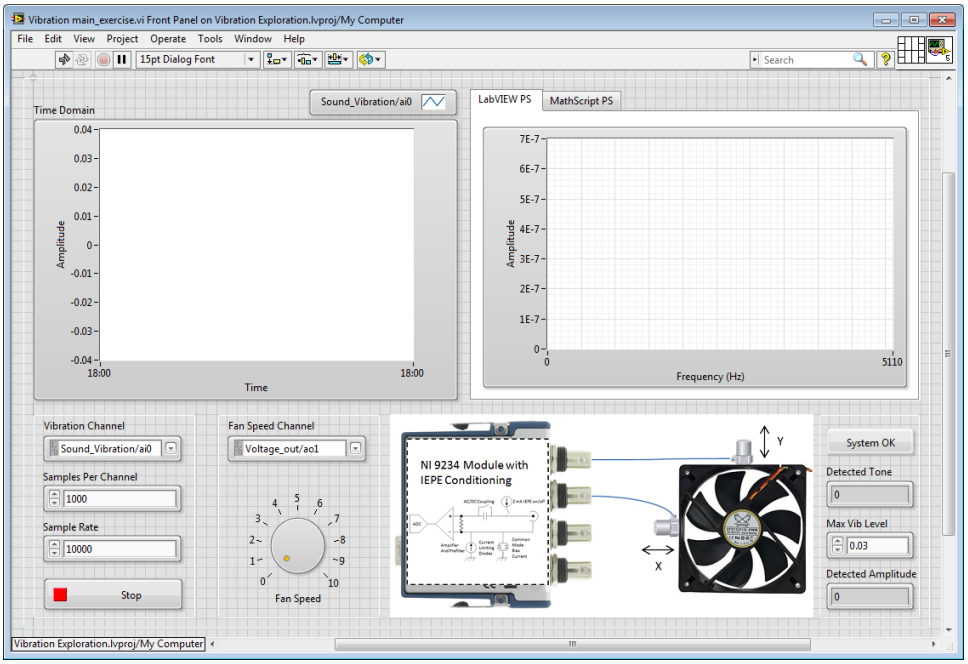


Figure 7. Beginning with the Front Panel design is a good way to organize your thoughts around the inputs and outputs that you expect.

19. Notice that the run arrow in the top right corner is broken ( ). This is because the code contains errors, or in this case, is not in a running state. LabVIEW continuously compiles in the edit environment, so you will always know if your code is able to run. Pressing the arrow when it is broken will show a list of errors. Try pressing the run arrow. If your code compiles with no errors, the run arrow will no longer be broken ( ).

20. Close the Error Dialog and select **Window » Show Block Diagram**. The keyboard shortcut for this is Ctrl + E. Use this shortcut to toggle back and forth between the Front Panel and Block Diagram. When the Block Diagram opens, you will see the following code:

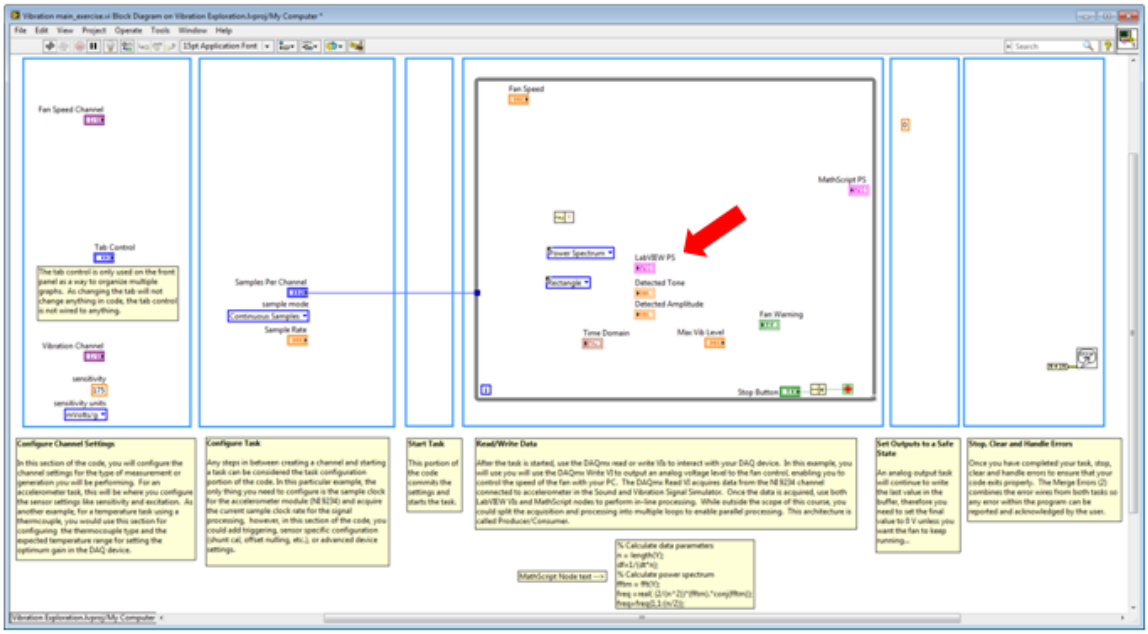


Figure 8. The Block Diagram does the work of your program. This is where you will write the code to control the Front Panel.

Because the code has been started, this Block Diagram includes some basic elements already. Each element on the Front Panel has a corresponding element on the Block Diagram. To find the corresponding Front Panel element, you can double click the element on the block diagram to highlight it. Try double-clicking on the **LabVIEW PS** icon. Then double click on the same item on the Front Panel. This is a convenient way to find items on either the Front Panel or Block Diagram.

The light yellow boxes are annotations. You can create notes on the block diagram simply by double clicking. It's always a good idea to properly document your code so you (if you come back a week later) or someone else can understand what is going on.

21. Now that you are familiar with the elements of the program, you can start building the code to acquire strain data. To start, make sure that the Block Diagram is visible.

22. For this exercise, a large part of the coding will come from the DAQmx palette. To access this palette right click on any white space on the Block Diagram and navigate to **Measurement I/O » NI-DAQmx**, like in Figure 9.

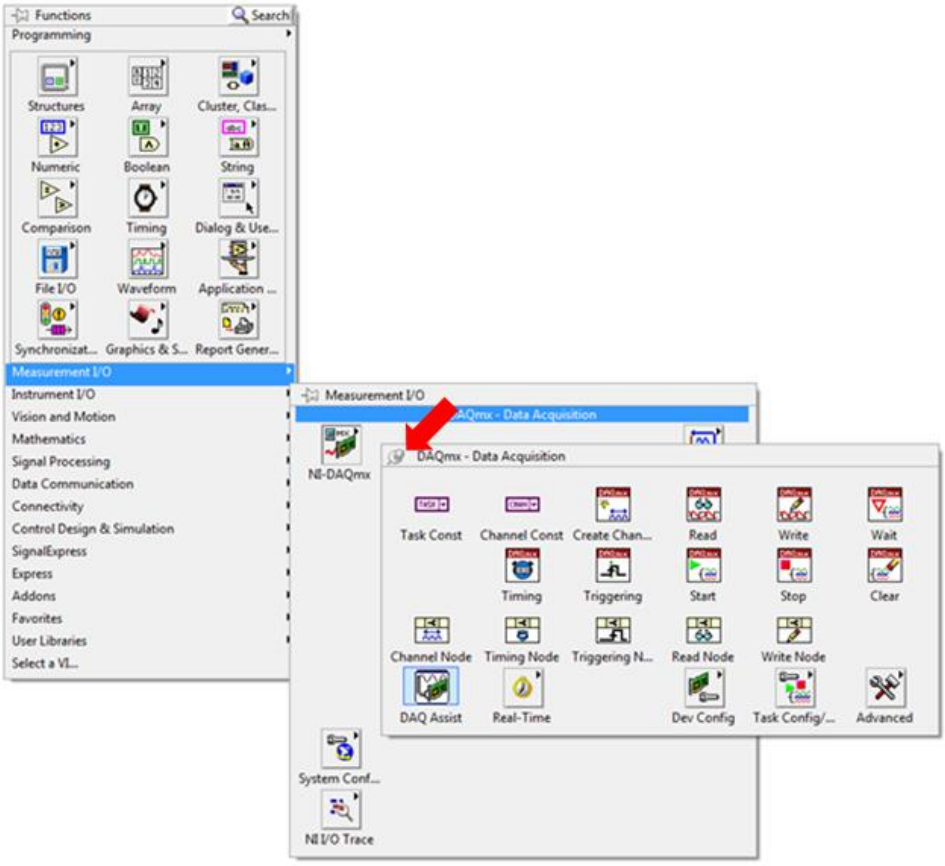


Figure 9. The DAQmx palette includes all the required VIs for communicating with your NI CompactDAQ chassis.

23. Click on the pin to keep this palette visible (red arrow in Figure 9).

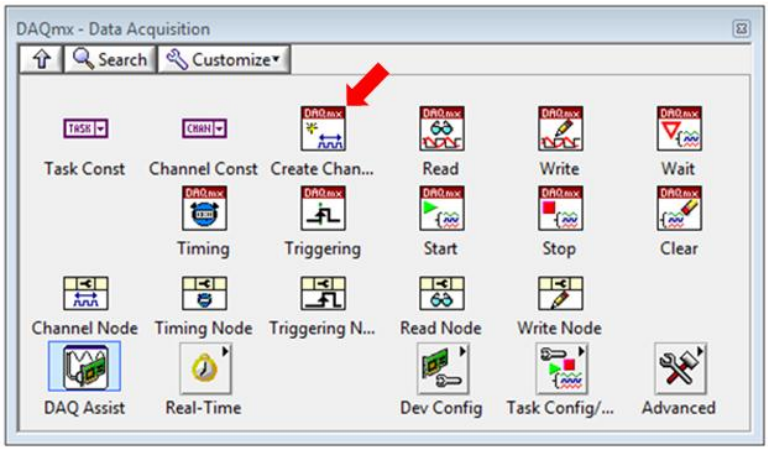


Figure 10. Pinning a palette enables you to click elsewhere in your code with the palette still visible.

24. To start, drag the **NI-DAQmx Create Channel** (red arrow in Figure 10) and place it in the position shown in Figure 11.

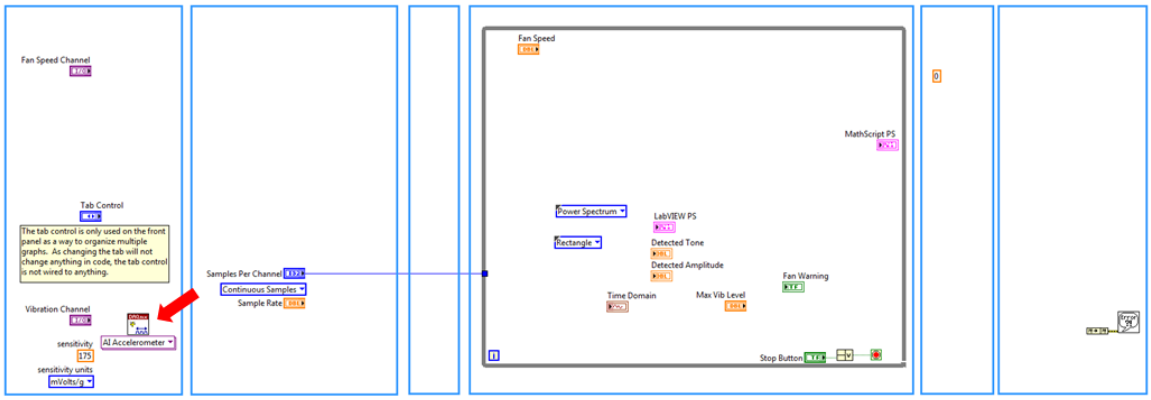


Figure 11. The DAQmx Create Channel VI establishes communication with your DAQ device.

25. By default, this VI is configured to create a simple analog input voltage. To configure this for accelerometer measurements, click on the drop down list underneath the VI and select **Analog Input » Acceleration » Accelerometer**.

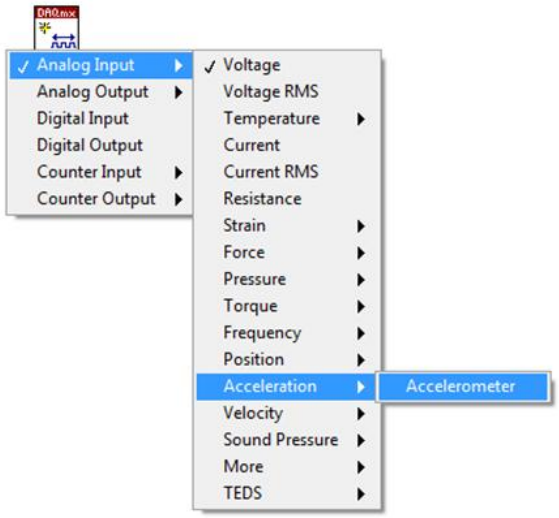


Figure 12. The DAQmx Create Channel VI is called polymorphic because it changes to adapt to the selected measurement type.

