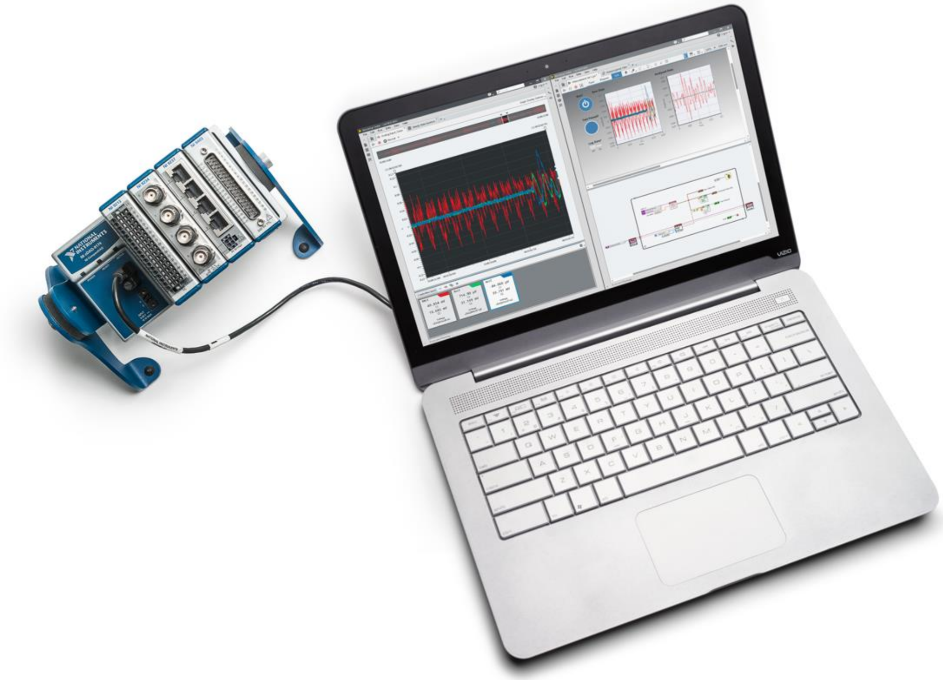
Ph.D. School ITALO GORINI
ON LINE - September 04-09, 2020

Introduction to LabVIEW and Computer-Based Measurements



Contents

MANDATORY:

Chapter 1 Hardware Configuration and Taking First Measurement 5

Chapter 2 Using Examples to Automate Measurements with LabVIEW NXG 17

OPTIONAL:

Chapter 3 Temperature Measurement and Data Logging 26

Chapter 4 Combining Measurements from Light and Temperature 47

Chapter 5 Strain Gage Measurement with User LED Feedback 63

Chapter 6 Audio Input and Output with Analysis 92

Chapter 7 Vibration Measurements and Analog Output Control 122

Chapter 1 Hardware Configuration and Taking First Measurement

Goals

- Connect your NI CompactDAQ system to the PC
- Configure your hardware and verify signal connections using LabVIEW NXG.

  Acquire and record temperature data to perform a limit test to take your first measurement.

Part A Hardware Connection and Setup

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.

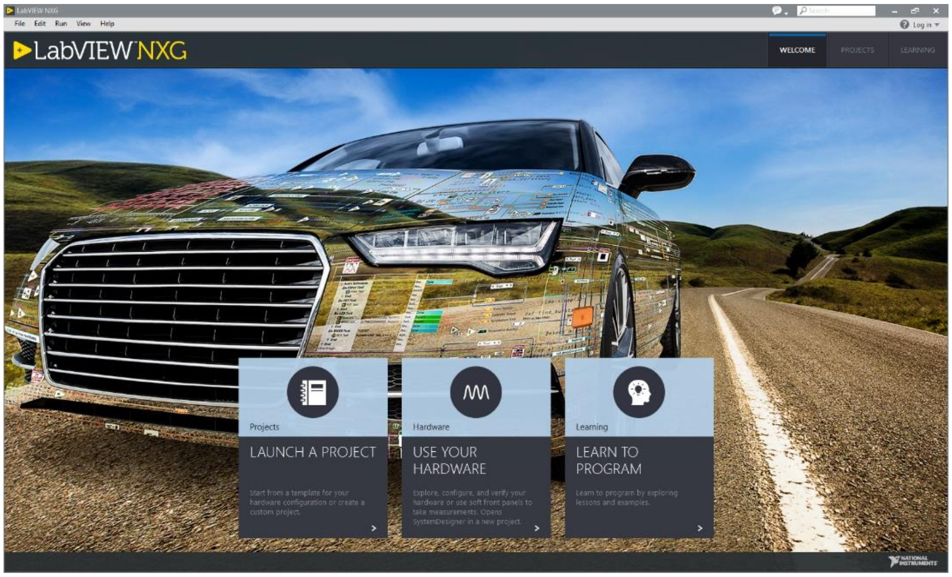
3. Once you have confirmed everything is connected and powered on, proceed to Part B.

Part B Software Configuration

To set-up your measurement system, use LabVIEW NXG to connect to and configure your NI data acquisition hardware. In this exercise, you will discover your NI CompactDAQ chassis and configure your measurement channels in LabVIEW NXG to start acquiring data.

Note: Only NI data acquisition devices and third-party instruments are supported and discoverable in LabVIEW NXG. To connect to other hardware, such as CompactRIO controllers, real-time PXI systems, and RF hardware, use LabVIEW 201x and Measurement & Automation Explorer.

1. Once your hardware is connected and powered on, open LabVIEW NXG by navigating to **Windows » LabVIEW NXG.**



Start from a template for your hardware configuration or create a custom project.

Explore, configure, and verify your hardware or use soft front panels to take measurements. Opens SystemDesigner in a new project.

Learn to program by exploring lessons and examples.

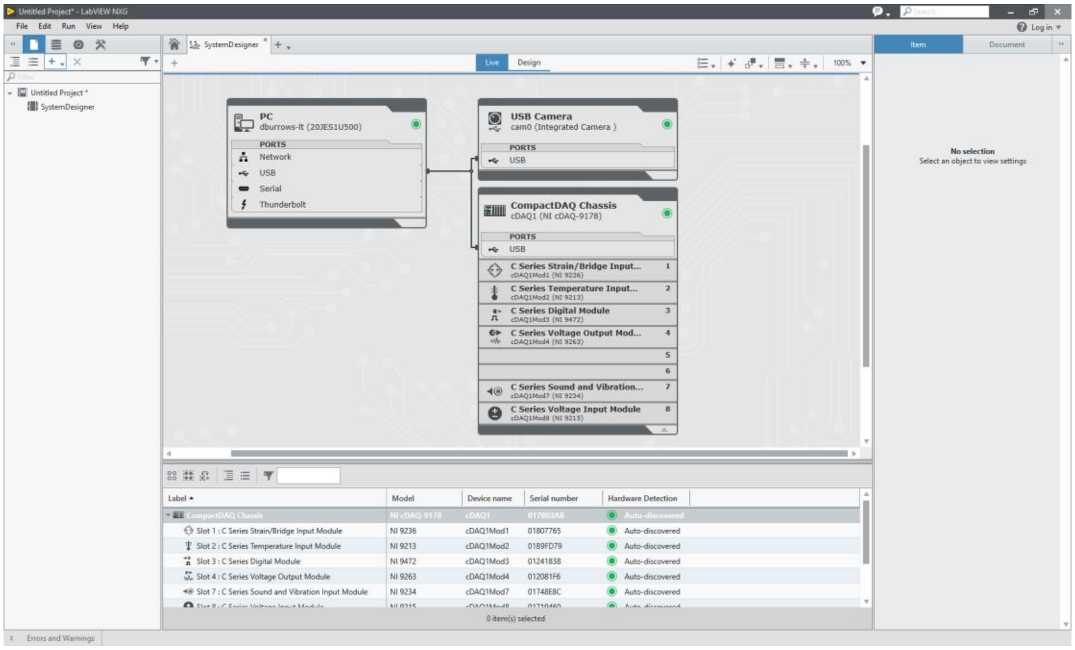
Figure 1-1 The welcome screen appears every time LabVIEW NXG is opened to give you several paths to discover hardware, start a project, or to access learning materials.

The options, shown in Figure 1-1, are:

a. Launch a Project- Create a new LabVIEW project from scratch or continue building an existing program.
b. Explore Your Hardware- Discover and configure hardware connected to your system prior to taking measurements using **SystemDesigner.**
c. Learn to Program- Learn new skills and concepts with tutorials, examples, and real-world applications.

NOTE: As you move forward creating programs and taking measurements, you can always return to this home screen of LabVIEW NXG by selecting the **Home Icon** 🏠 at the top left of the window.

2. Select **Explore Your Hardware.** This will bring up the **SystemDesigner**. In the SystemDesigner view, you can see the hardware available to your system, including your PC, peripherals like integrated webcams, and your CompactDAQ hardware. Note the modules listed under the CompactDAQ Chassis, and verify that they match the modules in your system.

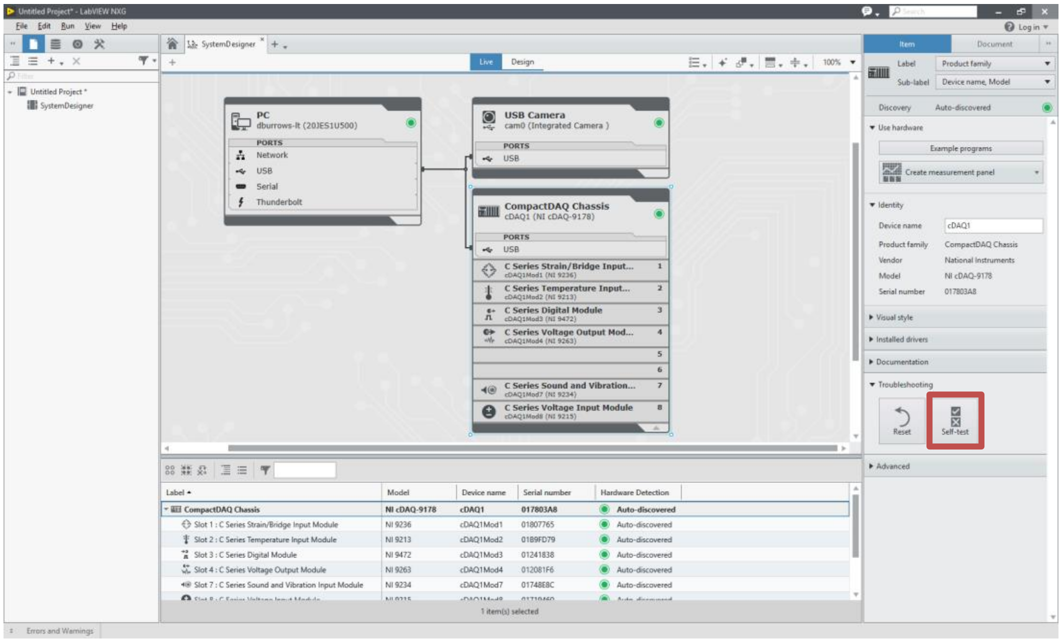


Label	Model	Device name	Serial number	Hardware Detection
CompactDAQ Chassis	NI cDAQ-9178	cDAQ1	017803A8	● Auto-discovered
Slot 1 : C Series Strain/Bridge Input Module	NI 9236	cDAQ1Mod1	01807765	● Auto-discovered
Slot 2 : C Series Temperature Input Module	NI 9213	cDAQ1Mod2	01B9FD79	● Auto-discovered
Slot 3 : C Series Digital Module	NI 9472	cDAQ1Mod3	01241838	● Auto-discovered
Slot 4 : C Series Voltage Output Module	NI 9263	cDAQ1Mod4	012081F6	● Auto-discovered
Slot 7 : C Series Sound and Vibration Input Module	NI 9234	cDAQ1Mod7	01748E8C	● Auto-discovered
Slot 8 : C Series Voltage Input Module	NI 9215	cDAQ1Mod8	[illegible]	● Auto-discovered

0 item(s) selected

Figure 1-2: All the hardware that is connected to your system will be displayed in the Live tab of SystemDesigner.

3. After installing new hardware or software, it is recommended to test the connectivity of the CompactDAQ chassis. You can do this using a **Self-Test**. Click on the NI CompactDAQ Chassis. In the configuration window to the right of the screen, expand the **Troubleshooting** tab, and click **Self-Test**.



Label: Product family
Sub-label: Device name, Model
Discovery: Auto-discovered ●

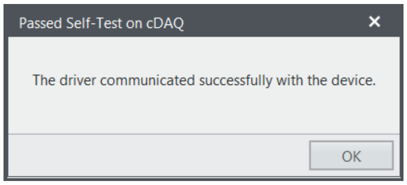
Identity
Device name: cDAQ1
Product family: CompactDAQ Chassis
Vendor: National Instruments
Model: NI cDAQ-9178
Serial number: 017803A8

Label	Model	Device name	Serial number	Hardware Detection
CompactDAQ Chassis	NI cDAQ-9178	cDAQ1	017803A8	● Auto-discovered
Slot 1 : C Series Strain/Bridge Input Module	NI 9236	cDAQ1Mod1	01807765	● Auto-discovered
Slot 2 : C Series Temperature Input Module	NI 9213	cDAQ1Mod2	01B9FD79	● Auto-discovered
Slot 3 : C Series Digital Module	NI 9472	cDAQ1Mod3	01241838	● Auto-discovered
Slot 4 : C Series Voltage Output Module	NI 9263	cDAQ1Mod4	012081F6	● Auto-discovered
Slot 7 : C Series Sound and Vibration Input Module	NI 9234	cDAQ1Mod7	01748E8C	● Auto-discovered
Slot 8 : C Series Voltage Input Module	NI 9215	cDAQ1Mod8	[illegible]	● Auto-discovered

1 item(s) selected

Figure 1-3: After clicking on the NI CompactDAQ chassis, you can access documentation, troubleshooting, and more information about your device in the Configuration pane.

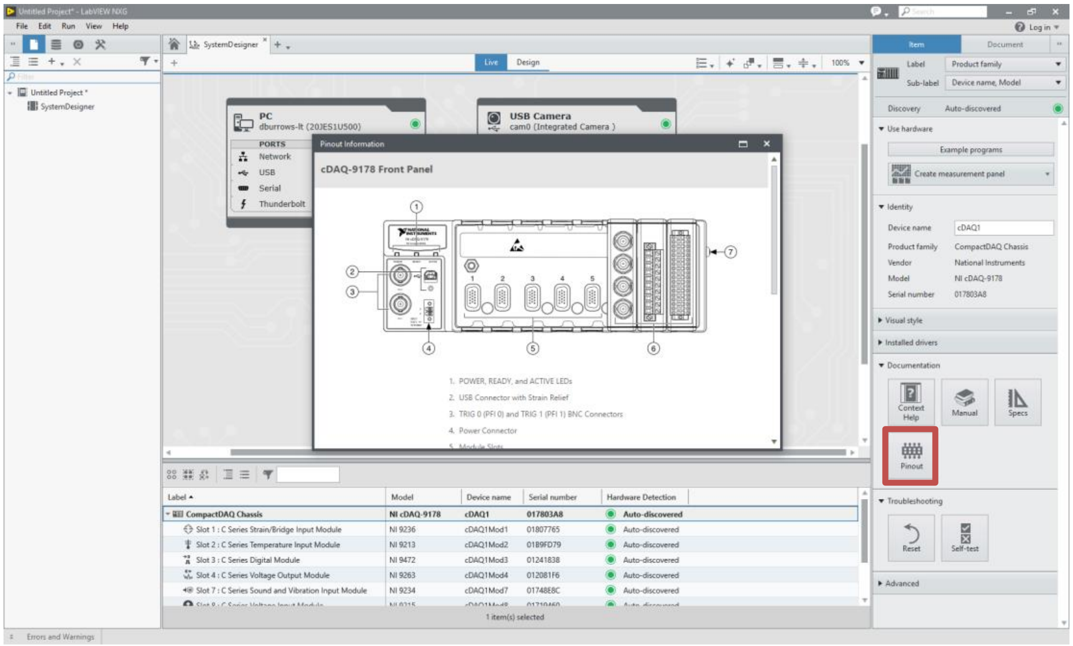
If your chassis is properly connected and able to communicate with your PC, then you will see the following dialog box:



Passed Self-Test on cDAQ

The driver communicated successfully with the device.

4. In the configuration pane to the right of the screen, open the **Documentation** tab. Here, you can view hardware documentation, such as pinouts, manuals, and specifications that are useful to reference when connecting to your device. You can access this information for the CompactDAQ chassis (pictured below), or for individual modules.



1. POWER, READY, and ACTIVE LEDs
2. USB Connector with Strain Relief
3. TRIG 0 (PFI 0) and TRIG 1 (PFI 1) BNC Connectors
4. Power Connector
5. Module Slots

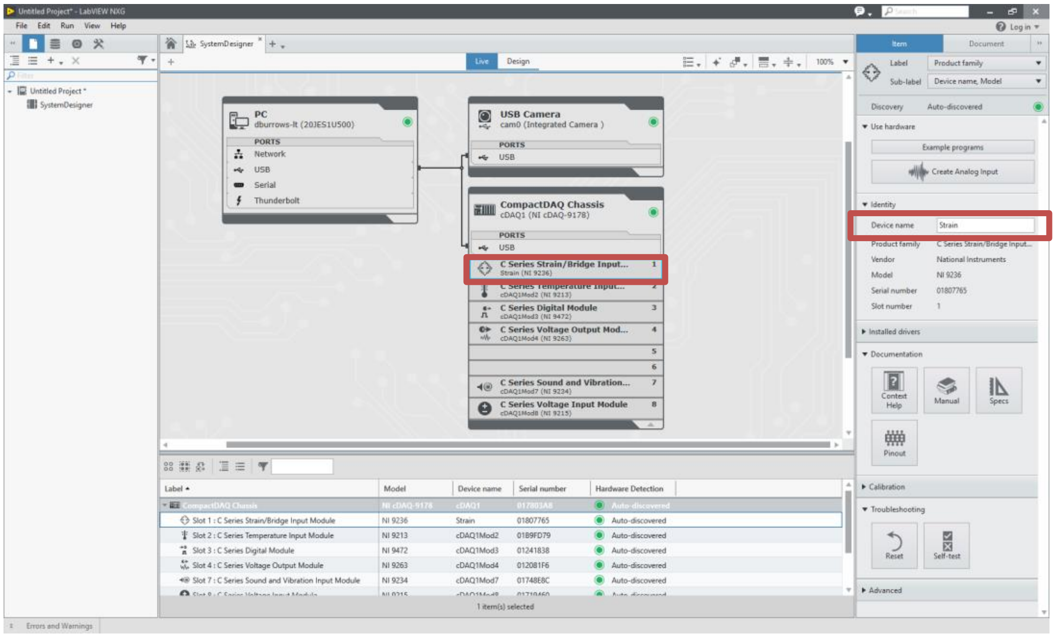
Device name	cDAQ1
Product family	CompactDAQ Chassis
Vendor	National Instruments
Model	NI cDAQ-9178
Serial number	017803A8

Label	Model	Device name	Serial number	Hardware Detection
CompactDAQ Chassis	NI cDAQ-9178	cDAQ1	017803A8	Auto-discovered
Slot 1 : C Series Strain/Bridge Input Module	NI 9236	cDAQ1Mod1	01807765	Auto-discovered
Slot 2 : C Series Temperature Input Module	NI 9213	cDAQ1Mod2	01B9FD79	Auto-discovered
Slot 3 : C Series Digital Module	NI 9472	cDAQ1Mod3	01241838	Auto-discovered
Slot 4 : C Series Voltage Output Module	NI 9263	cDAQ1Mod4	012081F6	Auto-discovered
Slot 7 : C Series Sound and Vibration Input Module	NI 9234	cDAQ1Mod7	01748E8C	Auto-discovered
Slot 8 : C Series Voltage Input Module	NI 9215	cDAQ1Mod8	01719460	Auto-discovered

Figure 1-4:View pinout information to ensure correct connections when setting up hardware.

5. For many applications, simply using the default assigned names is not sufficient. Within LabVIEW NXG, you can rename your modules to match the module type or measurement.

   a. To rename each module, select each module and update the **Device Name** field in the Identity tab of the Configuration Pane.



Label: Product family
Sub-label: Device name, Model
Discovery: Auto-discovered

Identity
Device name: Strain
Product family: C Series Strain/Bridge Input...
Vendor: National Instruments
Model: NI 9236
Serial number: 01807765
Slot number: 1

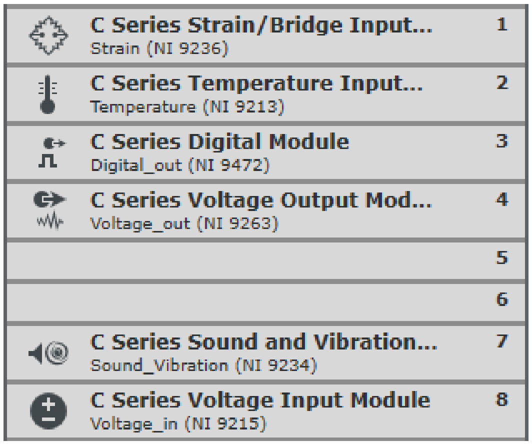
Label ▲	Model	Device name	Serial number	Hardware Detection
CompactDAQ Chassis	NI cDAQ-9178	cDAQ1	017803A8	Auto-discovered
Slot 1 : C Series Strain/Bridge Input Module	NI 9236	Strain	01807765	Auto-discovered
Slot 2 : C Series Temperature Input Module	NI 9213	cDAQ1Mod2	01B9FD79	Auto-discovered
Slot 3 : C Series Digital Module	NI 9472	cDAQ1Mod3	01241838	Auto-discovered
Slot 4 : C Series Voltage Output Module	NI 9263	cDAQ1Mod4	012081F6	Auto-discovered
Slot 7 : C Series Sound and Vibration Input Module	NI 9234	cDAQ1Mod7	01748E8C	Auto-discovered
Slot 8 : C Series Voltage Input Module	NI 9215	cDAQ1Mod8	01719460	Auto-discovered

1 item(s) selected

Figure 1-5: Change the names of modules by updating the name in the Device Name **field.**

b. Rename each of the module to the following descriptive names. These will be used to easily identify your modules in later exercises.

Slot Number	Module	Descriptive Name
1	9236	Strain
2	9213	Temperature
3	9472	Digital_out
4	9263	Voltage_out
7	9234	Sound_Vibration
8	9215	Voltage_in


C Series Strain/Bridge Input... 1
Strain (NI 9236)

C Series Temperature Input... 2
Temperature (NI 9213)

C Series Digital Module 3
Digital_out (NI 9472)

C Series Voltage Output Mod... 4
Voltage_out (NI 9263)

5

6

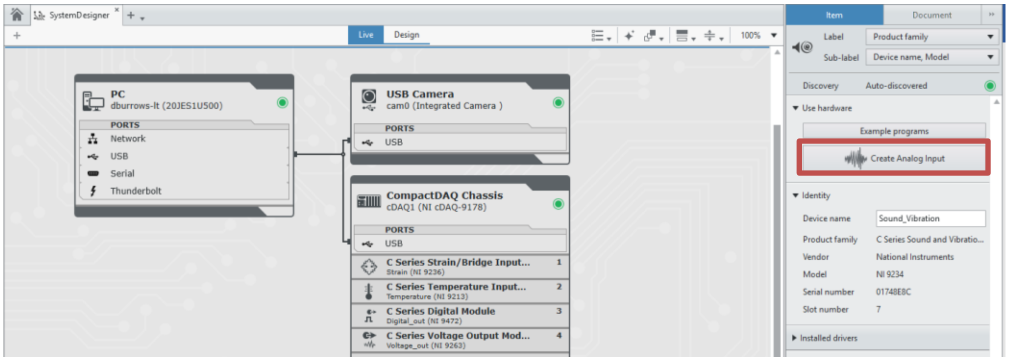
C Series Sound and Vibration... 7
Sound_Vibration (NI 9234)

C Series Voltage Input Module 8
Voltage_in (NI 9215)

Figure 1-6: Name all modules according to the table above.

6. To check the signal connections to the NI CompactDAQ chassis, you will use a Measurement Panel. As an example, we will check the signal connectivity of the NI 9234 analog input, *Sound_Vibration,* module.

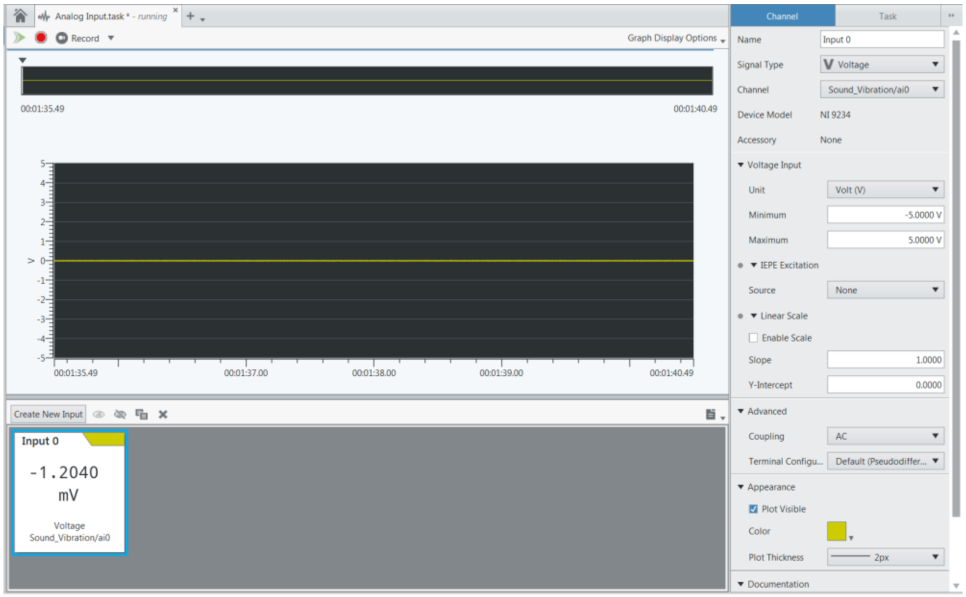
    a. Click on the Sound_Vibration module and select **Create Analog Input…** from the **Use Hardware** tab of the Configuration Pane.



Label: Product family
Sub-label: Device name, Model
Discovery: Auto-discovered
Device name: Sound_Vibration
Product family: C Series Sound and Vibratio…
Vendor: National Instruments
Model: NI 9234
Serial number: 01748E8C
Slot number: 7

Figure 1-4: Measurement Panels let you check your signal connections without writing any code.

    b. Selecting **Create Analog Input** opens a Measurement Panel to check the signals on each of the channels of the module. Immediately, you should see voltage measured scaled to acceleration units from the NI 9234:



Input 0
-1.2040 mV
Voltage
Sound_Vibration/ai0

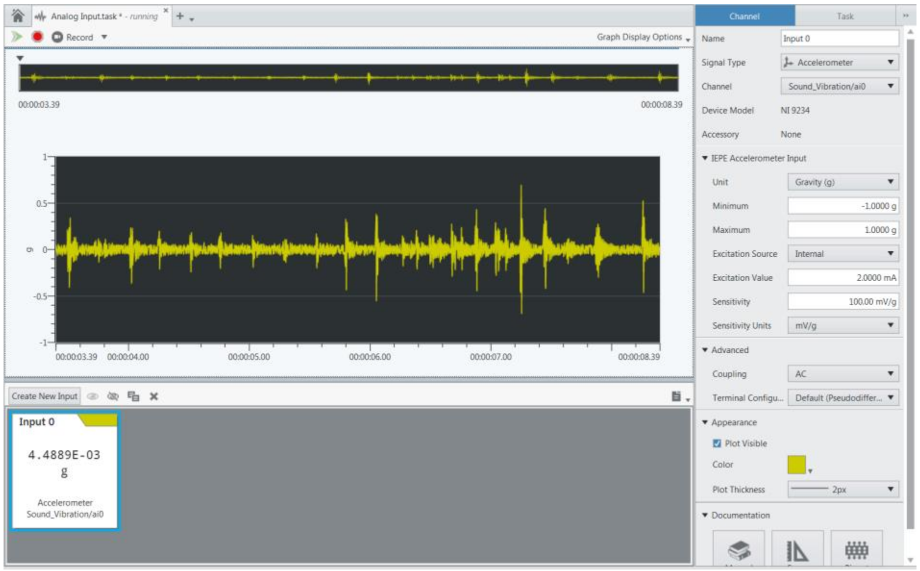
Name: Input 0
Signal Type: Voltage
Channel: Sound_Vibration/ai0
Device Model: NI 9234
Accessory: None
Unit: Volt (V)
Minimum: -5.0000 V
Maximum: 5.0000 V
IEPE Excitation
Source: None
Linear Scale
☐ Enable Scale
Slope: 1.0000
Y-Intercept: 0.0000
Coupling: AC
Terminal Configu…: Default (Pseudodiffer…
☑ Plot Visible
Color: 🟨
Plot Thickness: 2px

Figure 1-5: The Measurement Panel lets you instantly view your data and configure your hardware for your measurement.

    c. Use the Configuration Pane on the right to modify the channel settings for your measurement.

        i. Change the signal type to **Accelerometer.**

        ii. Change the minimum and maximum values to -1 g and +1 g respectively.