26. To find out more about this VI, bring up the Context Help by pressing Ctrl+H.

The Context Help gives you a brief description of anything you hover over with your cursor. Hover over the DAQmx Create Channel VI to see the inputs and outputs of the VI.

For an accelerometer channel, we need to provide the channel list you are measuring from and the sensor specific information. To provide these inputs, wire up the VI as shown in Figure 13.

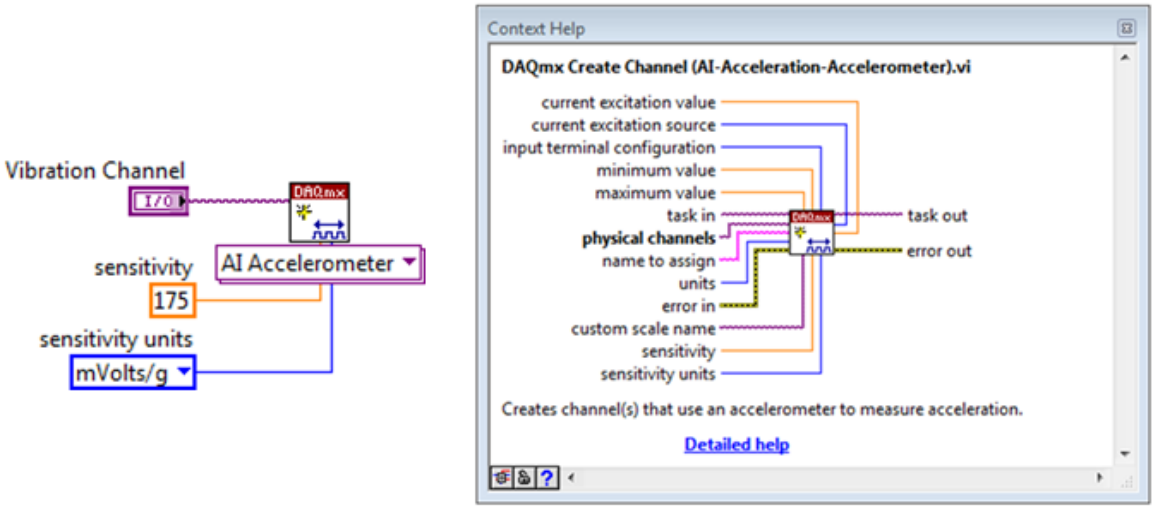


Figure 13. Wire the inputs to the DAQmx Create Channel VI.

27. Drag the following VIs from the DAQmx palette and place them as shown in Figure 14.

    a) DAQmx Timing

    b) DAQmx Start Task

    c) DAQmx Read

    d) DAQmx Stop Task

    e) DAQmx Clear Task

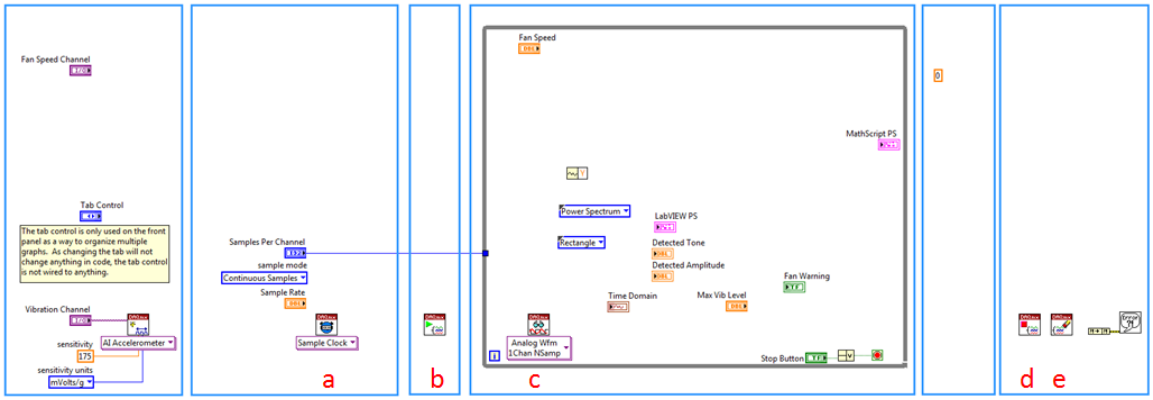


Figure 14. The flow of DAQmx code almost always follows the same pattern

If you recall from the example programs you examined in the earlier exercises, the DAQmx pattern almost always follows the same flow. A channel is created, settings like timing and triggering are configured, the task is started, the channel is read then the task is stopped and the channel is cleared.

28. The DAQmx Timing VI configures the sample rate, sample mode, and clock source for your task. You are going to be sampling continuously, using the on-board clock source.

Wire the DAQmx Timing VI as follows:

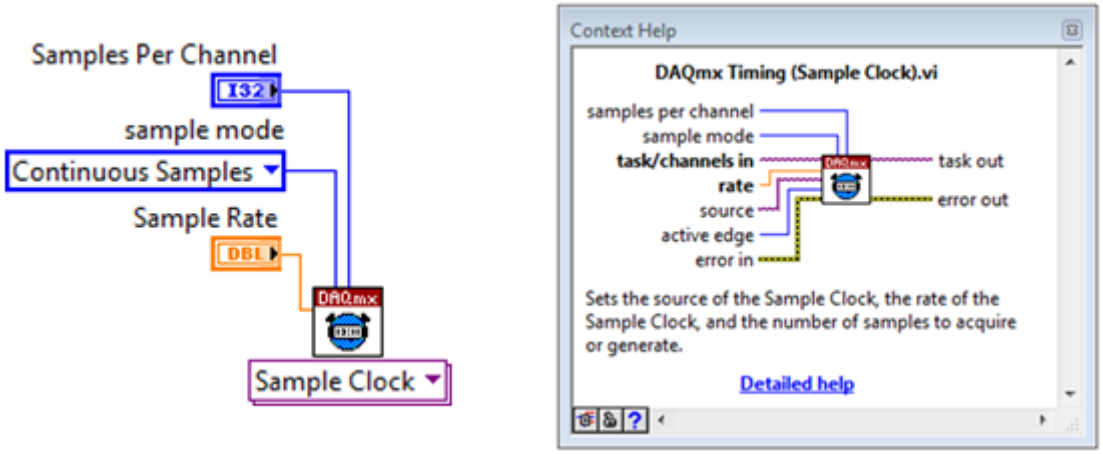


Figure 15. The DAQmx Timing VI configures your clock parameters.

29. The DAQmx Start Task VI transitions the code to the running state. Once your code is past this VI, you are ready to read or write data. To learn more about this VI, make sure the Context Help is still visible (Ctrl+H) and hover over the DAQmx Start VI.

30. After the task is started, the DAQmx Read VI pulls data off the DAQ device buffer. Like the DAQmx Create Channel VI, the DAQmx Read VI is polymorphic. To configure this VI to read a data from the strain gage, click the drop-down arrow and select **Analog » Single Channel » Multiple Samples » Waveform**.

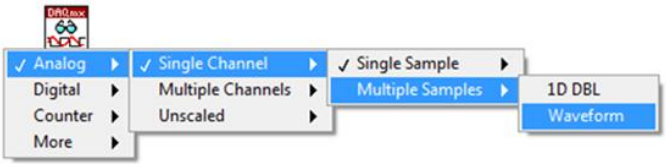


Figure 16. Select a single channel of waveform measurement to read from the supplied accelerometer.

31. Wire the data output of the DAQmx Read VI to the Time Domain chart. This will enable a user to visualize the data on the front panel. Also, wire the waveform wire to the Get Waveform Components – you will use this function in a later part of this exercise.

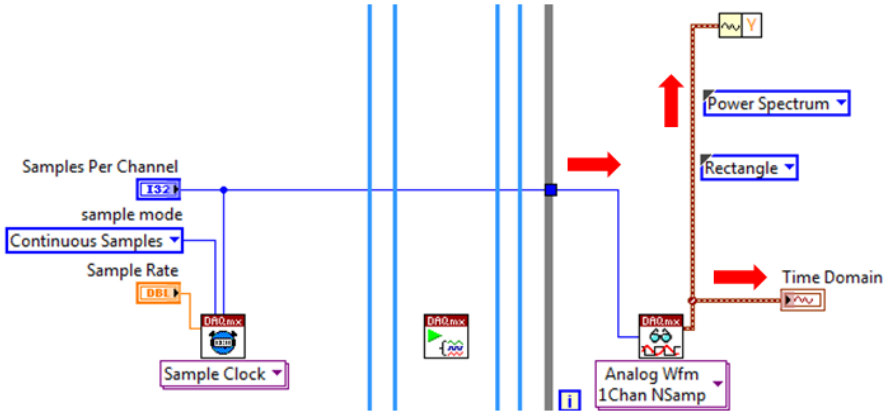


Figure 17. Get Waveform Components splits the waveform data into the array of points (Y), the delta between the points (dt) and the starting time (t0).

32. The DAQmx Stop VI and DAQmx Clear VI ensure that your program properly closes communication and frees up the hardware before stopping the code. Again, to learn more about these VIs, hover over them with the Context Help visible.

33. Wire the task and error wires through the code. This will pass the task values and any errors that may arise through the code.

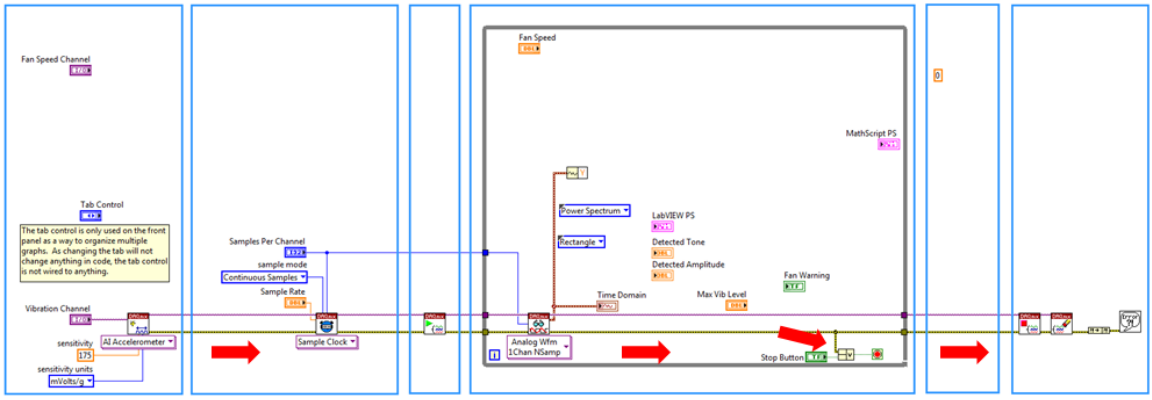


Figure 18. The task and errors within your code will flow through the wires to the simple user dialog at the end of your code.

Because LabVIEW uses dataflow to enforce order of execution, you can visualize the data traveling across each of the wires, from one VI to the next. Each VI will only execute once all the wired inputs have been accounted for.

Wiring the error wire thought the code ensures that any errors in the code will pass through the code and be reported to the user.

34. Wire the error wire from the DAQmx Read VI to the input of the Compound Arithmetic VI. This VI will OR the inputs to stopping the While Loop from executing. The code you just wrote will stop if an error occurs or a user stops presses the stop button.

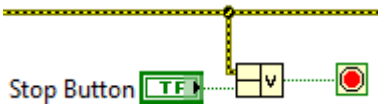


Figure 19. The Compound Arithmetic ORs the inputs, stopping the While Loop if the code has an error or the user presses stop.

35. Once you wire the error wire to the Compound Arithmetic, your code should be ready to run. Check to see that the Run Arrow is intact ( ).

36. Press **Ctrl+E** to toggle to the Front Panel.

37. Press the drop-down arrow on the Vibration Channel to select the correct channel. In this exercise, the accelerometer is attached to channel 0 or channel 1 of the NI 9234. If you renamed your modules, select **Sound_Vibration/ai1**.

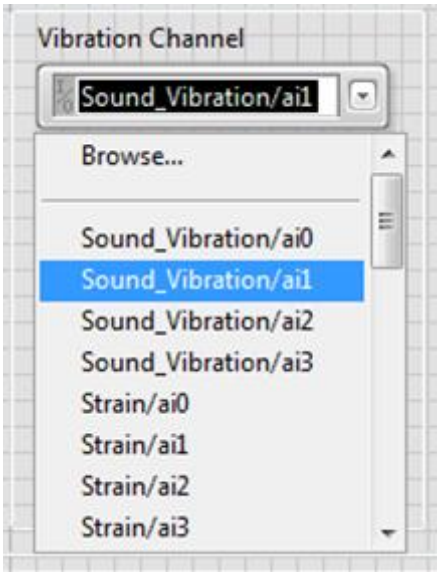


Figure 20. Select the channel to which your accelerometer is attached.