Input 0

4.4889E-03
g

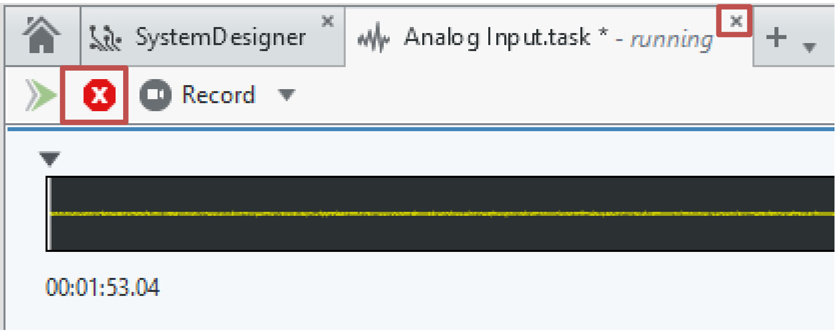
Accelerometer
Sound_Vibration/ai0

Name	Input 0
Signal Type	Accelerometer
Channel	Sound_Vibration/ai0
Device Model	NI 9234
Accessory	None
IEPE Accelerometer Input	
Unit	Gravity (g)
Minimum	-1.0000 g
Maximum	1.0000 g
Excitation Source	Internal
Excitation Value	2.0000 mA
Sensitivity	100.00 mV/g
Sensitivity Units	mV/g
Advanced	
Coupling	AC
Terminal Configu...	Default (Pseudodiffer...
Appearance	
☑ Plot Visible	
Color	🟨
Plot Thickness	2px

Figure 1-6: In the Configuration Pane, you can change settings for your channel including, signal type, units, and sensor configuration.

d. Physically tap on the NI CompactDAQ Demo Box to see the vibration data.

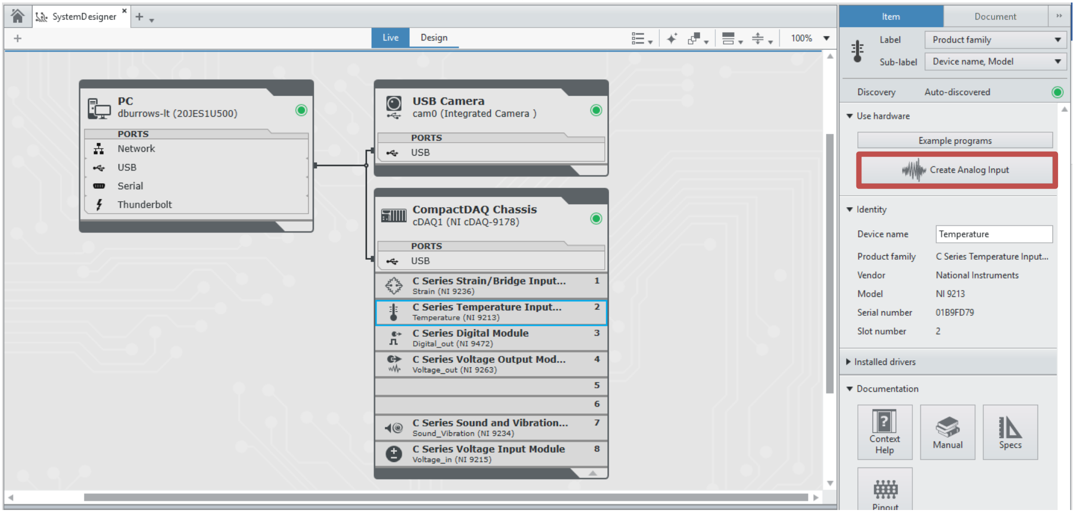
e. When finished, stop the acquisition by pressing the ☒ stop button. Close the Analog Input Task tab.



Part C Take Your First Measurement with LabVIEW NXG

With LabVIEW NXG, you can acquire, record, and analyze your measurements without programming, allowing you to make decisions faster. For this exercise, you will acquire thermocouple data, record signals, and perform a limit test to determine if your temperature readings kept to a defined range.

1. You will use the NI 9213 and a Measurement Panel to acquire thermocouple data. Create a Measurement Panel for the NI 9213 to instantly see incoming temperature data by selecting the 9213 module and clicking **Create Analog Input**, as in Figure 1-7.

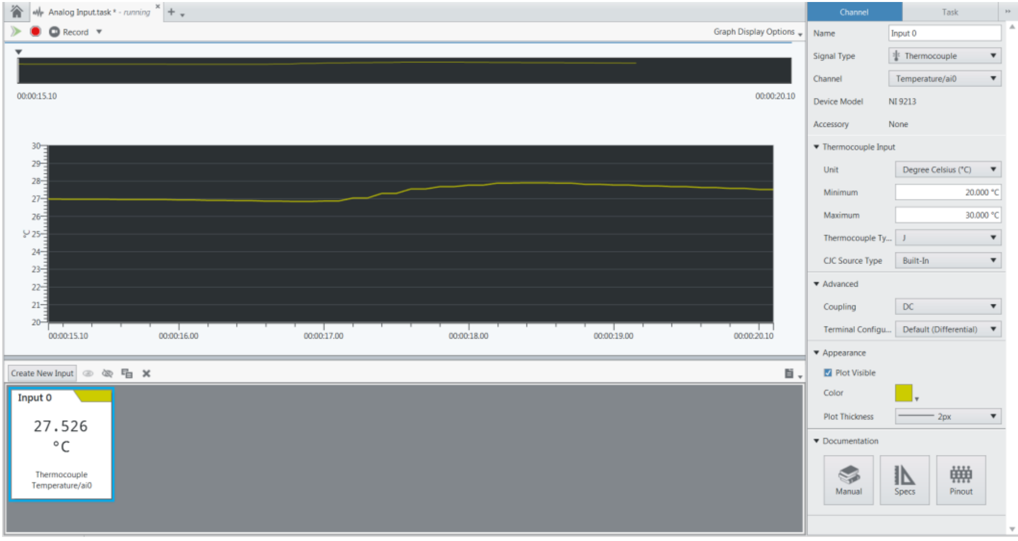


Label: Product family
Sub-label: Device name, Model
Discovery: Auto-discovered

Device name: Temperature
Product family: C Series Temperature Input...
Vendor: National Instruments
Model: NI 9213
Serial number: 01B9FD79
Slot number: 2

Figure 1-7: Create a Measurement Panel to begin acquiring temperature data.

2. Use the Configuration Pane on the right to modify the channel settings for your measurement. Change the minimum and maximum values to 20°C and 30°C respectively.



Input 0
27.526
°C
Thermocouple
Temperature/ai0

Name: Input 0
Signal Type: Thermocouple
Channel: Temperature/ai0
Device Model: NI 9213
Accessory: None

Unit: Degree Celsius (°C)
Minimum: 20.000 °C
Maximum: 30.000 °C
Thermocouple Ty...: J
CJC Source Type: Built-In

Coupling: DC
Terminal Configu...: Default (Differential)

☑ Plot Visible
Color
Plot Thickness: 2px

Figure 1-8: After modifying the channel settings in the Configuration Pane, you will be able to view your temperature.

3. From the Measurement Panel, you can record your signals to view your data later, either in LabVIEW NXG or export the information to another program. You can select how you want to capture data:

    a. Record- save continuous data values until you stop recording
    b. Record (Timed) – save continuous data values over a defined time period
    c. Capture frame – save the current frame of data values

Select to **Record (Timed)** to save temperature data for five seconds. Touch the thermocouple while you are recording to vary the temperature.

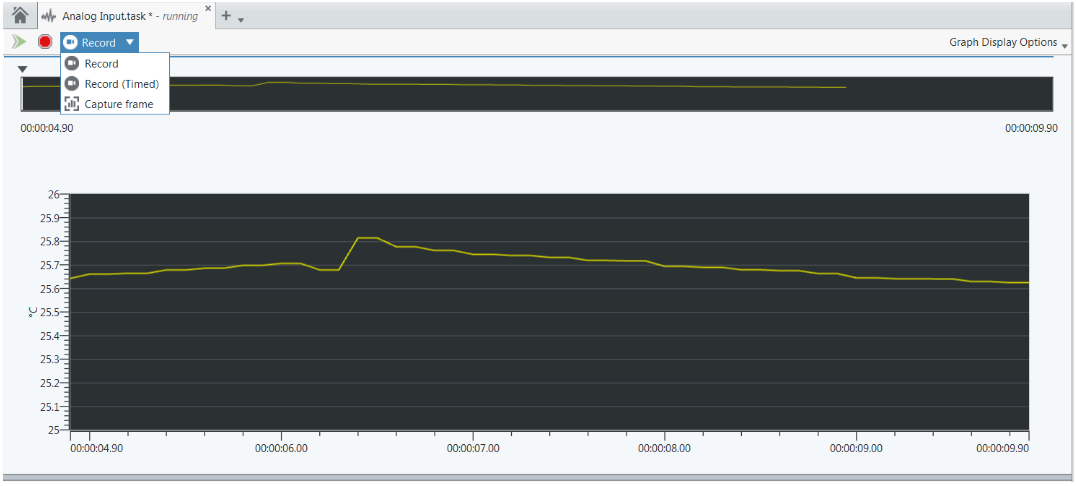
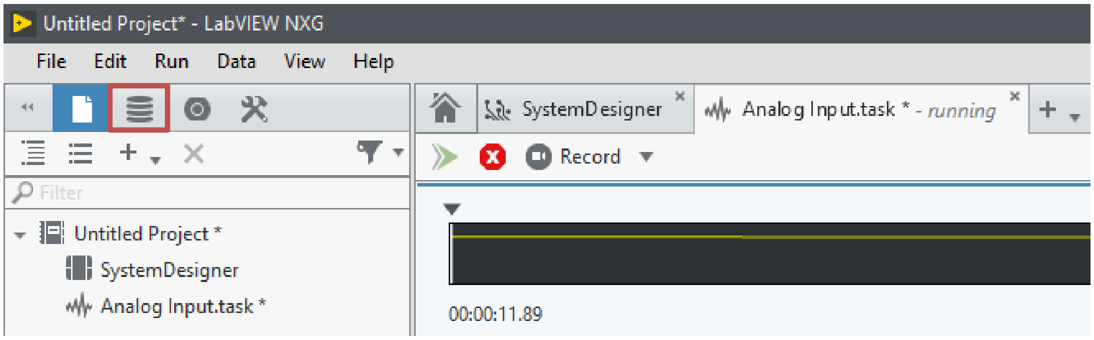
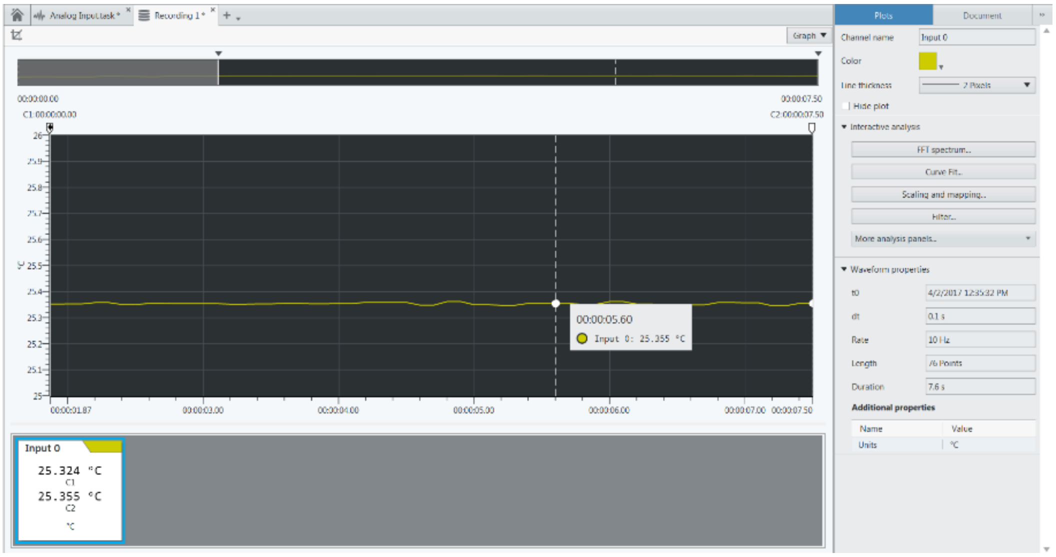


Figure 1-9: Select to perform a timed recording of the thermocouple data.

4. Once your recording is complete, you can find your data in the Data Pane on the left side of the workspace. Anytime you record data in LabVIEW NXG, the captured data will be stored in the Data Pane. Access the data pane by clicking the ≡ symbol:



5. Double-click on your recording to open the Data Viewer:



00:00:05.60
Input 0: 25.355 °C

Input 0
25.324 °C
C1
25.355 °C
C2

Channel name: Input 0
Color
Line thickness: 2 Pixels
□ Hide plot

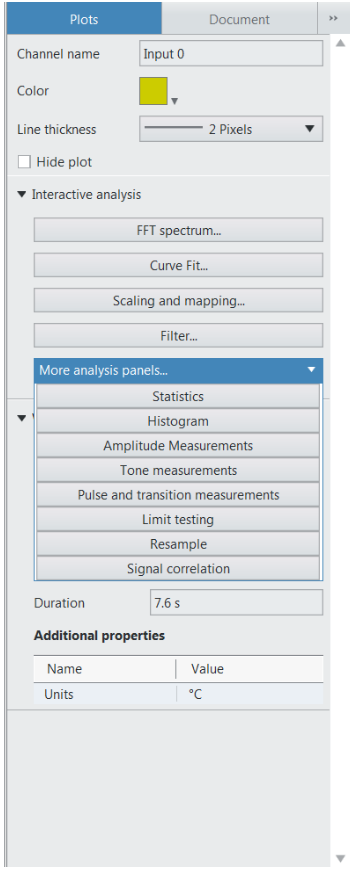
t0: 4/2/2017 12:35:32 PM
dt: 0.1 s
Rate: 10 Hz
Length: 76 Points
Duration: 7.6 s

Additional properties

Name	Value
Units	°C

Figure 1-10: In the Data Viewer, you can use the cursors to focus on a particular segment of the signal or to view individual data points.

6. In the Data Viewer, you can apply an analysis function to the recorded signal to gain insight on your data. Under **Interactive Analysis** in the Configuration Pane on the right-side, select the **More analysis panels…** to view the analysis functions that are available.



Channel name: Input 0
Color
Line thickness: 2 Pixels
□ Hide plot

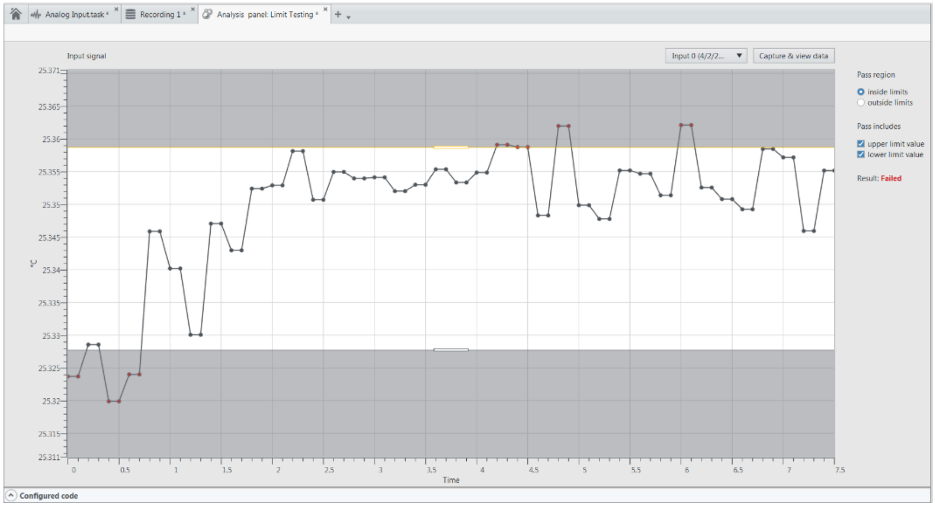
Duration: 7.6 s

Additional properties

Name	Value
Units	°C

Figure 1-11: In the Configuration Pane, you can select a variety of analysis functions to interactively apply to your data.

7. From the drop-down menu, select **Limit testing** to open an Analysis Panel to perform a limit test on your recorded temperature data. With this Analysis Panel, you can set parameters of the limit test, such as upper and lower limit values, to instantly see the results of your analysis. Use the cursors on the graph to set your upper and lower limit values.



Pass region

● inside limits
○ outside limits

Pass includes

☑ upper limit value
☑ lower limit value

Result: Failed

Figure 1-12: You can view in real-time how the effect of modifying analysis parameters on your data.

8. Save this project as **CompactDAQ Project** to your desktop by navigating to **File** >> **Save all** and browsing to the Desktop. The hardware configuration and analysis function you used in this exercise will be used in a later exercise.

<End of Exercise>

Chapter 2 Using Examples to Automate Measurements with LabVIEW NXG

Goals	• Explore the lessons and examples that are included with LabVIEW NXG. • Run a pre-built example program to automate your first measurement. • Use configurations from the first exercise to start automating from the beginning • Experiment with other examples (optional)

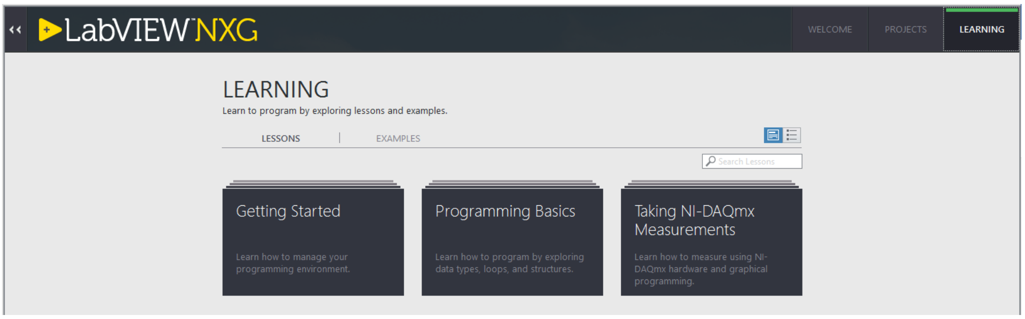
Part A Explore Lessons and Examples in LabVIEW NXG

Reduce the time spent taking manual measurements by automating acquiring your data using LabVIEW NXG and NI-DAQmx. Furthermore, customize your measurements with built-in analysis and storage functions that allow you to meet your project requirements.

The NI-DAQmx driver allows you to interface with NI data acquisition hardware and also installs examples that 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 NXG by pressing the **Windows key** and selecting **LabVIEW NXG.** From the welcome scree, select **Learn to Program.**

2. The Learning tab provides tutorials and examples that get started on how to take measurements in LabVIEW NXG. Use tutorials to gain knowledge on specific topics, such as programming basics and generating reports, and use examples to understand how software integrates with hardware in real-world applications.



Learn to program by exploring lessons and examples.

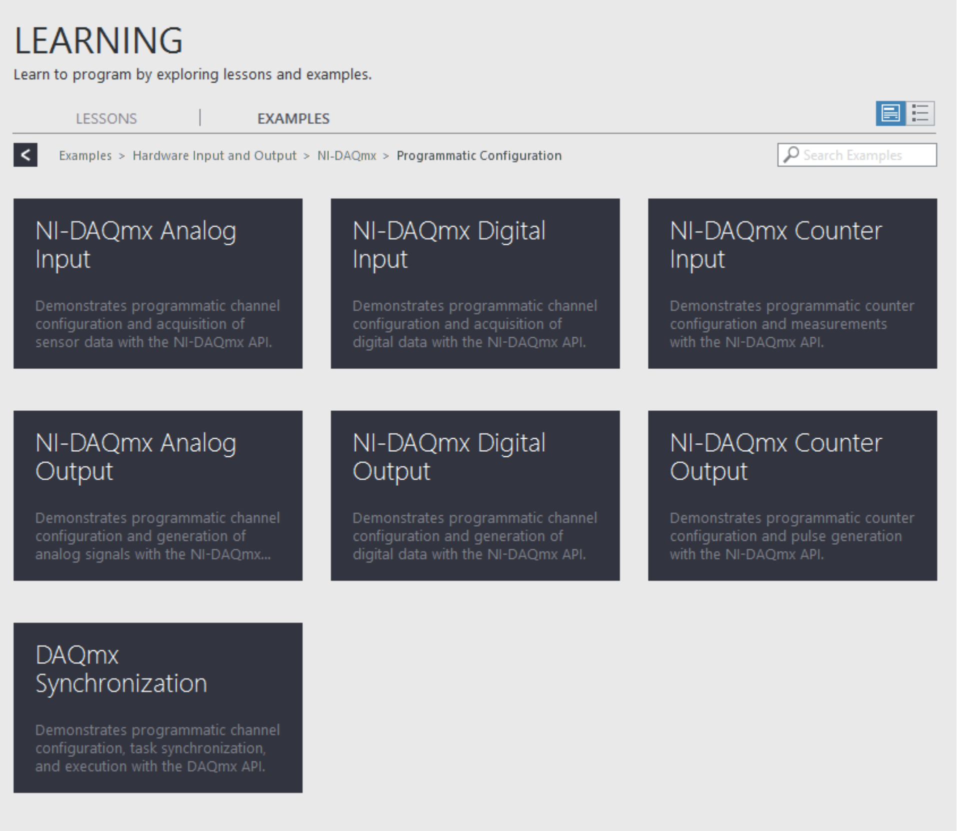
Learn how to manage your programming environment.

Learn how to program by exploring data types, loops, and structures.

Learn how to measure using NI-DAQmx hardware and graphical programming.

Figure 2-1: In the Learning tab, access getting started material, tutorials on learning programming concepts, and examples on how to interface with your hardware.

3. Today, we will be using an example as a starting point for customizing and automating our code. To open the NI-DAQmx examples select **Examples » Hardware Input and Output » NI-DAQmx » Programmatic Configuration.** In this window, you can find example projects on the types of measurements you can perform with the NI-DAQmx API.



LEARNING

Learn to program by exploring lessons and examples.

LESSONS
EXAMPLES
Examples > Hardware Input and Output > NI-DAQmx > Programmatic Configuration
Search Examples

NI-DAQmx Analog Input
Demonstrates programmatic channel configuration and acquisition of sensor data with the NI-DAQmx API.

NI-DAQmx Digital Input
Demonstrates programmatic channel configuration and acquisition of digital data with the NI-DAQmx API.

NI-DAQmx Counter Input
Demonstrates programmatic counter configuration and measurements with the NI-DAQmx API.

NI-DAQmx Analog Output
Demonstrates programmatic channel configuration and generation of analog signals with the NI-DAQmx...

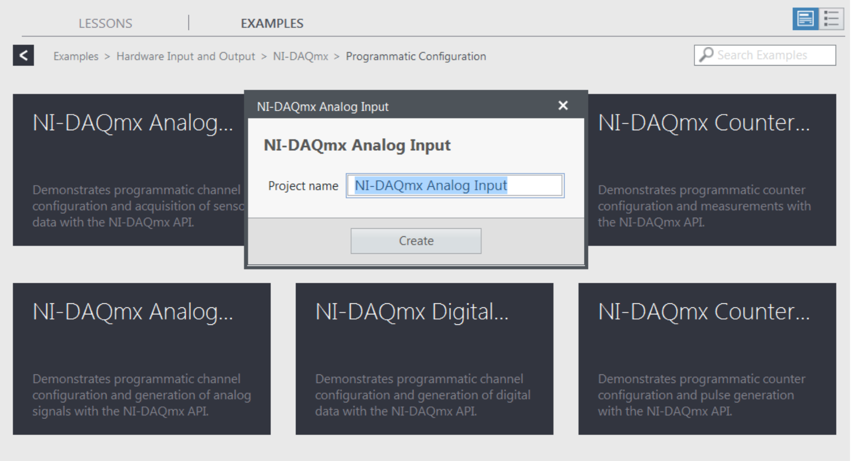
NI-DAQmx Digital Output
Demonstrates programmatic channel configuration and generation of digital data with the NI-DAQmx API.

NI-DAQmx Counter Output
Demonstrates programmatic counter configuration and pulse generation with the NI-DAQmx API.

DAQmx Synchronization
Demonstrates programmatic channel configuration, task synchronization, and execution with the DAQmx API.

Figure 2-2: Select an NI-DAQmx example project for your measurement type.

4. Select **NI-DAQmx Analog Input** to create a project that contains several examples on how to perform different types of analog input using LabVIEW NXG.



LESSONS
EXAMPLES
Examples > Hardware Input and Output > NI-DAQmx > Programmatic Configuration
Search Examples
Create

NI-DAQmx Analog...
Demonstrates programmatic channel configuration and acquisition of senso data with the NI-DAQmx API.

NI-DAQmx Analog Input ✕

NI-DAQmx Analog Input

Project name: NI-DAQmx Analog Input

NI-DAQmx Counter...
Demonstrates programmatic counter configuration and measurements with the NI-DAQmx API.

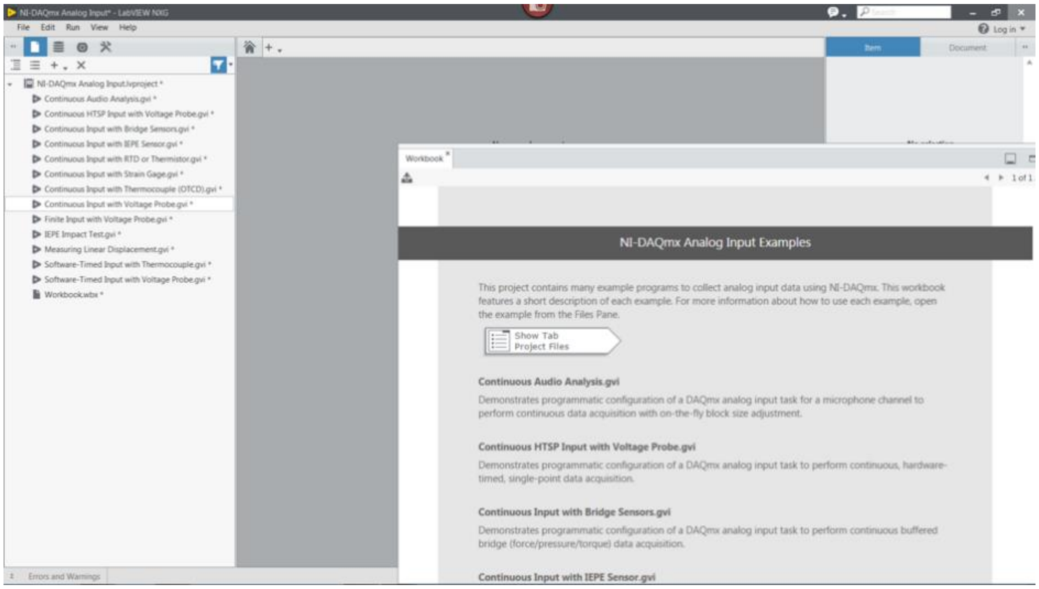
NI-DAQmx Analog...
Demonstrates programmatic channel configuration and generation of analog signals with the NI-DAQmx API.

NI-DAQmx Digital...
Demonstrates programmatic channel configuration and generation of digital data with the NI-DAQmx API.

NI-DAQmx Counter...
Demonstrates programmatic counter configuration and pulse generation with the NI-DAQmx API.

Figure 2-3: The NI-DAQmx Analog Input example project provides examples for different types of sensor measurements.

5. Workbooks accompany example projects and are documentation that can guide you through a concept. The workbook will open automatically – you can also open **Workbook.wbx** in the left pane to read about the examples that are included in the NI-DAQmx Analog Input project.


NI-DAQmx Analog Input* - LabVIEW NXG
File
Edit
Run
View
Help
NI-DAQmx Analog Input.lvproject *
Continuous Audio Analysis.gvi *
Continuous HTSP Input with Voltage Probe.gvi *
Continuous Input with Bridge Sensors.gvi *
Continuous Input with IEPE Sensor.gvi *
Continuous Input with RTD or Thermistor.gvi *
Continuous Input with Strain Gage.gvi *
Continuous Input with Thermocouple (OTCD).gvi *
Continuous Input with Voltage Probe.gvi *
Finite Input with Voltage Probe.gvi *
IEPE Impact Test.gvi *
Measuring Linear Displacement.gvi *
Software-Timed Input with Thermocouple.gvi *
Software-Timed Input with Voltage Probe.gvi *
Workbook.wbx *
Errors and Warnings
Item
Document
Log in
Workbook
1 of 1
Show Tab
Project Files

NI-DAQmx Analog Input Examples

This project contains many example programs to collect analog input data using NI-DAQmx. This workbook features a short description of each example. For more information about how to use each example, open the example from the Files Pane.

Continuous Audio Analysis.gvi

Demonstrates programmatic configuration of a DAQmx analog input task for a microphone channel to perform continuous data acquisition with on-the-fly block size adjustment.

Continuous HTSP Input with Voltage Probe.gvi

Demonstrates programmatic configuration of a DAQmx analog input task to perform continuous, hardware-timed, single-point data acquisition.

Continuous Input with Bridge Sensors.gvi

Demonstrates programmatic configuration of a DAQmx analog input task to perform continuous buffered bridge (force/pressure/torque) data acquisition.

Continuous Input with IEPE Sensor.gvi

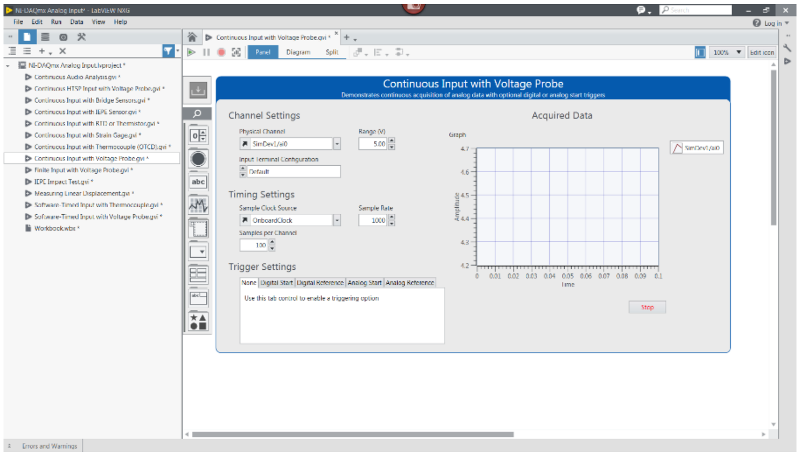
Figure 2-4: A workbook is a media-rich document that accompanies a project to provide documentation for code.