38. Set the sample rate to 10000 Hz and the samples per channel to 1000, like in Figure 21.

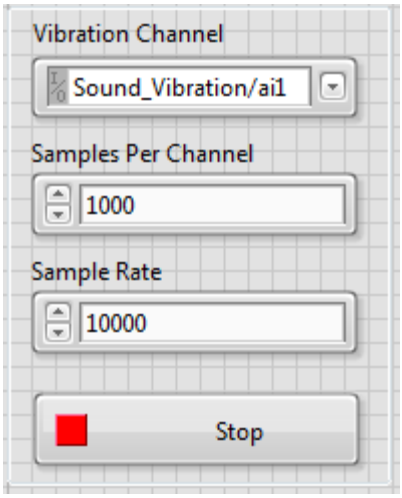


Figure 21. A good rule of thumb is to set the number of samples to 1/10th of the sample rate.

The sample rate dictates how fast your DAQ device samples the selected channel. The Samples Per Channel dictates how much data is pulled from the DAQ buffer each time the loop runs. In the example above, DAQmx Read will wait until there are 1000 samples on the buffer before it pulls them from the DAQ device. This translates to an update every 1/10th of a second. If you change the Samples Per Channel to 10000, your code will update once every second. If you get buffer overflow errors, it usually means that your loop is not executing fast enough to pull data off the DAQ device before the buffer is full. Try increasing the number of Samples Per Channel if this is the case. In some cases, you may need to split your code into parallel loops. For more information about this type of architecture, search *Producer Consumer* on ni.com.

39. Press the run button. You are now acquiring vibration data from the accelerometer in the Sound and Vibration Signal Simulator. Make sure that the switch on the Sound and Vibration Signal Simulator is set to DIAL and try increasing and decreasing the speed of the fan. Notice the changing signal on the screen.

Often, time-domain signals of sound or vibration signals cannot tell you what is happening to the device being monitored. Perhaps the most common processing on time domain sound or vibration signals is an FFT. The next portion of the exercise will step you though adding signal processing, using both LabVIEW VIs and the MathScript node.

40. Press **Stop**. Save this VI, as you will use it in the next part of the exercise.

Part B Controlling the Fan Speed with Analog Output

Estimated time: 25 minutes

Many times, data acquisition code needs to do more than simply measure a signal. Adding control to the code enables an operator to use the software to control something. This could be a digital output, like an alarm or PWM speed signal, or an analog output, like a stimulus waveform. In this example, you will use a DC voltage signal to control the speed of the fan that was controlled with an external knob in the first part of the exercise.

Before you start this exercise, switch the physical Fan Speed Control switch to BNC.

1. To begin this exercise, open the Block Diagram of Vibration main_exercise.vi
2. For this exercise, a large part of the coding will come from the DAQmx palette. To access this palette right click on any white space on the Block Diagram and navigate to **Measurement I/O » NI-DAQmx**, like in Figure 22.

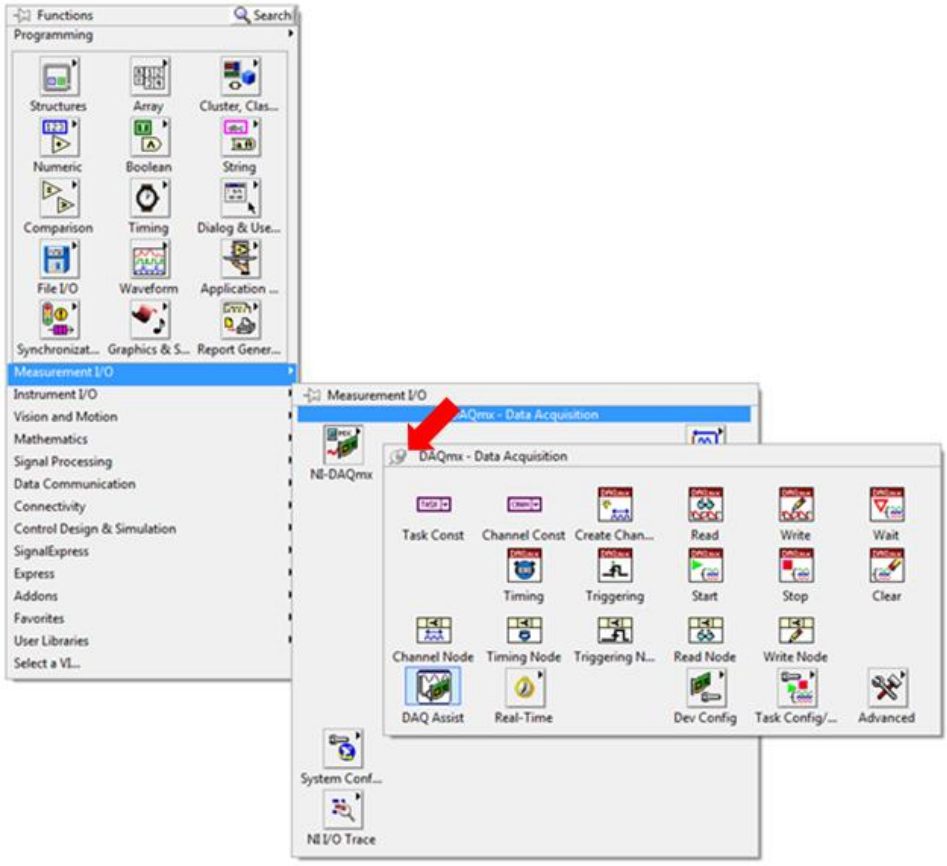


Figure 22. The DAQmx palette includes all the required VIs for communicating with your NI CompactDAQ chassis.

3. Click on the pin to keep this palette visible (red arrow in Figure 22).

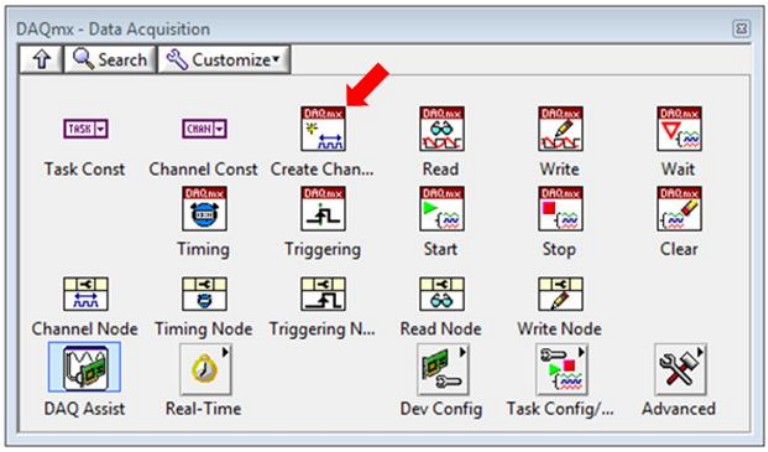


Figure 23. Pinning a palette enables you to click elsewhere in your code with the palette still visible.

4. To start, drag the **NI-DAQmx Create Channel** (red arrow in Figure 23) and place it in the position shown in Figure 24.

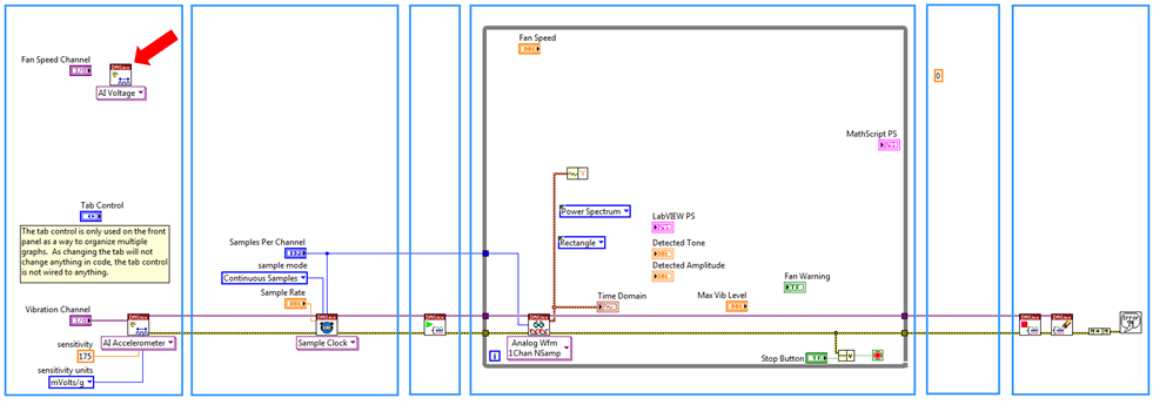


Figure 24. The same VI for creating and acquisition channel is used to create a generation channel, but you can change the polymorphic instance by using the drop down menu.

5. To configure this VI for generation, click on the menu below the VI and select **Analog Output » Voltage**.

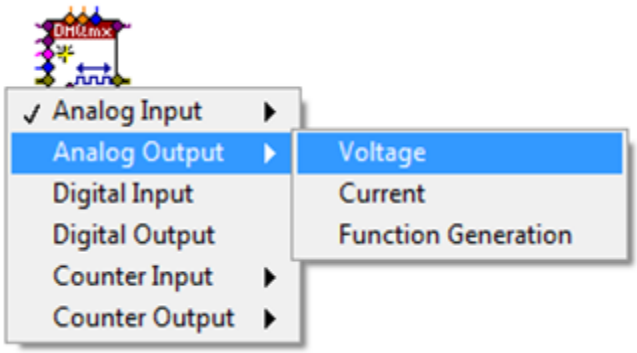


Figure 25. The polymorphic DAQmx Create Channel VI can be configured to create an analog output value.

6. Wire the VI as seen in Figure 26.

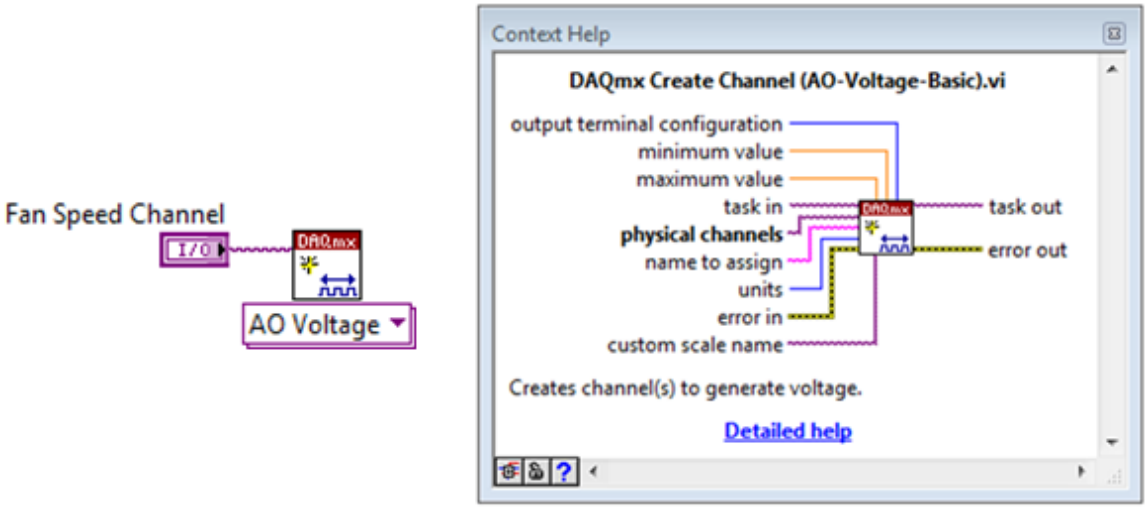


Figure 26. In this instance, the only thing you need to wire is the channel input to the physical channels input.