Part B Run a Pre-Built Example

Now that you are familiar with 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-DAQmx Analog Input project, open **Continuous Input with Voltage Probe.gvi**. 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 example, notice the controls for selecting the channel, voltage input ranges, timing parameters, and trigger settings. Additionally, the front panel includes a graph indicator for instantly viewing the data on the voltage channels.



Continuous Input with Voltage Probe

Demonstrates continuous acquisition of analog data with optional digital or analog start triggers

Channel Settings

Physical Channel: SimDev1/ai0

Range (V): 5.00

Input Terminal Configuration: Default

Timing Settings

Sample Clock Source: OnboardClock

Sample Rate: 1000

Samples per Channel: 100

Trigger Settings

None | Digital Start | Digital Reference | Analog Start | Analog Reference

Use this tab control to enable a triggering option

NI-DAQmx Analog Input* - LabVIEW NXG
File
Edit
Run
Data
View
Help
NI-DAQmx Analog Input.lvproject *
Continuous Audio Analysis.gvi *
Continuous HTSP Input with Voltage Probe.gvi *
Continuous Input with Bridge Sensors.gvi *
Continuous Input with IEPE Sensor.gvi *
Continuous Input with RTD or Thermistor.gvi *
Continuous Input with Strain Gage.gvi *
Continuous Input with Thermocouple (OTCD).gvi *
Continuous Input with Voltage Probe.gvi *
Finite Input with Voltage Probe.gvi *
IEPE Impact Test.gvi *
Measuring Linear Displacement.gvi *
Software-Timed Input with Thermocouple.gvi *
Software-Timed Input with Voltage Probe.gvi *
Workbook.wbx *
Continuous Input with Voltage Probe.gvi *
Panel
Diagram
Split
100%
Edit icon
Search
Log in
Acquired Data
Graph
Amplitude
4.7
4.6
4.5
4.4
4.3
4.2
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
Time
SimDev1/ai0
Stop
Errors and Warnings

Figure 2-5: The front panel is the user interface of a VI and has controls and indicators that allow users to interact with the code.

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

4. Zoom out of the block diagram window (**<Ctrl+Mouse Scroll Wheel>**) 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 NXG uses dataflow to pass information along wires to sequential VIs. This programming paradigm enables you to better program like you think.

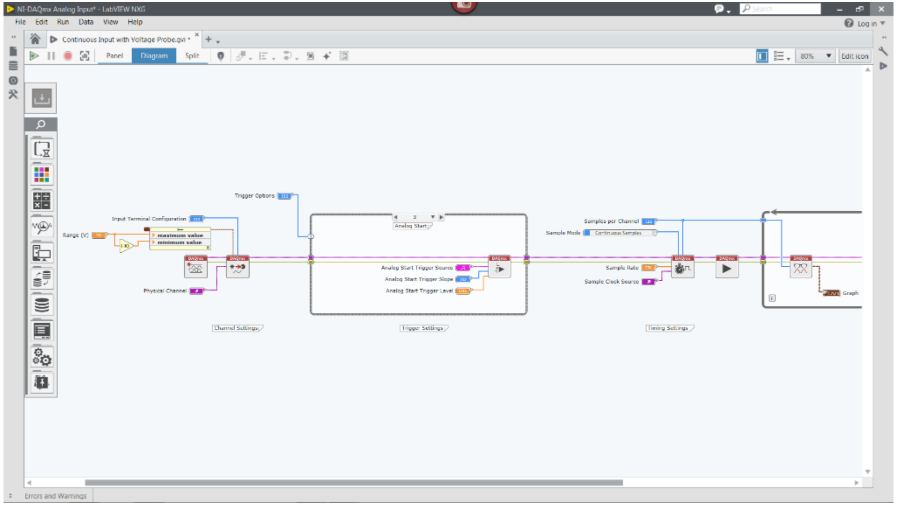


Figure 2-6: LabVIEW NXG uses a graphical programming syntax that passes data through the code via wires that connect nodes.

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. **Trigger Settings –** In the Trigger Settings section, the *DAQmx Start Trigger (Digital Edge)* is used to start acquiring samples on a rising or falling edge of a digital signal. Depending on the input from the user on the front panel, either the acquisition will be triggered by a digital signal or analog signal or will automatically begin. The case structure surrounding this VI will determine which trigger type is used.

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:

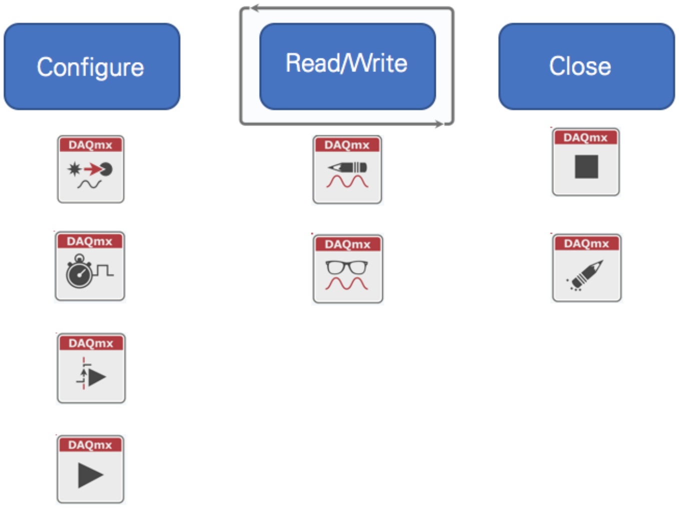
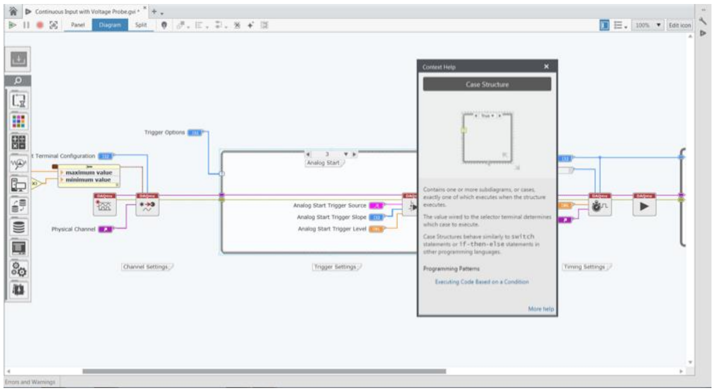


Figure 2-7: The most common VIs for each section are shown above.

6. Compare the example program block diagram to the VIs shown in Figure 2-7. All DAQmx programs will follow the same configure >> Read/Write >> Close prototype, using some or all of these functions.

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 **More help** link. Try hovering over a few elements on your block diagram.



Context Help

Case Structure

Contains one or more subdiagrams, or cases, exactly one of which executes when the structure executes.

The value wired to the selector terminal determines which case to execute.

Case Structures behave similarly to `switch` statements or `if-then-else` statements in other programming languages.

Programming Patterns

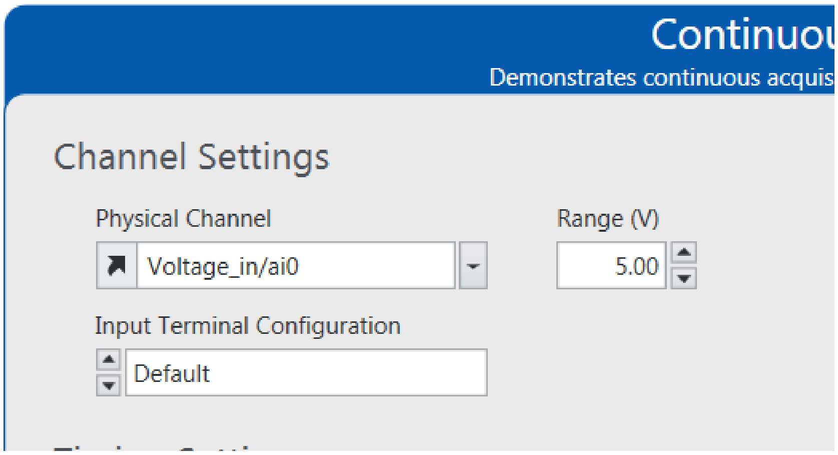
Executing Code Based on a Condition

More help

Figure 2-8: 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 **Panel** or by pressing **Ctrl+E**.

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



Continuo

Demonstrates continuous acquis

Channel Settings

Physical Channel: Voltage_in/ai0

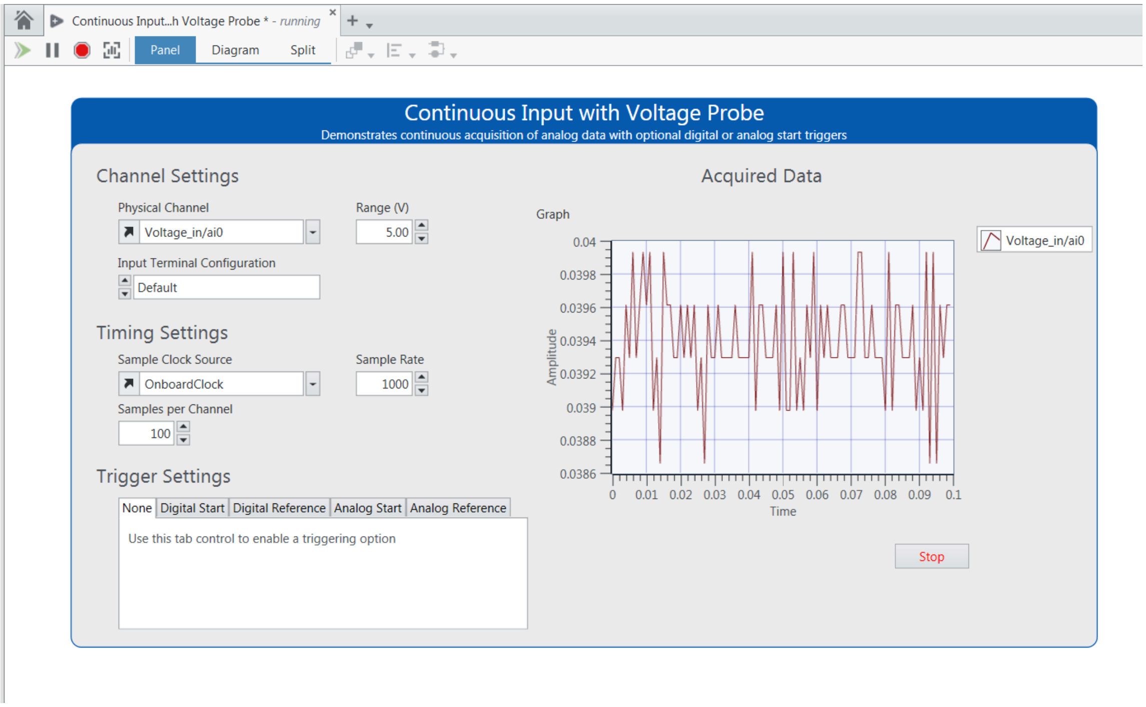
Range (V): 5.00

Input Terminal Configuration: Default

Figure 2-9: 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?



Continuous Input with Voltage Probe

Demonstrates continuous acquisition of analog data with optional digital or analog start triggers

Channel Settings

Physical Channel: Voltage_in/ai0

Range (V): 5.00

Input Terminal Configuration: Default

Timing Settings

Sample Clock Source: OnboardClock

Sample Rate: 1000

Samples per Channel: 100

Trigger Settings

None | Digital Start | Digital Reference | Analog Start | Analog Reference

Use this tab control to enable a triggering option

Acquired Data

Figure 2-10: With this program, you are sampling a channel at 10 kHz. Throughout this seminar, you will modify and customize examples to automate many tasks.

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

12. Exit the program without saving.

Part C Experiment with Other Example Programs (Optional)

Try exploring other example programs by returning to **Programmatic Configuration (Help » Examples)** page. Examples that might be useful to look include:

1. **NI-DAQmx Analog Input** project – Continuous Input with Thermocouple.gvi
   Note:
   - Be sure to select **Temperature/ai0** from the Physical Channel control.

2. **NI-DAQmx Analog Input** project – Continuous Input with Strain Gage.gvi
   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. **NI-DAQmx Digital Output** project – Continuous Digital Output.gvi
   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. **NI-DAQmx Analog Output** project – Continuous (No Regeneration) Output Voltage.gvi
   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, return to **Explore Your Hardware** and reset the NI 9263 by selecting the **Voltage_out** module and choosing **Reset**.

<End of Exercise>

Chapter 3 Temperature Measurement and Logging

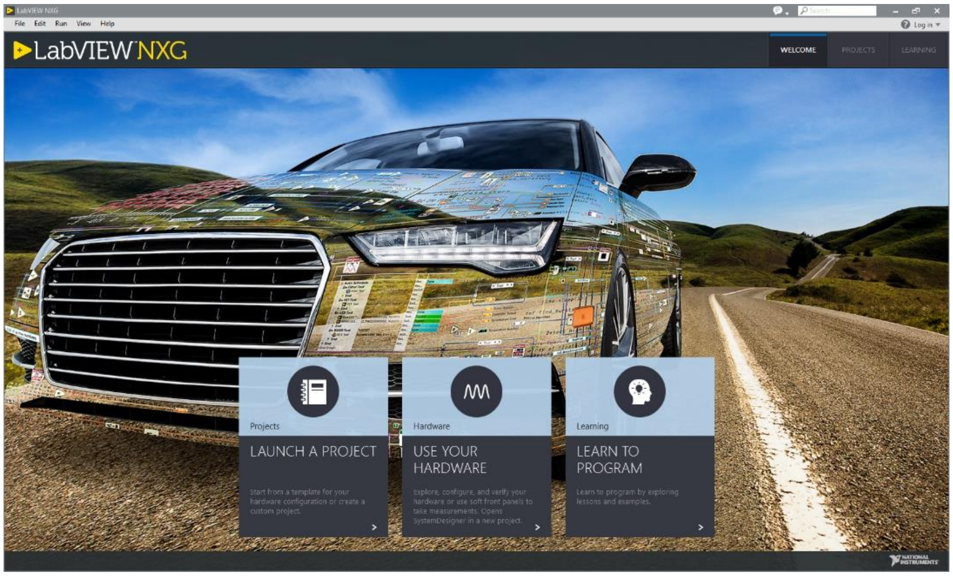
Goals

- The first goal is to quickly acquire a set of temperature data by reusing the thermocouple measurement configuration in an application.
- 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 NXG environment.

1. If you have not already done so, launch LabVIEW NXG by navigating to **Windows » LabVIEW NXG.** Once you launch LabVIEW NXG, the welcome screen window appears:



Start from a template for your hardware configuration or create a custom project.