7. Like in the previous exercise, drag the following VIs onto the block diagram and place them like you see in Figure 25.

    a) DAQmx Create Channel

    b) DAQmx Start Task

    c) DAQmx Write

    d) DAQmx Write

    e) DAQmx Stop

    f) DAQmx Clear

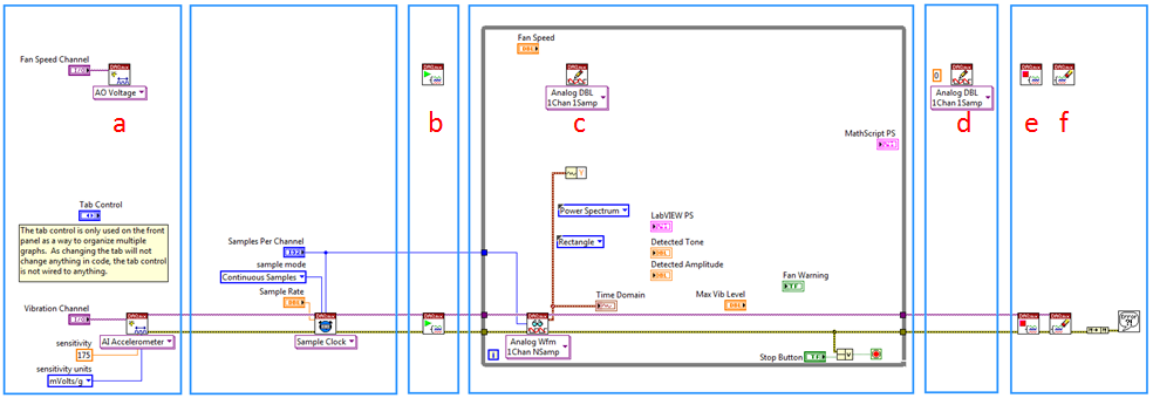


Figure 27. Again, the DAQmx flow, even for output, follows nearly the same pattern.

You may have noticed that you placed two DAQmx Write VIs. This is because the last value on the output buffer will persist even after the task is stopped and cleared. In this example, you may end up with a running fan after stopping the code, depending on the last value. To make sure the fan is always stopped when the code is stopped, simply write a value of zero to the buffer before stopping and clearing the task.

For this exercise, we will be writing a single sample to the analog output channel of the NI 9263 each time the loop runs, so the default instance of DAQmx Write VI will work for your program.

8. Wire the Fan Speed control into the data input of the first DAQmx Write VI.

9. Wire the 0 constant next to the second DAQmx Write VI as seen in Figure 28.

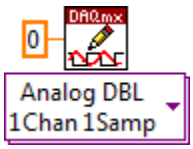


Figure 28. The second DAQmx Write writes a zero to the buffer to make sure the fan turns off when the VI is stopped.

10. Pull the Merge Errors VI up to reveal another input. The Merge Errors VI is the second VI from the end, between the Clear Task and Error Handler.

11. Wire the task and error wire through the code, as seen in Figure 29.

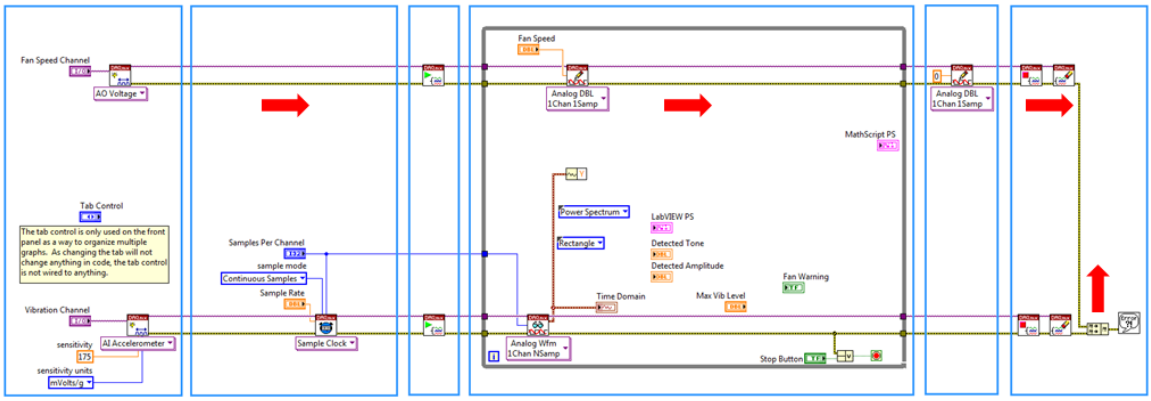


Figure 29. The task and error wire communicate information between each of the VIs in the code.

12. Switch to the Front Panel.

13. Set the *Fan Speed Channel* to **Voltage_out/ao1**. This is the channel of the voltage output module connected to the fan control on the Sound and Vibration Signal Simulator.

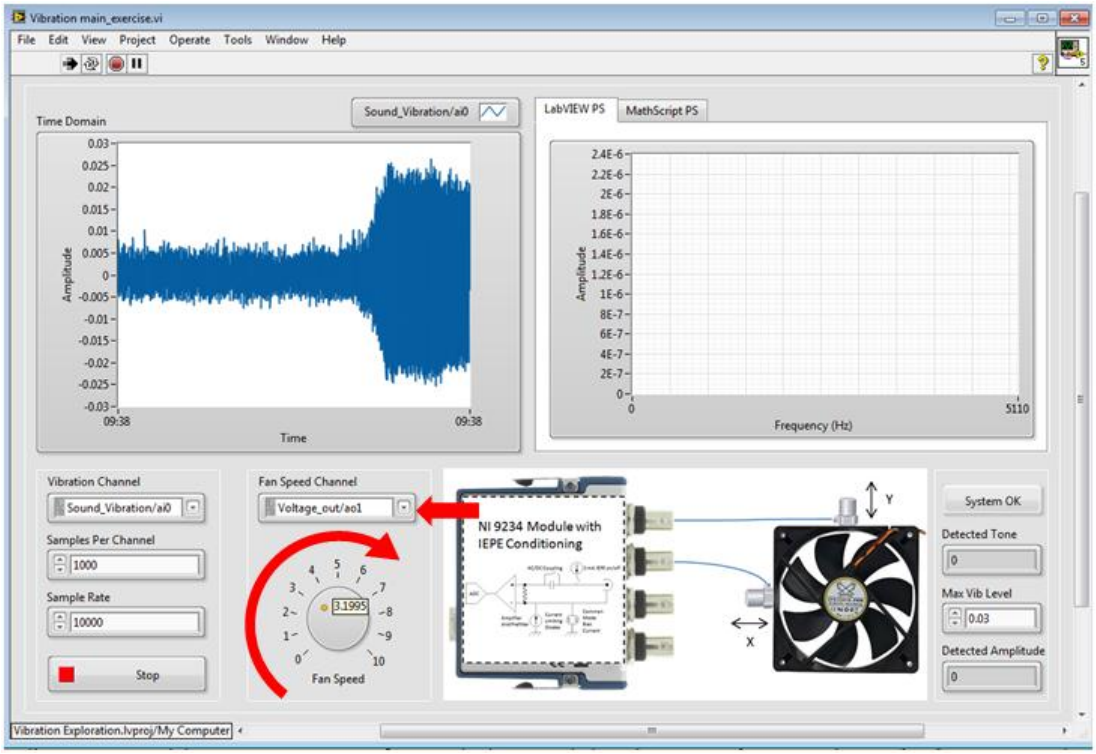


Figure 30. Using an analog output, you can control the speed of the fan in the Sounds and Vibration Signal Simulator.

14. Press the Run button.

15. Adjust the speed of the fan using the Fan Speed control knob. You should notice that the vibration signal in the time domain increases and decreases with the speed of the fan.

16. Stop the VI and save your work. You will use it in the next part of this exercise.

Part C Adding Analysis to Your Acquisition Code

Estimated time: 25 minutes

In real applications, the measured voltage signals are complex waveforms that contain multiple frequency components. Sound and vibration analysis usually involves identifying and examining these frequency components. To do so, you must convert the signals from the time domain to the frequency domain mathematically using Laplace, Z-, or Fourier transforms. Fourier analysis is the most common for this application because it obtains the magnitude and associated phase for each frequency component in a signal.

For the next part of the exercise, you will calculate the power spectrum using two methods. The first method will use built-in LabVIEW VIs and the second will integrate .m script to calculate the same power spectrum. Using this spectrum, you will calculate the frequency and amplitude of the vibration to determine the balance of a fan.

1. To begin the exercise, open the block diagram of Vibration main_exercise.vi.

2. Right click on the block diagram and select **Signal Processing » Waveform Measurements » FFT Power Spectrum and PSD** as seen in Figure 31.

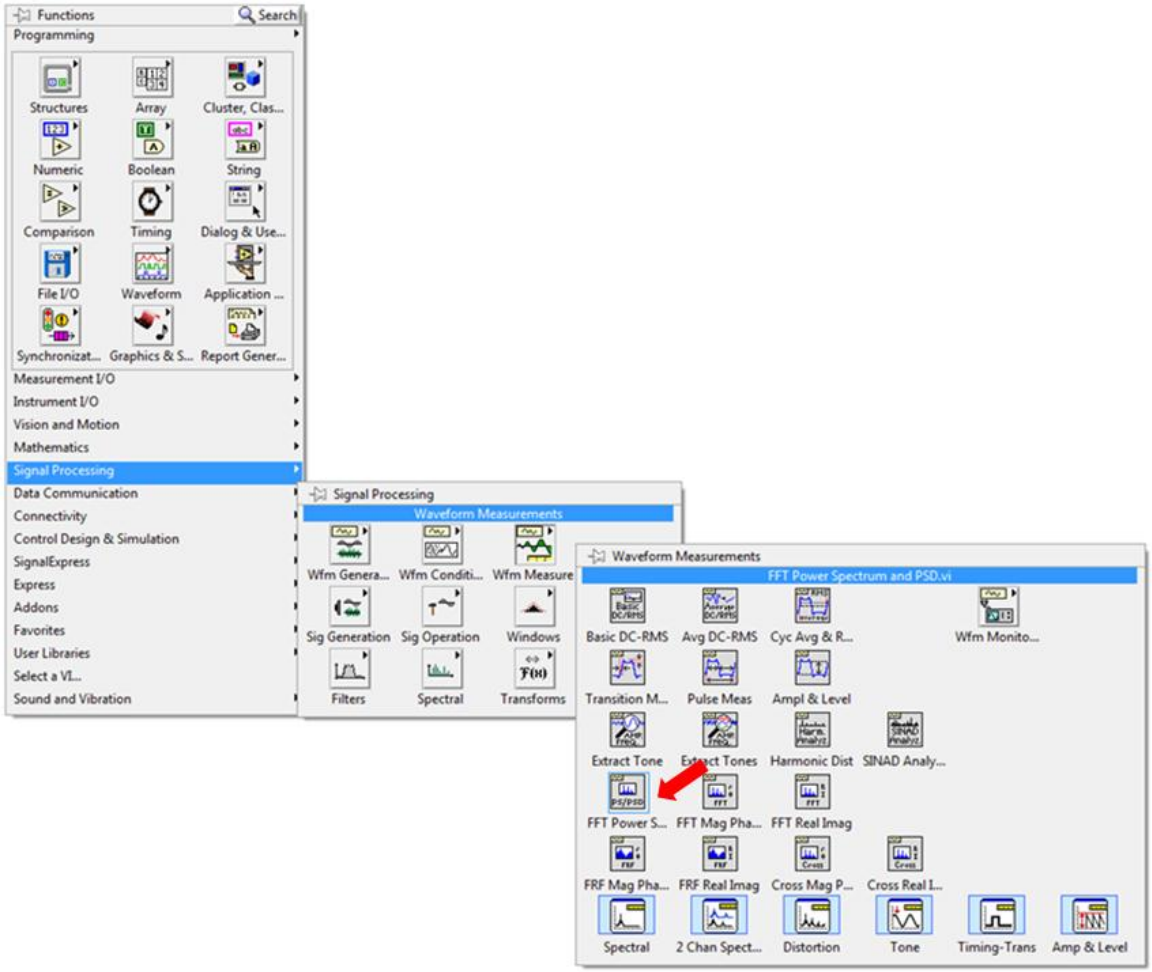


Figure 31. LabVIEW contains many built-in analysis functions and many others are supported through tool kits and add-ons.

3. Place the FFT Power Spectrum VI in the position shown in Figure 32

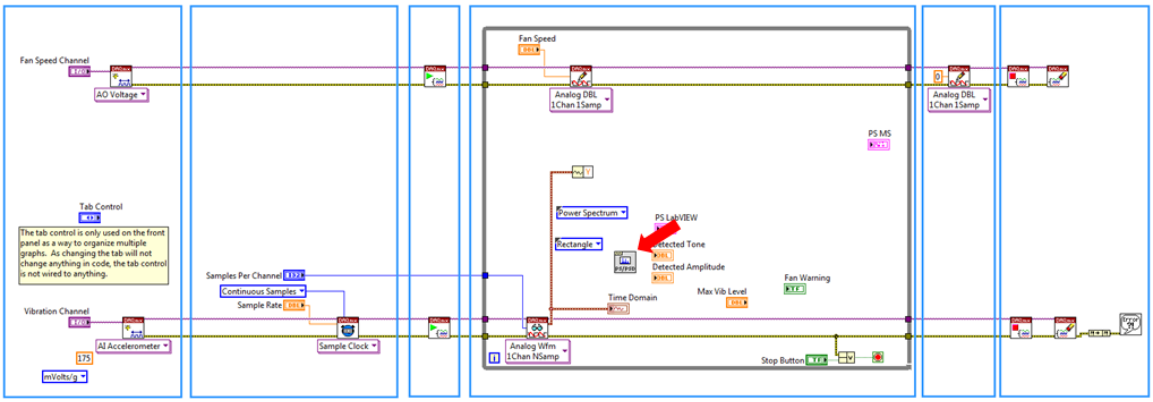


Figure 32. The FFT Power spectrum will calculate the frequency spectrum during the acquisition.

4. Wire the inputs to the VI as seen in Figure 33.

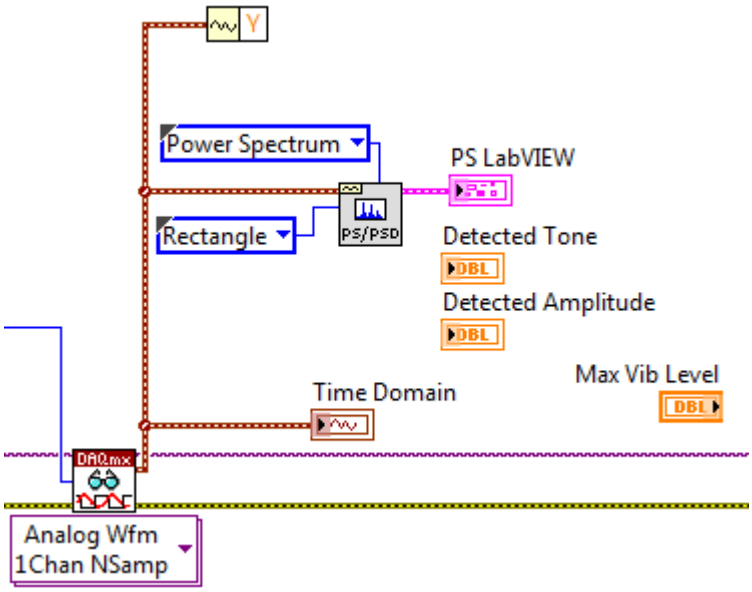


Figure 33. The power spectrum will be calculated each time the loop in the code runs.

5. To learn more about this VI, press Ctrl+H to bring up the Context Help and hover over the FFT Power Spectrum. If you want to learn even more about this VI, press the Detailed Help link.

6. With the FFT Power spectrum wired up, open the front panel and run the VI. You should now be able to see the frequency information from the on-board accelerometer.

7. Try increasing and decreasing the fan speed. Notice that both the time domain and frequency domain signal changes. Try changing the switch on the Sound and Vibration Signal Simulator from the balanced fan to the unbalanced fan. Notice how the signal changes. Measurements like these are commonly used for machine health diagnosis.

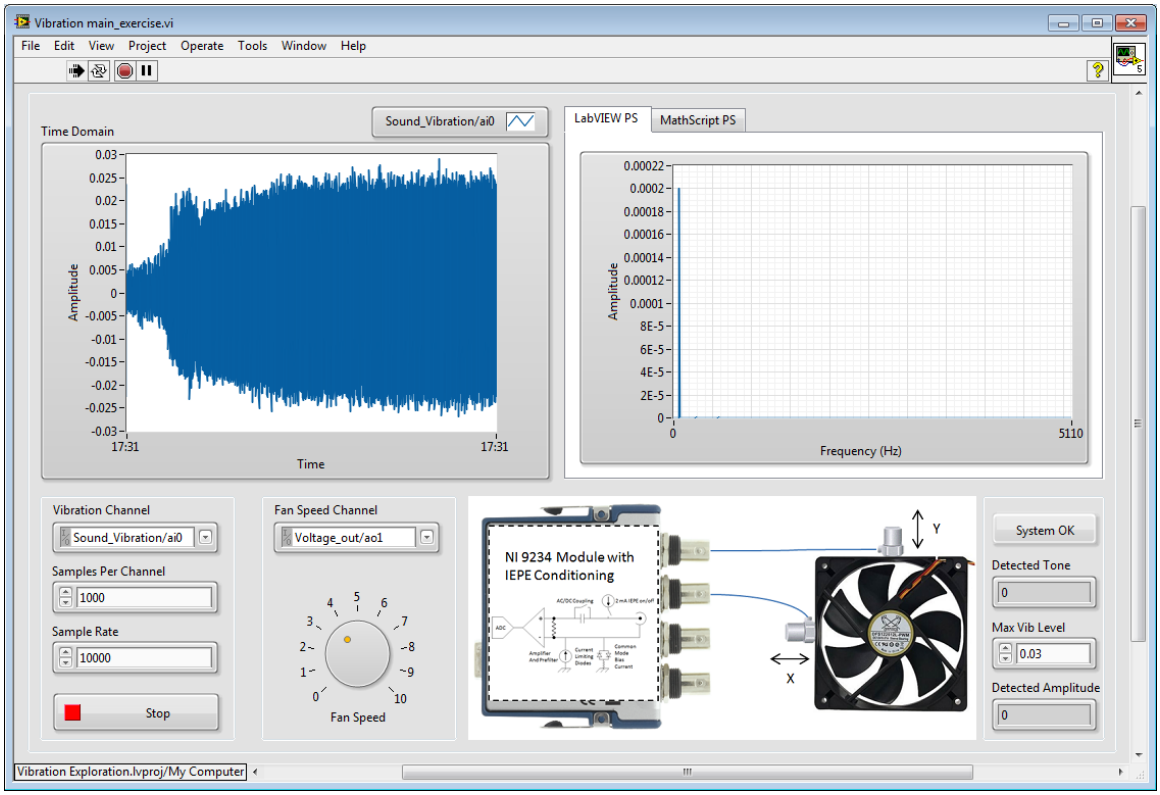


Figure 34. From the frequency domain signal, you can understand more about what is happening in your system.

While LabVIEW includes extensive analysis functions, sometimes the situation may require integration with other languages, like .m script. For the next portion of the exercise, you will perform the same analysis using the MathScript node, which enables you to integrate your existing .m script files into your LabVIEW program.

8. Stop your VI and open the Block Diagram.

9. Right click on the Block Diagram and select **Mathmatics » Scripts and Formulas » MathScript Node**.

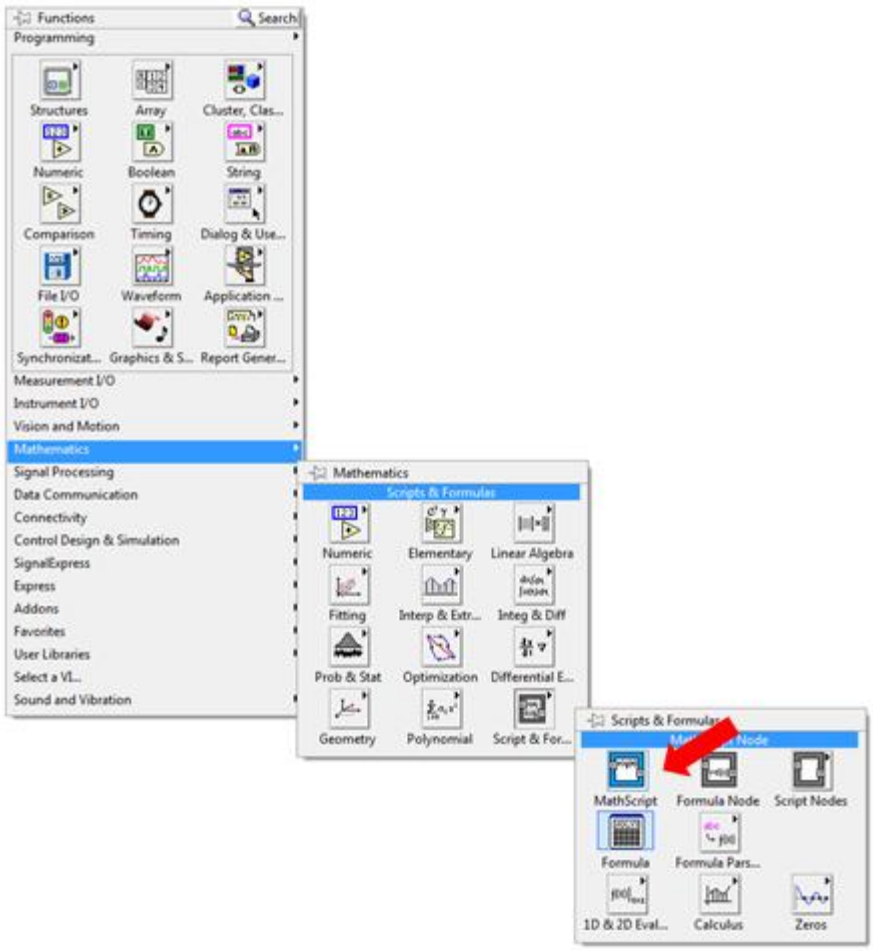


Figure 35. The MathScript Node integrates graphical LabVIEW code with text based .m script.

10. Drag a small box in the position shown in Figure 36.

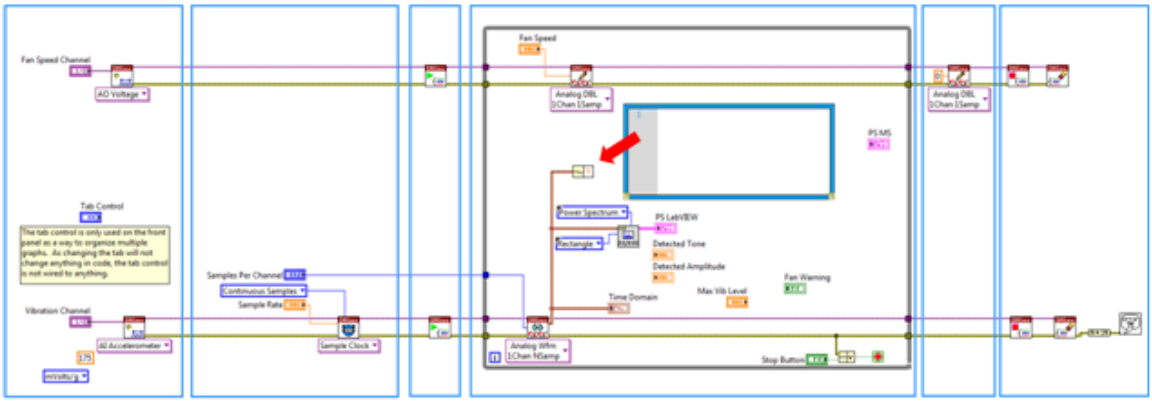


Figure 36. When you place the MathScript Node, give yourself enough room to type a few lines of text.

11. For this exercise, you will need to use the components from the time-domain waveform. To do this, you will use the Get Waveform Attributes, which is already placed for you on

the block diagram (red arrow in Figure 36). When you hover over this function, you should see handles appear. Drag the handle down.

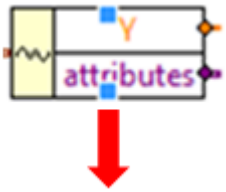


Figure 37. Drag the Get Waveform Attributes down to add the dt attribute.

12. Click on the attributes portion and select dt. You are now separating out the Y values of the time domain waveform and the change in time between each of the sample points.

13. Before you can reference data from your LabVIEW code inside the MathScript Node, you need to define the values that will be input and output from the node. To do this, right click on the left edge of the MathScript Node and select Add Input. Call this input dt.