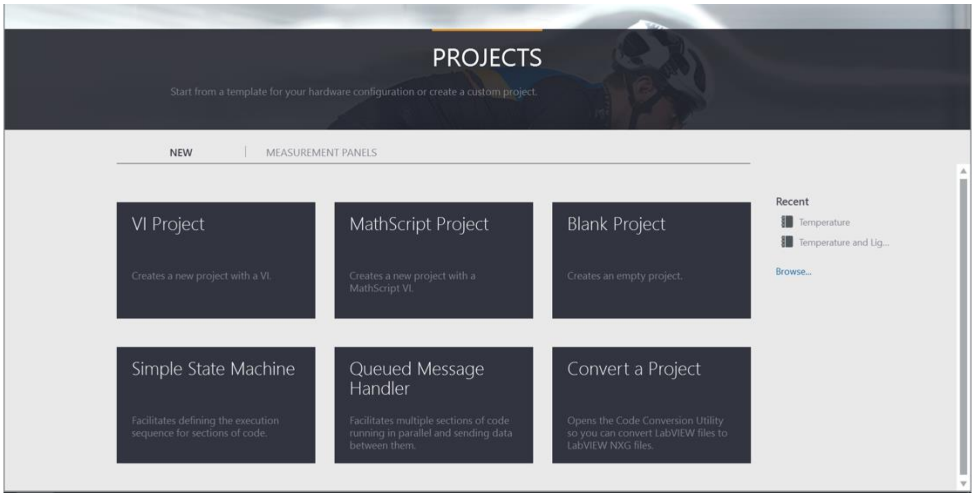
Explore, configure, and verify your hardware or use soft front panels to take measurements. Opens SystemDesigner in a new project.

Learn to program by exploring lessons and examples.

Figure 3-1 The LabVIEW NXG welcome screen provides several paths for exploring your hardware, beginning a project, and accessing learning materials.

The LabVIEW NXG welcome screen appears each time you launch LabVIEW to assist you in creating new applications or opening existing applications.

2. Select **Launch a Project** to open an existing project.



PROJECTS

Start from a template for your hardware configuration or create a custom project.

VI Project

Creates a new project with a VI.

MathScript Project

Creates a new project with a MathScript VI.

Blank Project

Creates an empty project.

Simple State Machine

Facilitates defining the execution sequence for sections of code.

Queued Message Handler

Facilitates multiple sections of code running in parallel and sending data between them.

Convert a Project

Opens the Code Conversion Utility so you can convert LabVIEW files to LabVIEW NXG files.

Recent

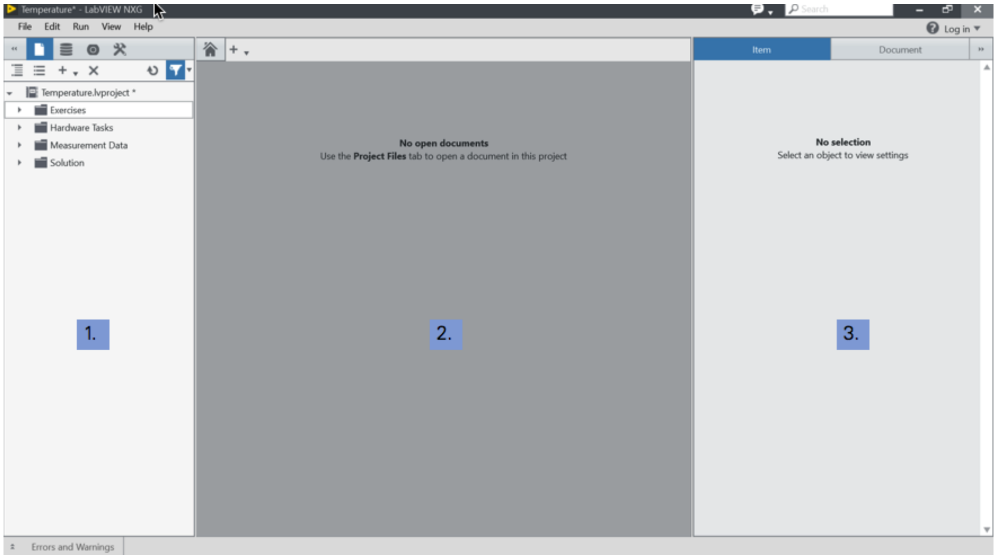
Temperature

Temperature and Lig...

Browse...

Figure 3-2: Under the Projects tab, you have the option to start a project from scratch or use a template using standard architecture.

Navigate to *C:\Seminars\cDAQ + LabVIEW NXG HO\Exercises\3 - Temperature* and select the **Temperature.lvproj** project. Once the project opens, you will see three distinct sections of the LabVIEW NXG editor.



No open documents
Use the **Project Files** tab to open a document in this project

No selection
Select an object to view settings

Figure 3-3: The LabVIEW NXG editor.

1. Navigation Pane – Repository of all files, data captures, and debugging tools associated with a LabVIEW NXG project.
2. Document – Measurement Panel, Analysis Panel, or a VI.
3. Configuration Pane – Modify hardware configurations, analysis parameters, and objects used in VIs.

The Navigation Pane includes a Project Viewer which provides a central location for you to include the different elements of an application including LabVIEW NXG code and other files like Microsoft Word and Excel documents. 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.

3. Expand the **Solutions** folder in the Project Viewer and open the *4 - Write to File (Solution).gvi* by double-clicking on it or right-clicking and selecting **Open**.
   Every LabVIEW NXG 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 **Panel** or **Diagram** to see the other window. You can also switch between the windows by pressing **Ctrl+E** on the keyboard.

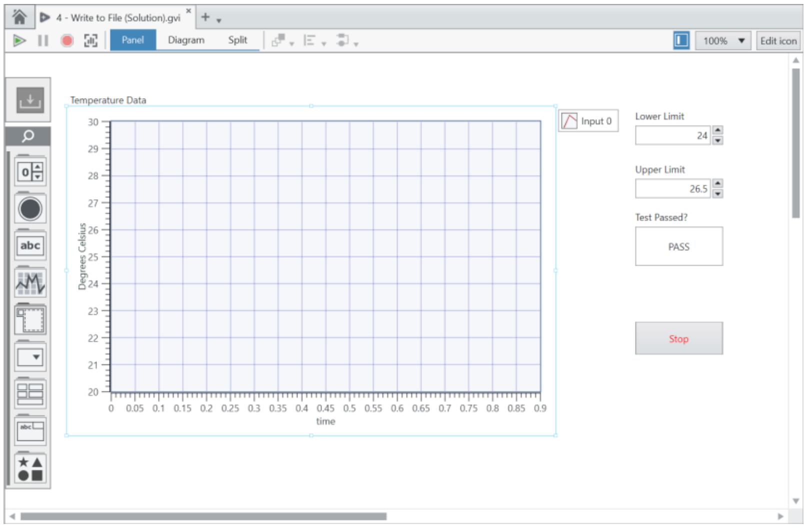


Figure 3-4: The Front Panel of a VI allows for a user to interface with the code executing on the block diagram.

Hover the cursor over the different objects on the front panel. Notice that your cursor turns into a move cursor when above the Temperature Data graph, and turns into a text editor when above a text field. By default, 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.

4. In the menu bar at the top of the window you can control when the application begins to execute by using the **Run** button.

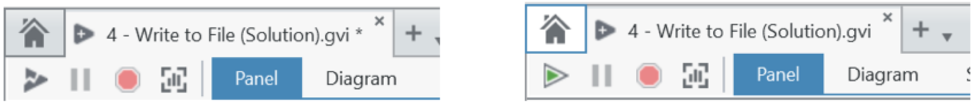

4 - Write to File (Solution).gvi *
Panel
Diagram
4 - Write to File (Solution).gvi
Panel
Diagram

Figure 3-5: A broken run arrow (left) indicates that there is an error compiling this application.

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

5. 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 NXG application and watch as the application begins to record temperature data from the module plugged into the second slot of the CompactDAQ chassis. Contact the instructor if your application isn't running as described.

6. Hold the end of the thermocouple and watch the values on the graph rise and fall accordingly. Change the **Upper Limit** 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

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

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

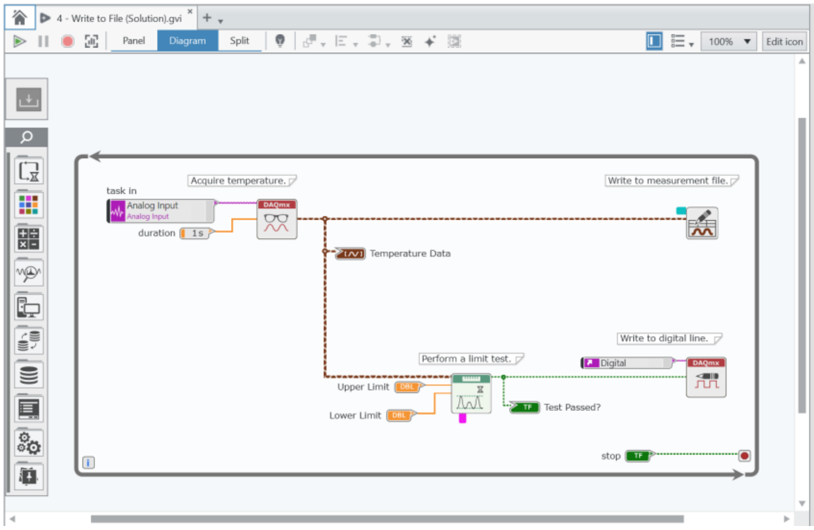


Figure 3-6: The block diagram is the source code of an application.

The graphical programming syntax in LabVIEW NXG uses dataflow to control the flow of an application. In this case our application does the following:

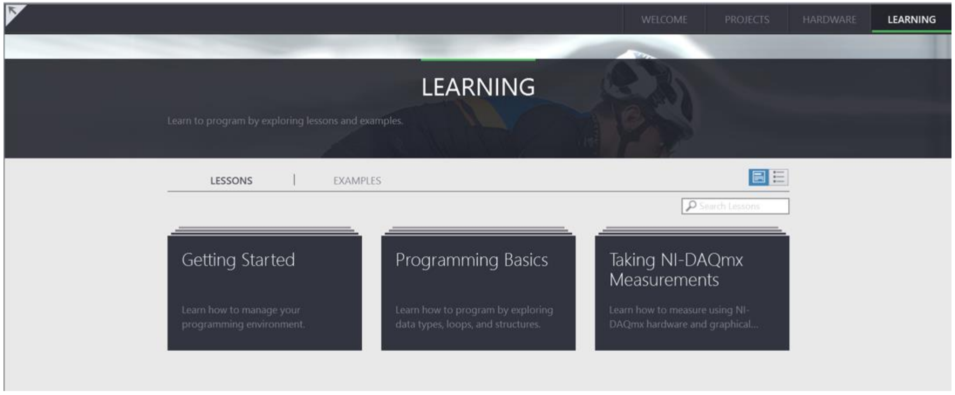
1.Acquires temperature data using the NI-DAQmx Read function and displays it on a chart.
2.Compares acquired data with Upper and Lower Limits.
3.Outputs 0 V or 5 V to the digital output module based on the comparison in #2.
4.Writes acquired data to file.

9. Select to **Split** View the front panel and block diagram windows so that both are visible.

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 to be used block diagram.

Additional Steps

10. Learning and Help documentation included in LabVIEW NXG is an effective way to learn about LabVIEW NXG and answer your programming questions. Select **Help » Lessons** to view the lessons available to get started in the LabVIEW NXG



Learn to program by exploring lessons and examples.

Getting Started

Learn how to manage your programming environment.

Programming Basics

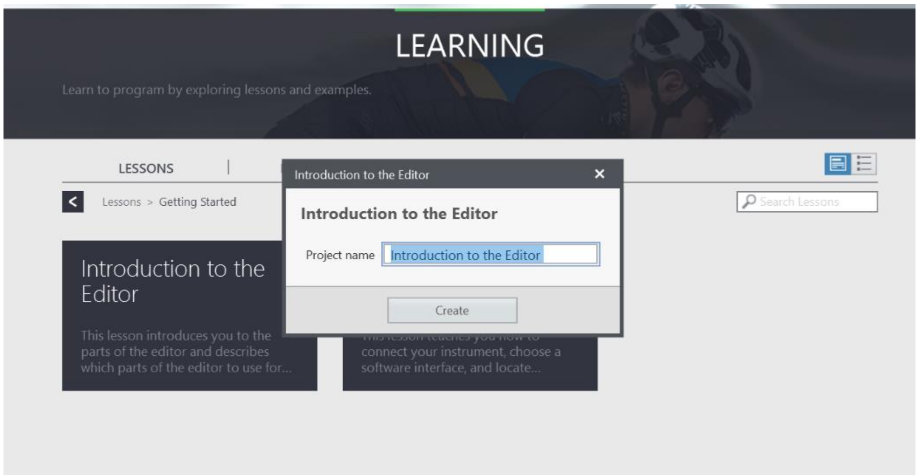
Learn how to program by exploring data types, loops, and structures.

Taking NI-DAQmx Measurements

Learn how to measure using NI-DAQmx hardware and graphical...

Figure 3-7: In LabVIEW NXG, lessons are available to get acclimated to the environment, learn new programming concepts, and discover how to begin taking hardware measurements.

11. Within the Learning tab, navigate to **Getting Started» Introduction to the Editor** to open a project that provides an overview of the LabVIEW NXG environment. Feel free to explore and get a feel for the depth of content and how it is organized.



Learn to program by exploring lessons and examples.

Introduction to the Editor

Project name: Introduction to the Editor

Introduction to the Editor

This lesson introduces you to the parts of the editor and describes which parts of the editor to use for...

This lesson teaches you how to connect your instrument, choose a software interface, and locate...

Figure 3-8: The Introduction to the Editor section provides an overview of each section of the environment and how to use learning materials.

12. Close the Introduction to the Editor project without saving.

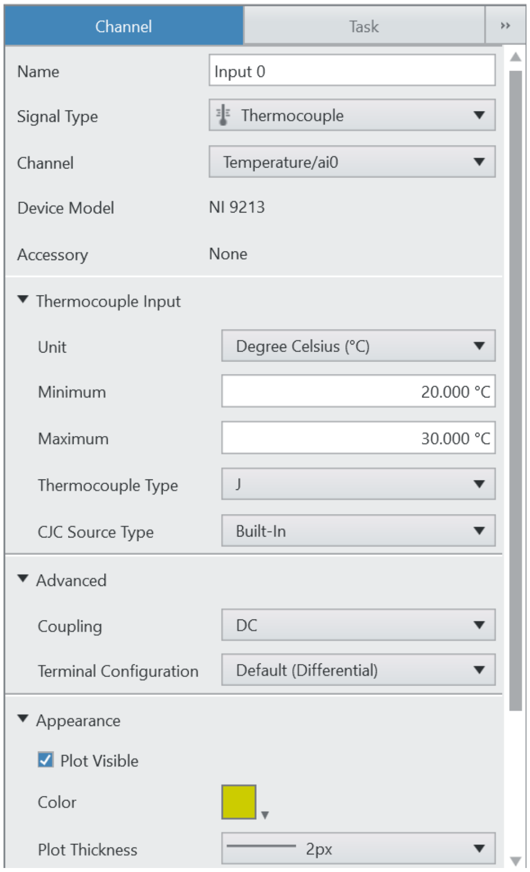
Part B

In Part B, we will build the program we explored in Part A. The purpose of this exercise is to use LabVIEW and NI CompactDAQ to quickly set up a program to acquire temperature data.

1. In the LabVIEW project view, right-click the Exercises folder and select **Add Folder**. Name it *Hardware Tasks*.

2. Right-click the Hardware Tasks folder and select **Add New » Analog Input**. Once opened, save the Analog Input task in the Hardware Tasks folder under the name *Analog Input.task*.

3. To configure a temperature measurement with a thermocouple, configure the Channel parameters in the configuration pane as shown in Figure 3-9.



Name: Input 0
Signal Type: Thermocouple
Channel: Temperature/ai0
Device Model: NI 9213
Accessory: None

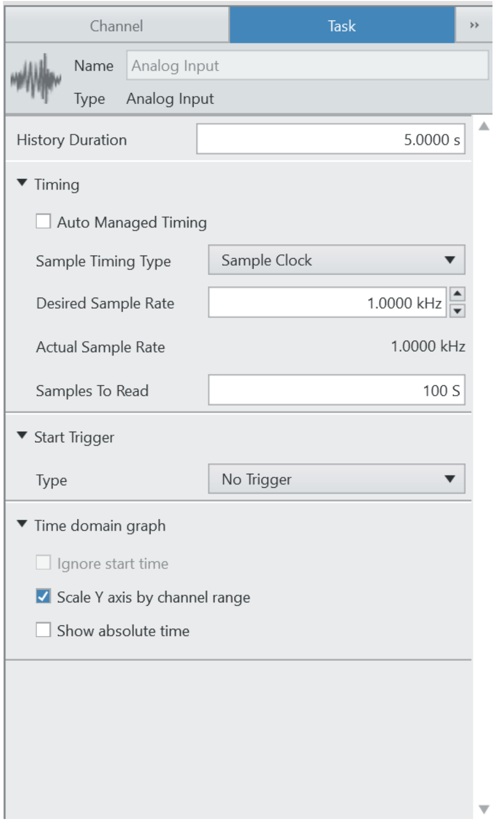
▼ Thermocouple Input
Unit: Degree Celsius (°C)
Minimum: 20.000 °C
Maximum: 30.000 °C
Thermocouple Type: J
CJC Source Type: Built-In

▼ Advanced
Coupling: DC
Terminal Configuration: Default (Differential)

▼ Appearance
☑ Plot Visible
Color: 🟨
Plot Thickness: 2px

Figure 3-9: The Channel tab in the Configuration Pane is used to modify the settings for one channel.

4. Switch to the Task tab in the Configuration Pane. Change the *Samples to Read* to **100** and change the *Desired Sample Rate* to **1kHz**, as shown below.



Channel | Task

Name: Analog Input

Type: Analog Input

History Duration: 5.0000 s

▼ Timing

□ Auto Managed Timing

Sample Timing Type: Sample Clock

Desired Sample Rate: 1.0000 kHz

Actual Sample Rate: 1.0000 kHz

Samples To Read: 100 S

▼ Start Trigger

Type: No Trigger

▼ Time domain graph

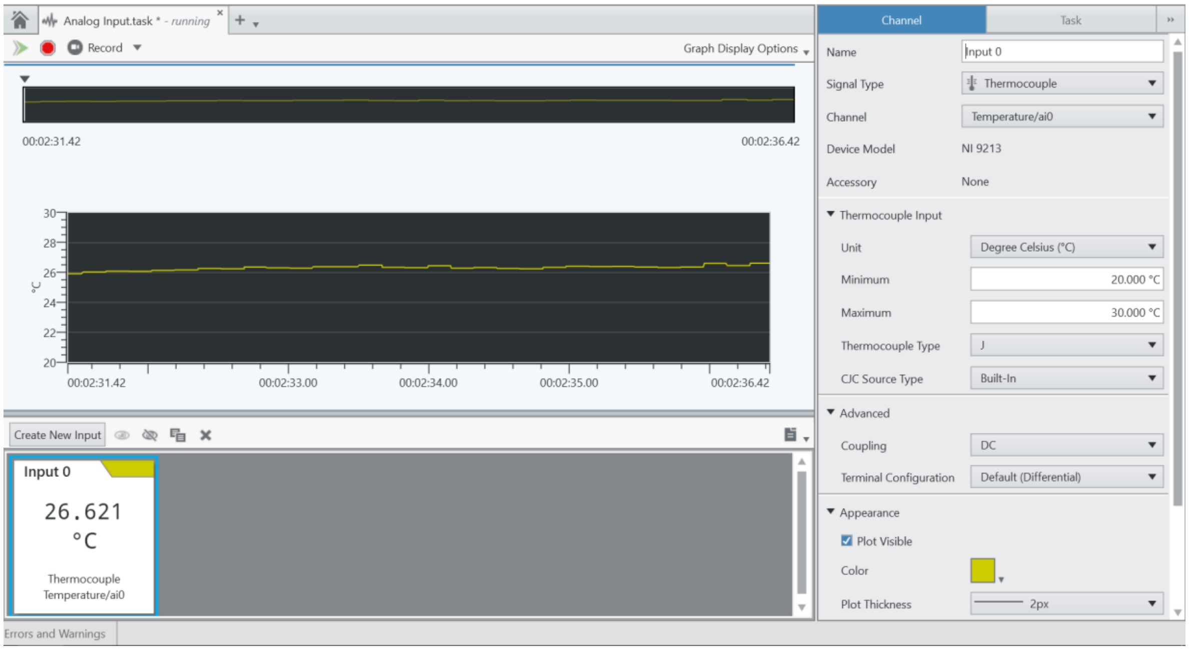
□ Ignore start time

☑ Scale Y axis by channel range

□ Show absolute time

Figure 3-10: Use the Task tab in the Configuration Pane to modify the timing settings that will be applied to all channels in a task.

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



Input 0

26.621 °C

Thermocouple
Temperature/ai0

Channel | Task

Name: Input 0

Signal Type: Thermocouple

Channel: Temperature/ai0

Device Model: NI 9213

Accessory: None

▼ Thermocouple Input

Unit: Degree Celsius (°C)

Minimum: 20.000 °C

Maximum: 30.000 °C

Thermocouple Type: J

CJC Source Type: Built-In

▼ Advanced

Coupling: DC

Terminal Configuration: Default (Differential)

▼ Appearance

☑ Plot Visible

Color: 🟨

Plot Thickness: 2px

Figure 3-11: Verify hardware settings by view thermocouple data in the Measurement Panel.

6. Click **Stop** to stop acquiring data.

The Measurement Panel allows us to acquire data interactively. However, project requirements evolved and we will need to automate acquiring temperature data or including inline analysis. In LabVIEW NXG, we can reuse this hardware configuration when building an automated data acquisition application in a VI.

7. In the Project Viewer, right-click the Exercises folder and select **Add New » VI.**

8. Save the VI as **2 – Basic Measurement.gvi.**

9. Select **Diagram** to open the block diagram where you will define the code that will continuously acquire thermocouple data.

10. Select **Analog Input.task** in the Hardware Tasks folder and drag the file onto the block diagram.

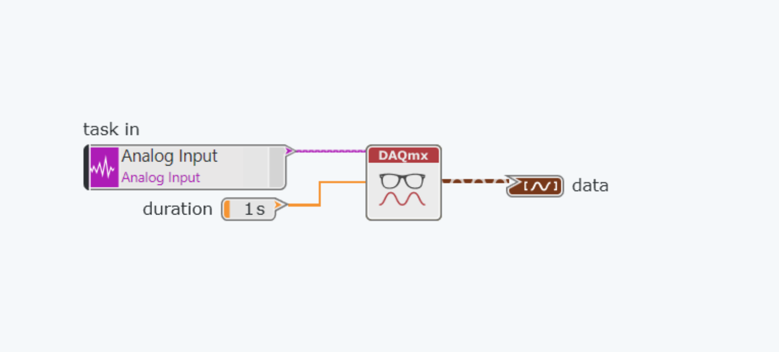


Figure 3-12: By dragging and dropping the hardware task on the block diagram, code will automatically be generated that can read thermocouple data.

11. Select **Panel** to build our user interface on the Front Panel. Notice a notification in the Unplaced Items tray and expand to see the graph that was generated when the *Analog Input.task* was placed onto the Block Diagram.

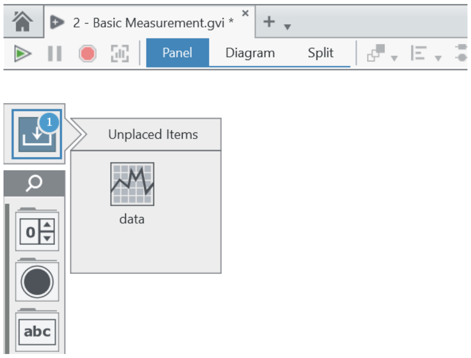


Figure 3-13: Any time a control or indicator is created on the block diagram, it is put into the Unplaced Item tray until you are able to place the item on the Front Panel.

12. Click on the graph and place on the Front Panel. Press the Run button to execute this code.

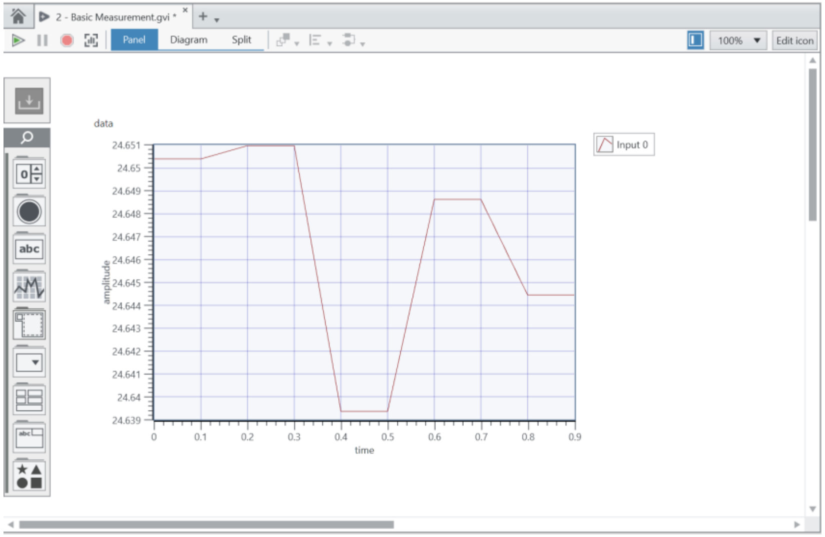


Figure 3-14: The VI will read thermocouple data for one second.

13. Notice we are not continuously acquiring thermocouple data because we are only calling the generated NI-DAQmx Read function once in our VI. To acquire data continuously, we need to place the acquisition code in a while loop.

14. Select the **Program Flow** icon on the Functions Palette on the right-side of the VI. Left-click on the While Loop and click and drag diagonally on the block diagram so the NI-DAQmx code is surrounded.

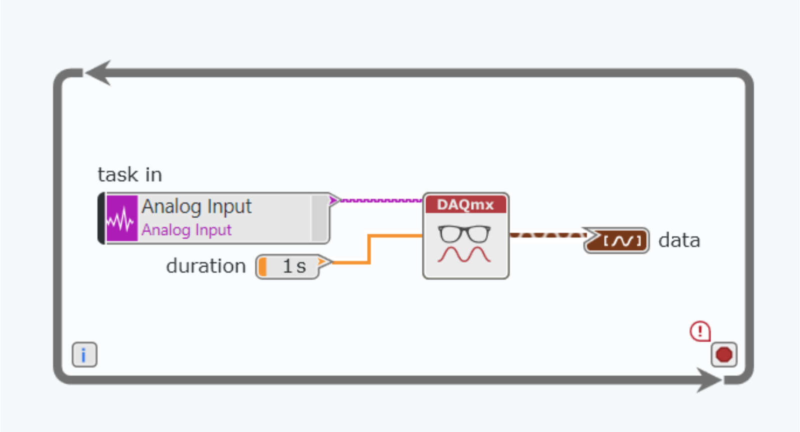


Figure 3-15: Code inside of the While Loop will continuously execute until a condition is met.

15. Press **Ctrl+H** to open Context Help to learn more about while loops. While Loops have two terminals in their bottom right and left corners. 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. LabVIEW NXG 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.

16. Create a Stop button by right-clicking on the conditional terminal. Select to create a control and a Boolean control will be automatically wired to the conditional terminal.

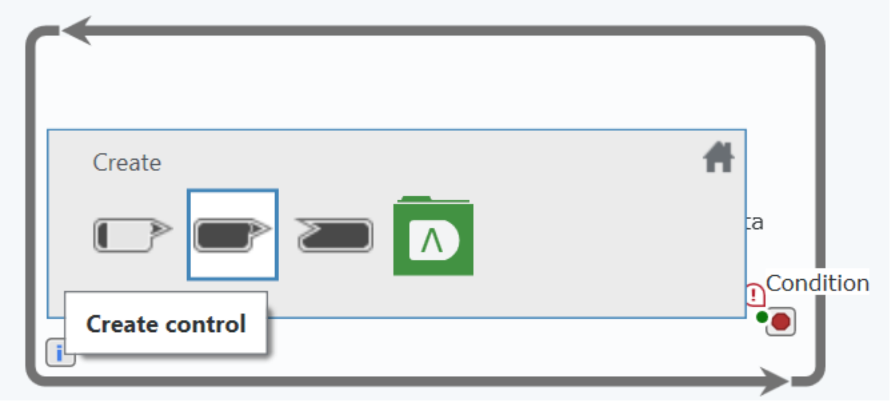


Figure 3-16: Right-click on any terminal to create a control, constant, or indicator.

17. Return to the Front Panel. Notice that we have an item in the Unplaced Item tray. Click on the **Stop** button and place on the Front Panel next to your previously placed graph.

18. Press the Run button. Notice that placing the While Loop has allowed us to continuously acquire thermocouple data. Touch the thermocouple to view the temperature response.

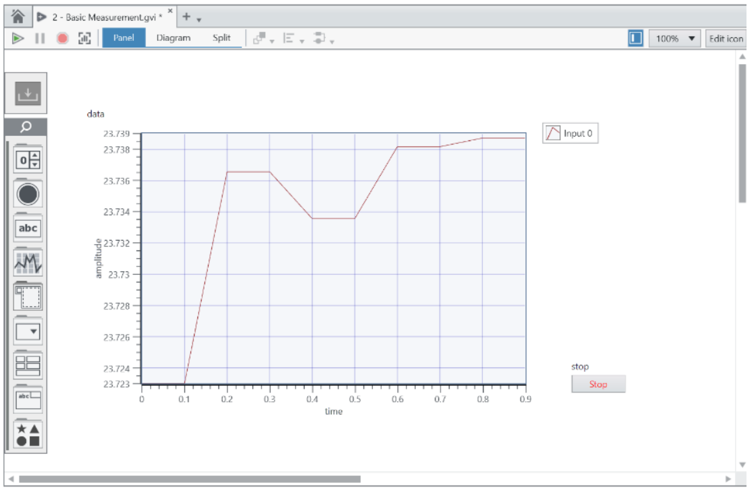


Figure 3-17: After the While Loop is placed, temperature data will continuously be acquired until the stop button is pressed on the Front Panel.