14. Right click on the left side again, select Add Input and call this one Y.

15. Right click again on the edge and select Add Input. Call this one dt.

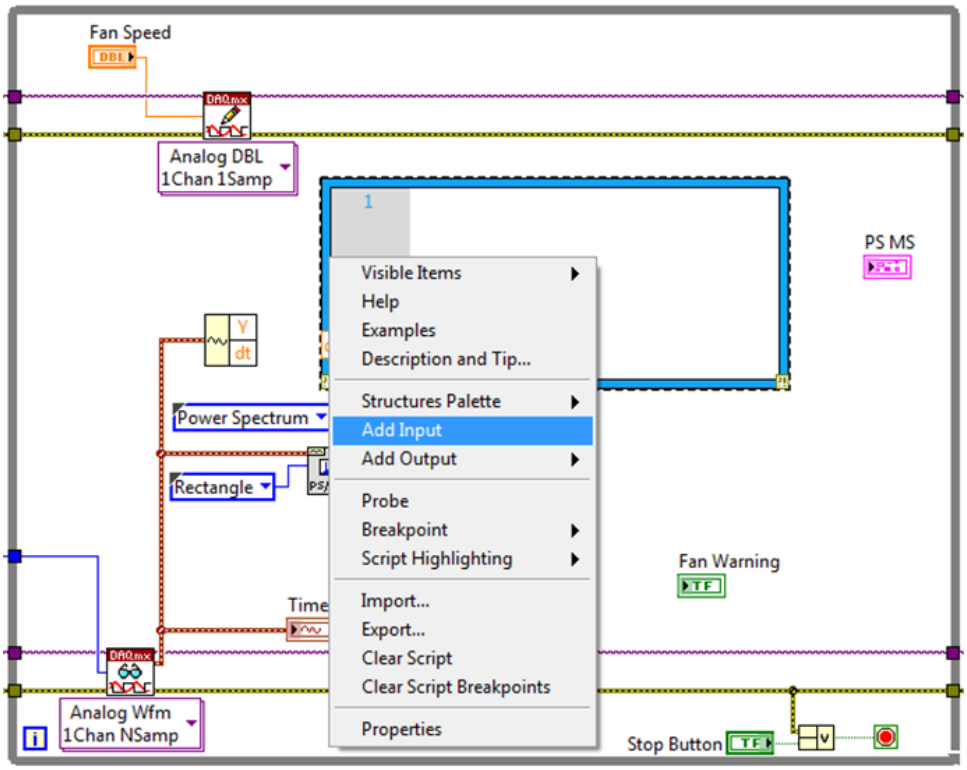


Figure 38. Add the inputs to the MathScript Node by right-clicking on the edge.

16. Open **m script text.txt** from the Support VIs directory in the LabVIEW Project Explorer.

17. When the file opens, press **Ctrl+A** to select all the text then press **Ctrl+C** to copy.

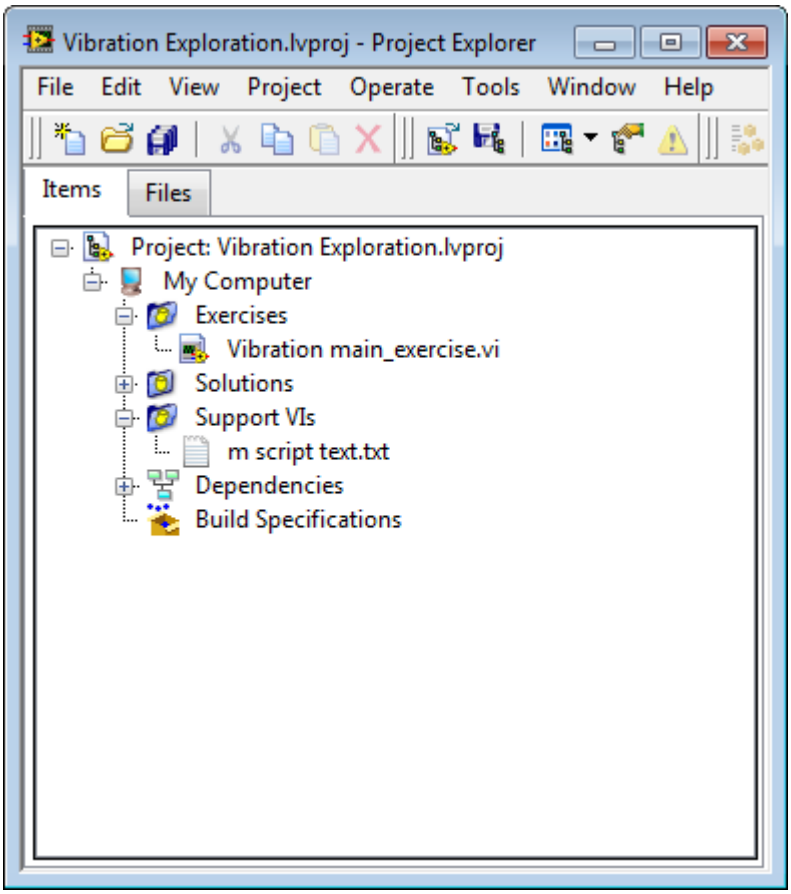


Figure 39. m script text.txt includes the required steps to perform frequency analysis using MathScript.

18. Navigate to the Block Diagram.

19. Left-click inside the MathScript Node then press Ctrl+v to past the text. Your MathScript Node should look like Figure 40.

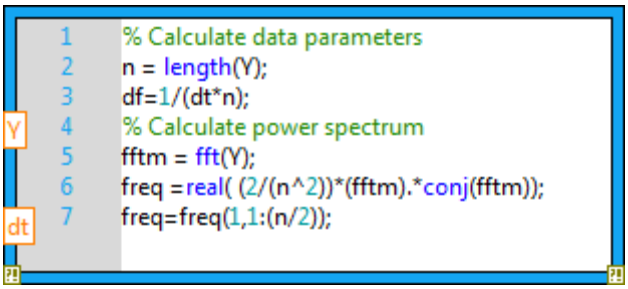


Figure 40. You can past any .m script functions into the MathScript Node to perform text-based math in-line with your graphical code.

20. Once you have the text entered, you can add the outputs to the MathScript Node. To

do this, right-click on the right edge of the MathScript Node and select **Add Output»df**.

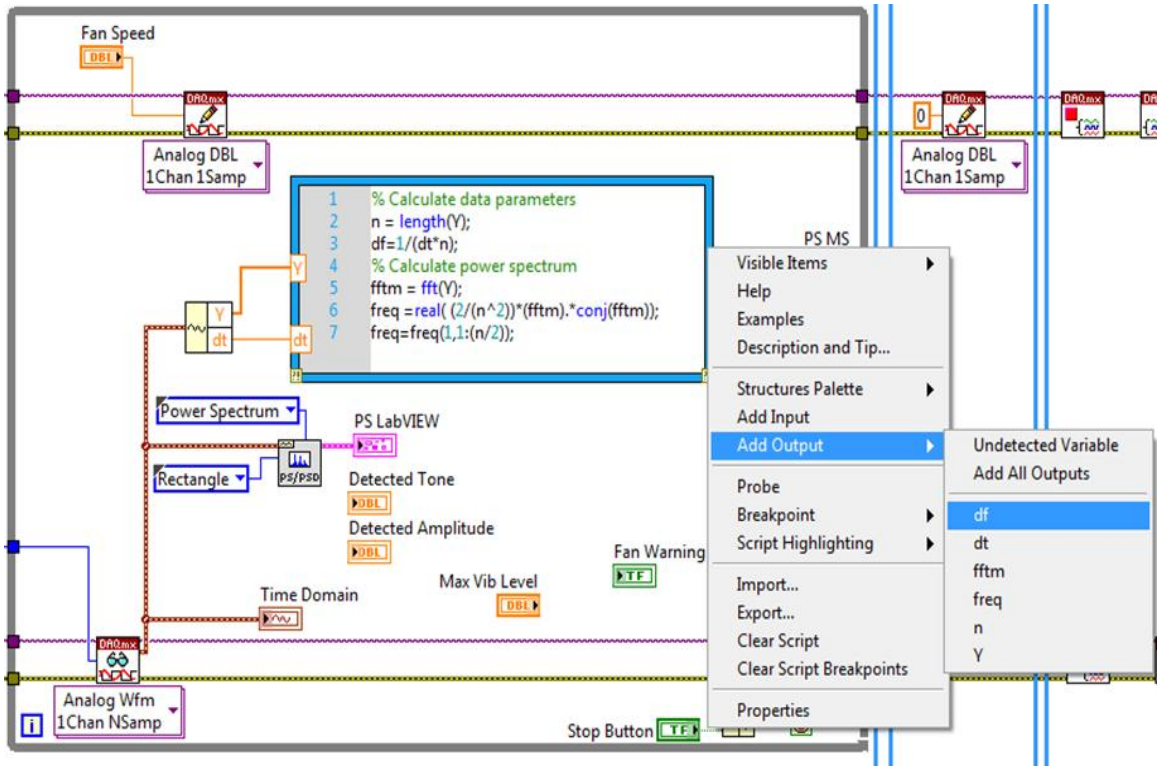


Figure 41. Right click on the edge of the MathScript Node to add outputs.

21. Right click on the right edge again and select **Add Output»freq**.

By inputting the change in frequency (df) and the power at that frequency (freq), you can build a graph to visually represent the power spectrum while the data is being acquired.

22. Wire the Outputs of the Get Waveform Components VI into the respective MathScript Node inputs (left side of the MathScript Node), as shown in Figure 41.

23. To build the graph, you will need to bundle the values into a cluster (a cluster is a way to group elements of differing data types, which is out of the scope of this lesson, but is thoroughly covered in the on-demand training). Right click on the Block Diagram and select **Programming»Cluster, Class & Variant»Bundle**, like in Figure 42.

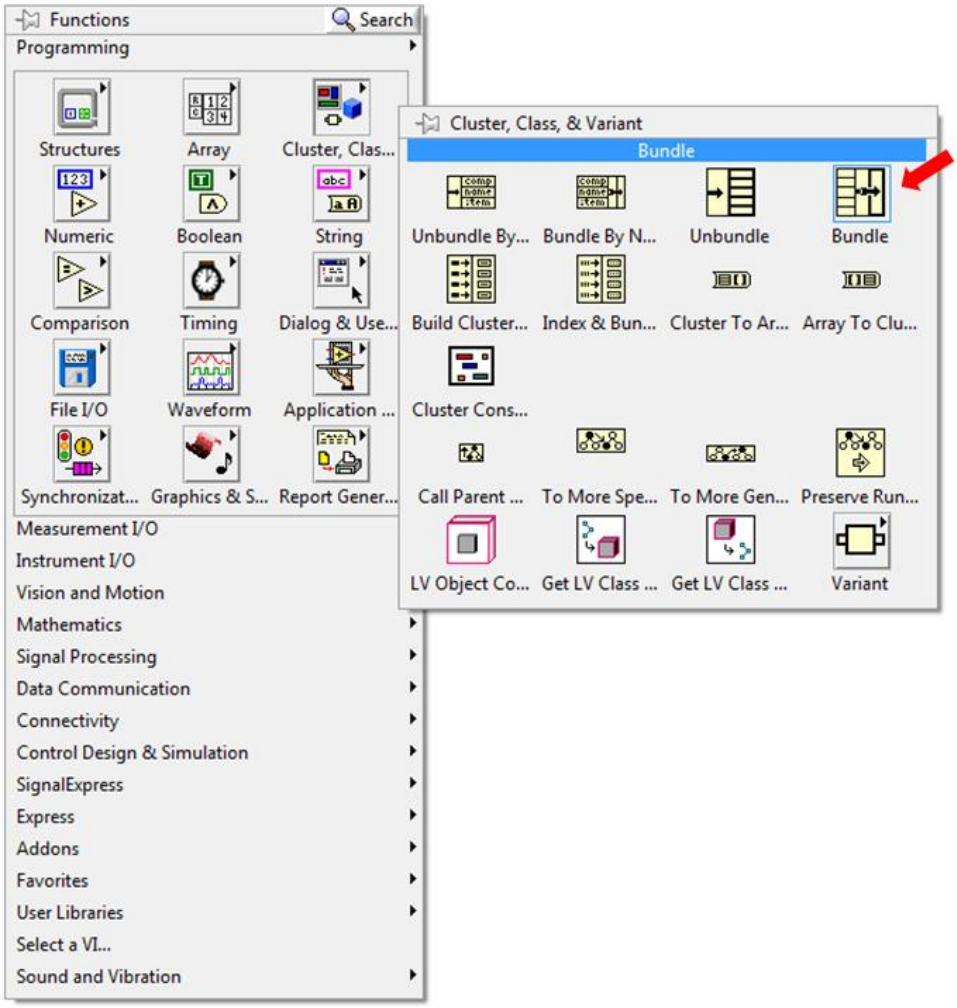


Figure 42. Use the Bundle function to combine the elements you need to create the graph.

24. Place this function in the location shown in Figure 43.

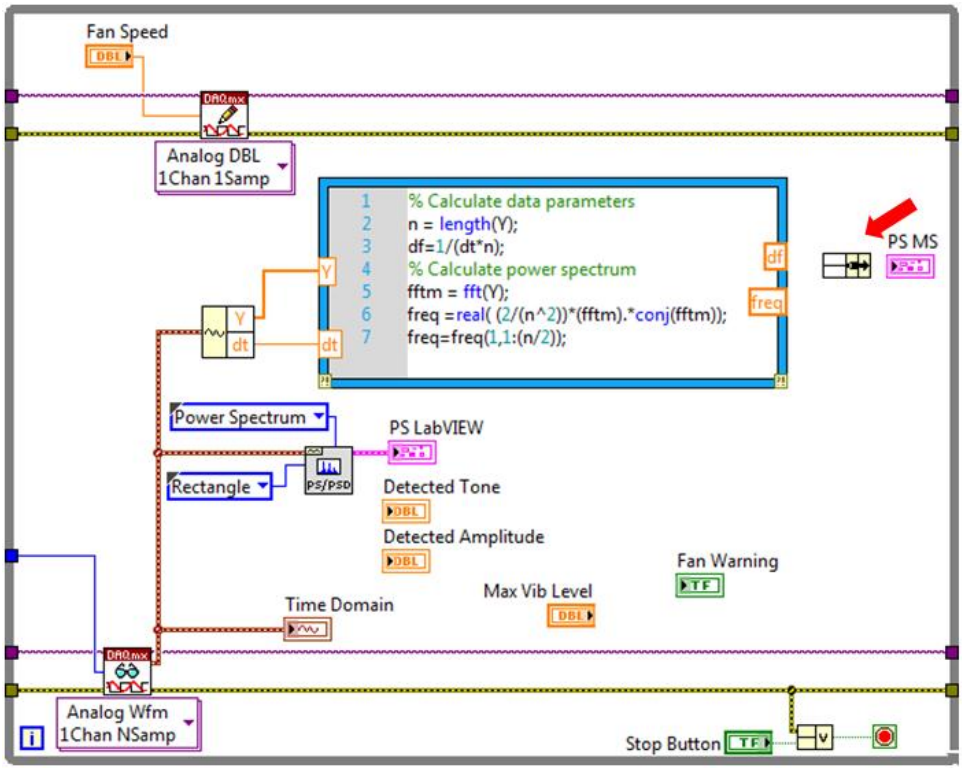


Figure 43. Place the Bundle Function in between the MathScript Node and the PS MS graph input.

25. To add another input, hover over the Bundle Function, then click and drag the handle that appears.

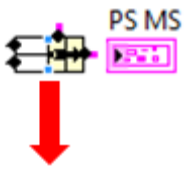


Figure 44. Drag the input down to add another input.

26. Add a zero constant to the block diagram by right clicking on the block diagram, selecting **Programming»Numeric»DBL Numeric Constant** and place it in the same position as figure 46.

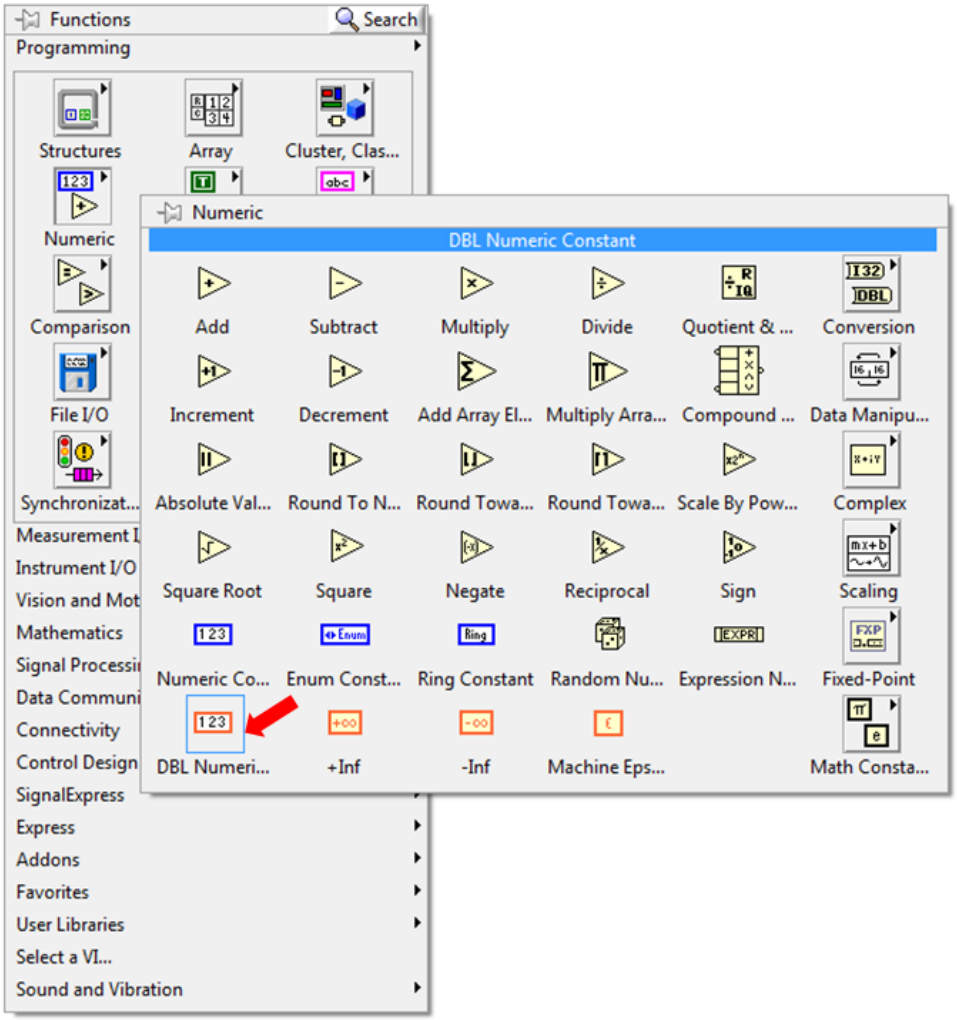


Figure 45. The DBL Numeric Constant is simply a numeric constant that you can use on the block diagram without requiring any input from the front panel.

27. Wire the Bundle Function as seen in Figure 46.

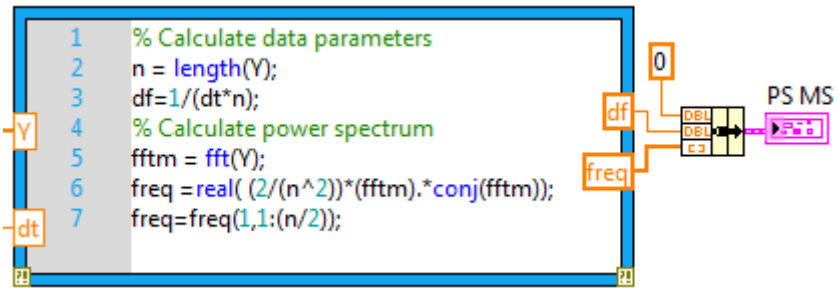


Figure 46. The Bundle Function combines functions that are different data types.

28. Switch to the Front Panel and run the VI.

29. Try switching between the LabVIEW calculated frequency domain and the MathScript

calculated frequency domain by clicking between the tabs.

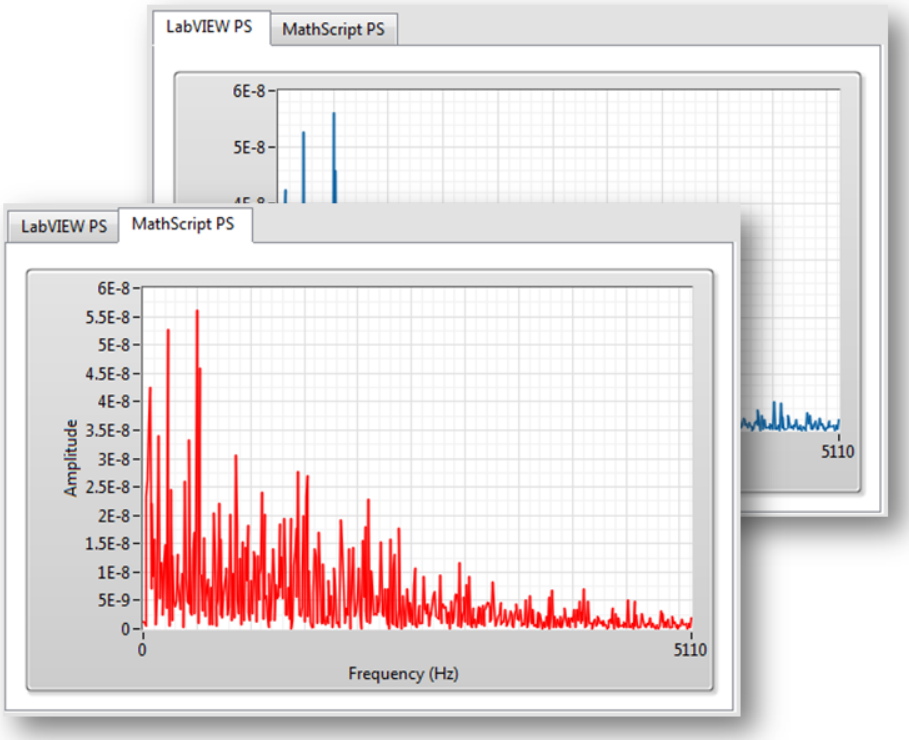


Figure 47. Both the MathScript and LabVIEW code calculate and display the exact same frequency domain.

What you have just seen is the integration of external, text-based code into the graphical LabVIEW environment. With these tools, you can choose the most effective approach to attacking a problem, whether text based or graphical, all in the same LabVIEW environment.

30. Stop the VI and save. Keep this VI open for the next exercise.

Part D Alarming Based on Vibration

Estimated time: 20 minutes

Accelerometer measurements can be used for applications like structural test, noise vibration harshness testing or machine health diagnosis. For the last part of this exercise, you will measure the magnitude of the vibration and alert a user/operator when that threshold is exceeded. For more detailed information on machine health and asset monitoring, check out ***ni.com/mcm****.*

1. To begin this part of the exercise, navigate to the block diagram of Vibration_main_exercise.vi.

2. For this exercise, you will simply be extracting the fundamental frequency and testing against a limit determined by the user. To extract this tone, right-click on the block diagram and navigate to **Signal Processing»Waveform Measurements»Extract Tone**

as shown in Figure 48.

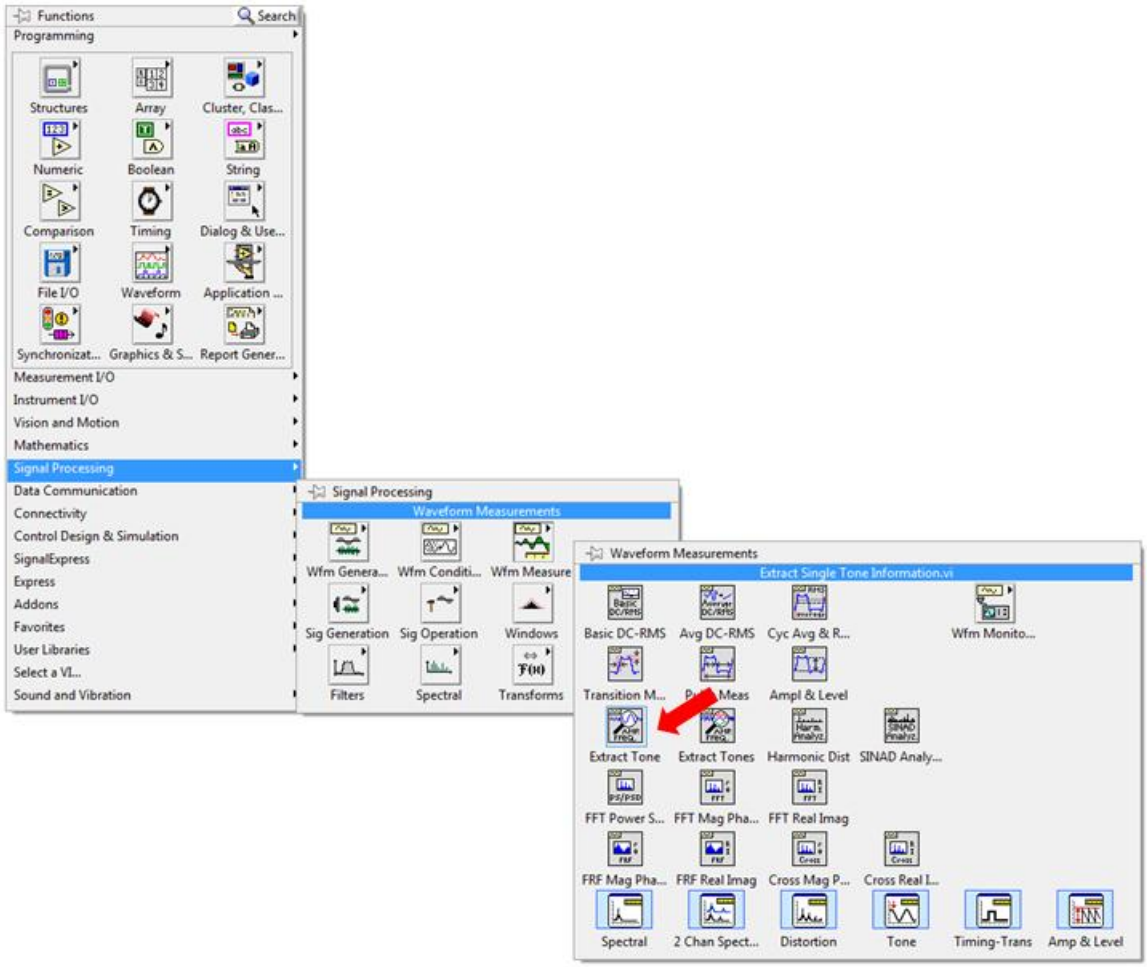


Figure 48. Select Extract Tone to measure the amplitude of the highest powered tone in the frequency domain.

3. Place the VI in the location shown in Figure 49 and wire the inputs and outputs as shown. If you need help figuring out the different inputs, remember that Ctrl+h will show the Context Help to provide you with a reminder.

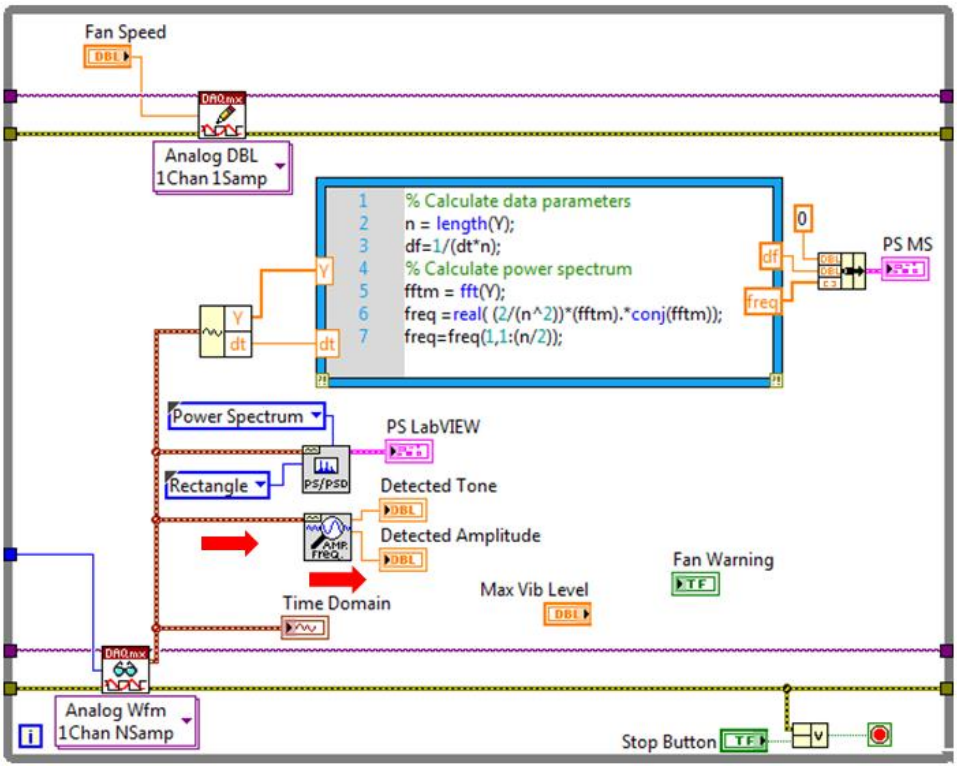


Figure 49. Wire the time domain waveform to the Extract Tone VI to pull the frequency and amplitude of the highest power tone.

4. Switch to the front panel.
5. If you run the VI, you should see the detected tone and amplitude on the front panel indicators.
6. Stop the VI.

With the tone and frequency information being calculated, all that is left is to add in a comparison and alarming.

7. Place a **Greater?** function from the Comparison Palette (Figure 50) in the location show in Figure 51.

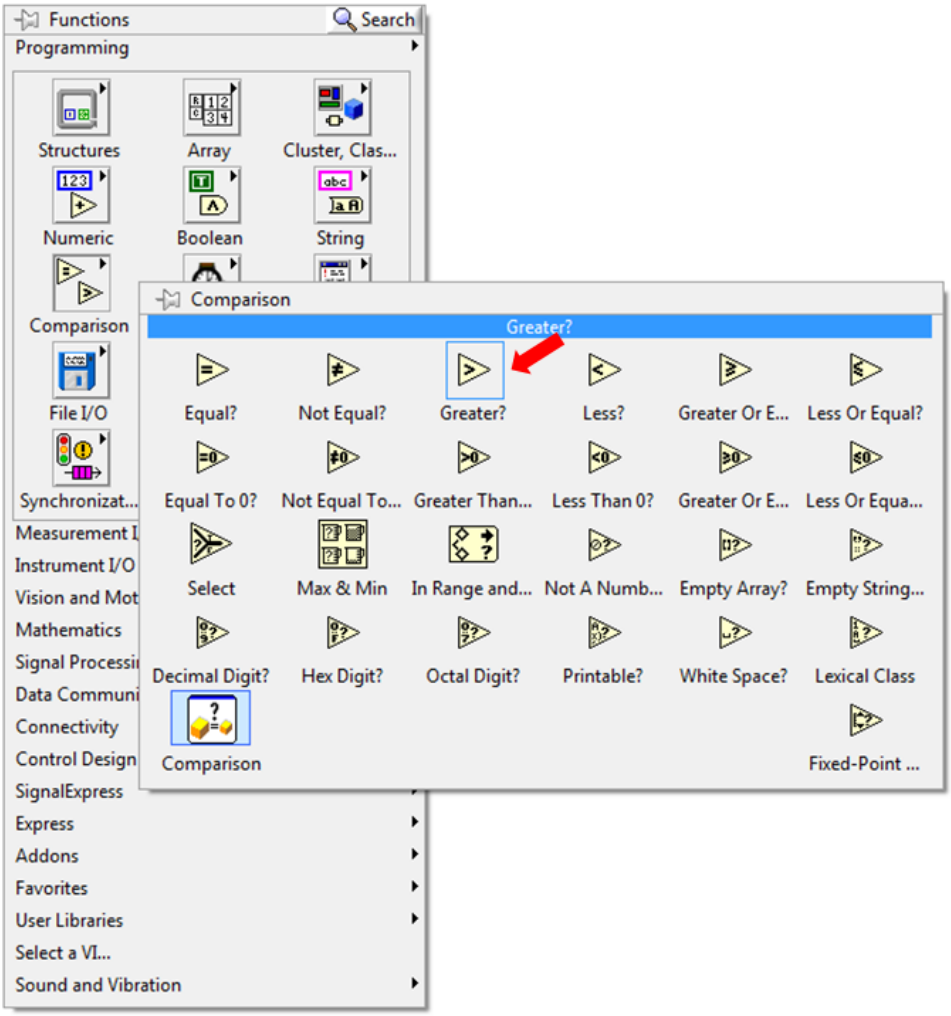


Figure 50. The Greater? **function will compare two numbers and output a Boolean True if the top number is greater than the**

8. Wire the **Greater?** function as seen in Figure 51.

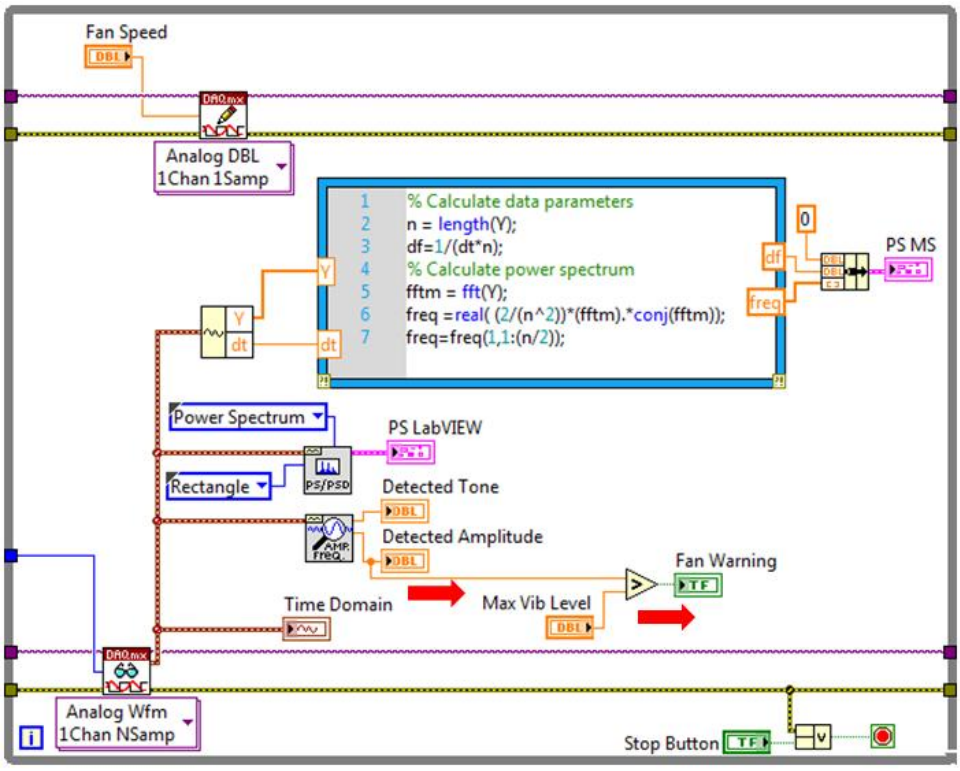


Figure 51. The Greater? **comparison compares the amplitude of the largest power tone with a user specified value.**

9. Open the Front Panel and run the VI. Make sure that the switch on the Sound and Vibration Signal Simulator is set to Unbalanced Fan.
10. Change the speed of the fan by adjusting the knob on the front panel. Notice that at higher speeds, the Fan Warning will show that there is an issue with the fan.
11. If you cannot get the fan alarm to light, try reducing the Max Vib Level input (shown in Figure 52).

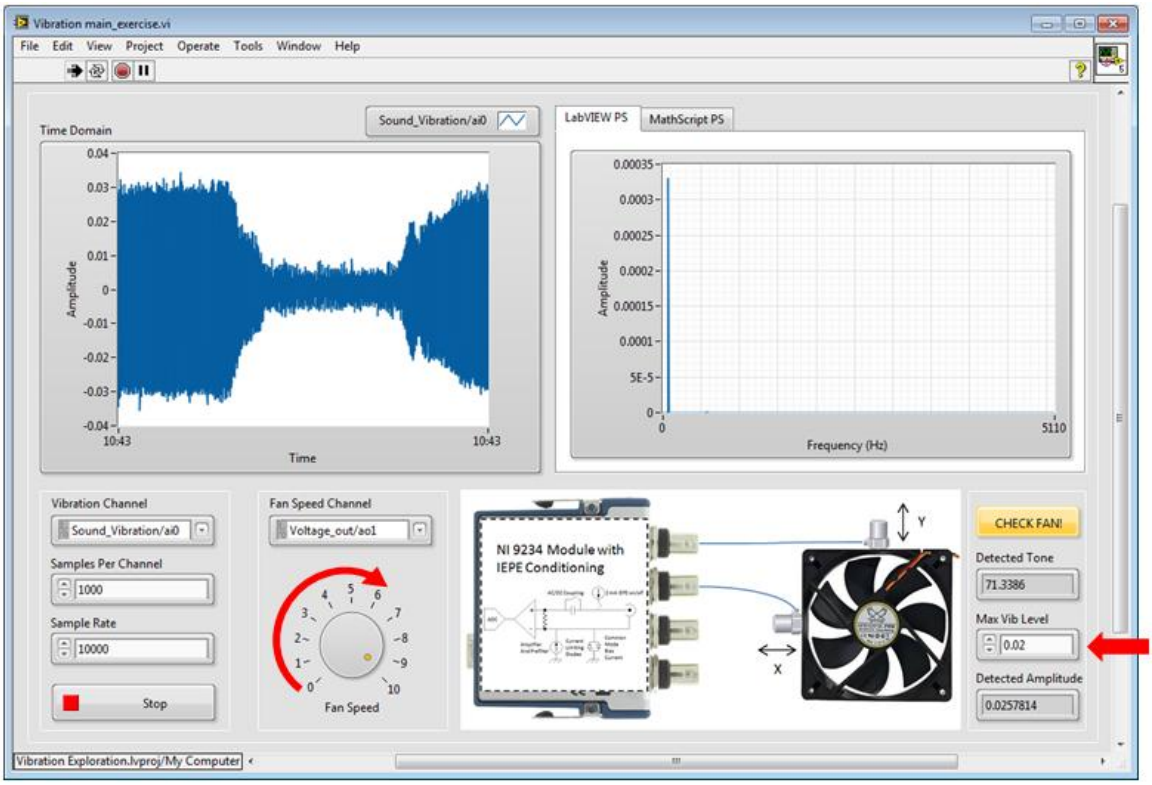


Figure 52. This VI should behave just like the solution you opened at the beginning of the exercise.

12. Stop the VI and save.

<End of Exercise>