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

Additional Steps

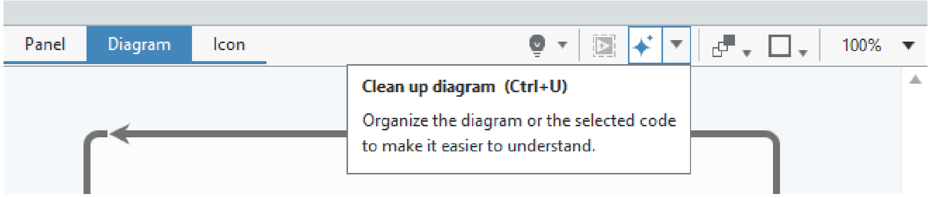
LabVIEW NXG provides several tools that can help you develop your applications. The next few steps will show how to use some of the most important programming assistance tools.

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.

The 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.

20. Press the **Clean up diagram** button found on the menu bar.



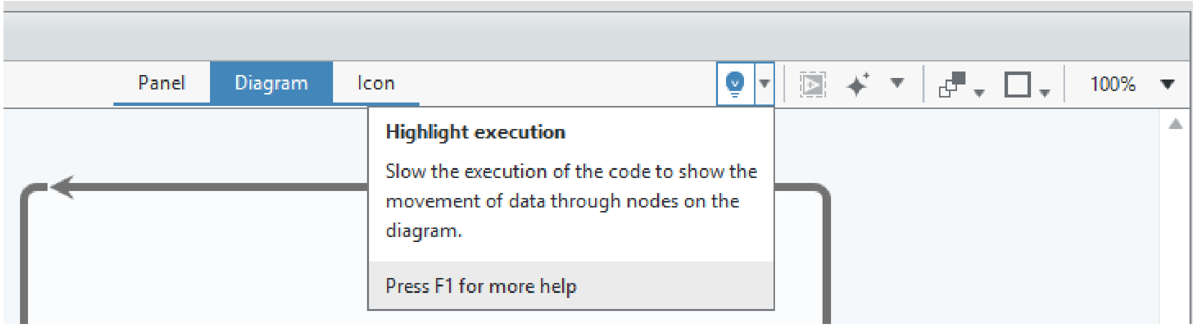
Clean up diagram (Ctrl+U)

Organize the diagram or the selected code to make it easier to understand.

Figure 3-18: Block diagram cleanup automatically organizes your code.

Highlight Execution

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



Highlight execution

Slow the execution of the code to show the movement of data through nodes on the diagram.

Press F1 for more help

Figure 3-19: Highlight Execution slows your code to let you visualize dataflow.

22. 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.

Part C

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

1. From within the *Exercises* folder of the open *Temperature* project, open **3 – Analysis and Output.gvi**. 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. On the block diagram, select the *Analog Input.task* from the **task in** constant wired into the NI-DAQmx Read function. This provides the DAQmx Read function the required hardware information in order to take an analog input measurement.

3. Navigate in the Functions palette and navigate to **Analysis » Signal Processing » Measurement.** Select and place a **Limit Testing** on the block diagram inside the while loop to perform analysis inline.

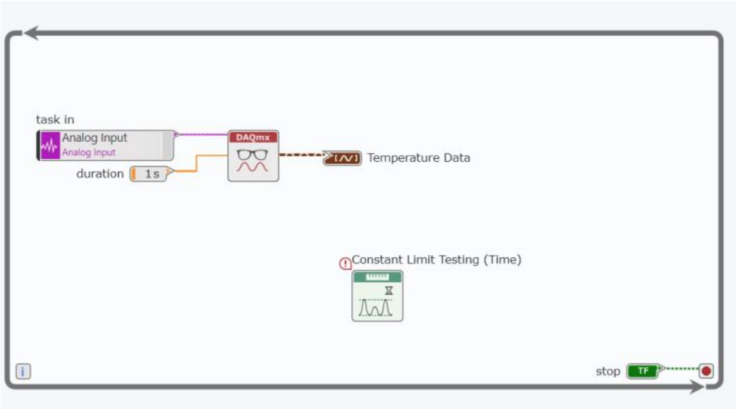
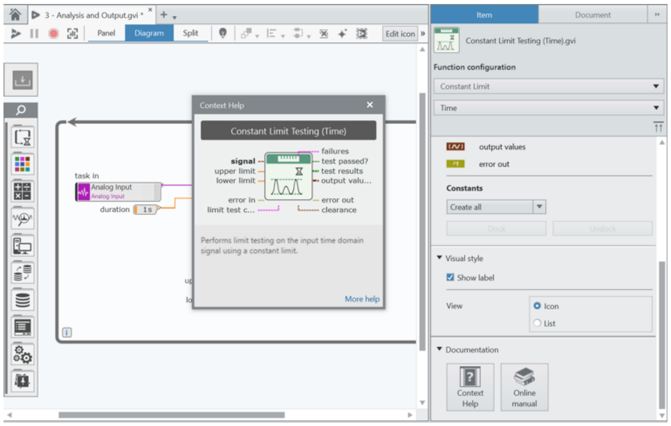


Figure 3-20: Placing the Limit Testing function in the While Loop allows the thermocouple data to be continuously analyzed.

4. Open the Configuration Panel for the Limit Testing. To learn more about this function, scroll to the bottom of the panel and select **Context Help.**



Performs limit testing on the input time domain signal using a constant limit.

Function configuration: Constant Limit; Time

☑ Show label

View: ● Icon ○ List

Figure 3-21: Use Context Help to find more information about front panel and block diagram objects.

5. Wire the output of the NI-DAQmx Read function to the **signal** input of the Limit Testing function.

6. For this application, we are going to let the user define the upper and lower limits of the temperature range. To create controls for the user to define those values, right-click the **upper limit** and **lower limit** terminals of the Limit Test function and select **Create Constant.**

7. Right-click the **test passed?** terminal of the Limit Testing function and select **Create Indicator.** This will alert the user on the front panel if the temperature remained within the defined limits.

8. The block diagram should resemble the code below.

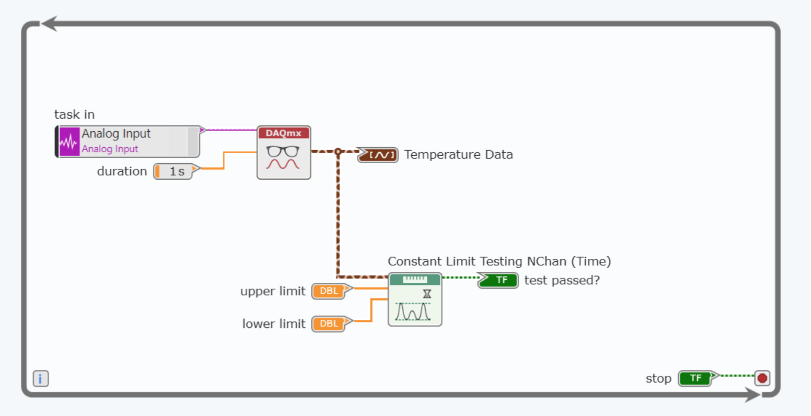


Figure 3-22: Our application continuously acquires thermocouple data and processes the data using a Limit Testing function.

9. Press **Panel** to return to the front panel of the VI. There should be three items in the Unplaced Items tray for you to put on the front panel. *Caption – mention using the control button trick*

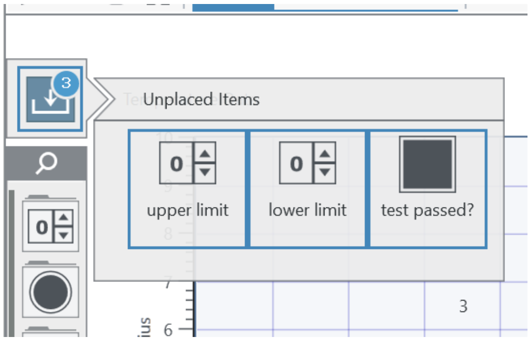


Figure 3-23: To select multiple objects from the Unplaced Items tray, press Control and click at the same time. Then, place the items on the front panel in the same order of clicking.

10. Run the VI.

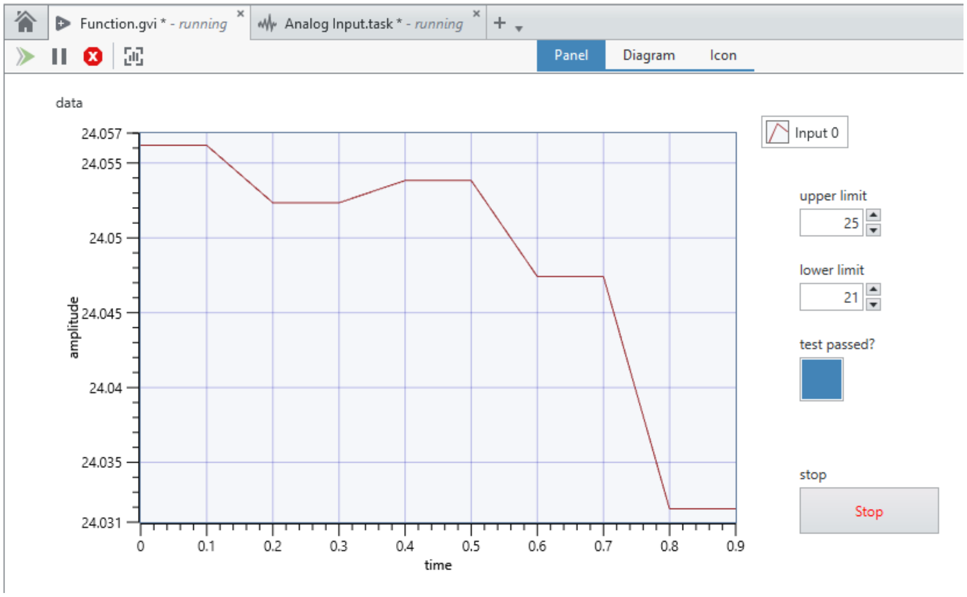


Figure 3-24: The Front Panel will allow the user to define the upper and lower limits of the temperature range.

11. We can modify the appearance of the **test passed?** indicator using the Configuration Pane. Select this indicator and expand the Configuration Pane to the right of your window.

12. Change the color of the LED's **True** or **False** value to the colors of your choice.

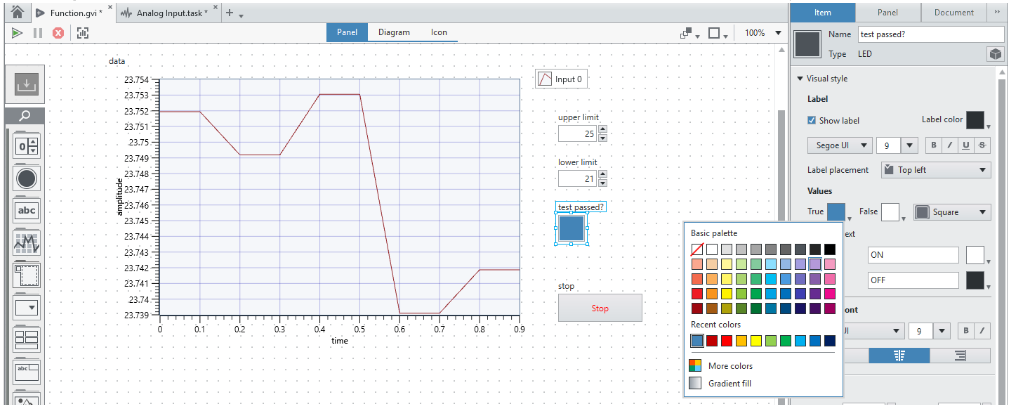
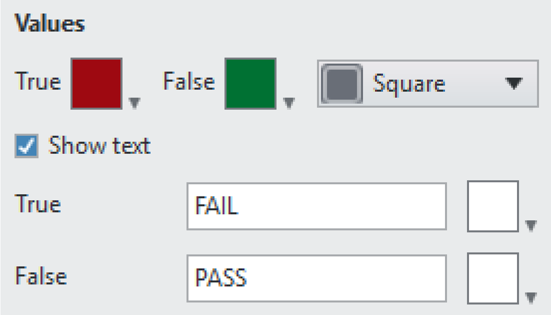


Figure 3-25: The visual style of a front panel object can be modified in the Configuration Pane.

13. Add text to the LED to let the user know if the temperature passed or failed the limit test.



Values

True 🟥 False 🟩 Square

☑ Show text

True FAIL □

False PASS □

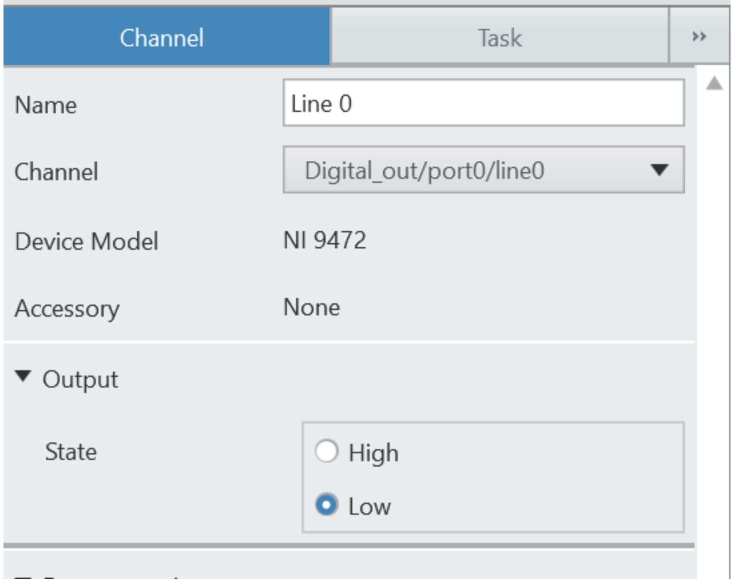
Figure 3-26: Modify the appearance of front panel objects in the Configuration Pane.

14. Run the VI by pressing the **Run** button. Try changing the **upper limit** and **lower limit** values and using your fingers to apply heat to the thermocouple. Note that if you hold the thermocouple until the temperature exceeds the **upper limit** value, the **test passed?** LED turns on. Next, we will update a digital line to turn on a physical LED, depending on this temperature threshold status.

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

16. In the Project Viewer, right-click on the Hardware Tasks folder and select **Add New » Digital Output.** Once opened, save the task as *Digital Output.task*.

17. In the Measurement Panel, use the *Channel* drop-down box to select **Digital_out/port0/line0.**



Name: Line 0

Channel: Digital_out/port0/line0

Device Model: NI 9472

Accessory: None

State: ○ High ● Low

Figure 3-27: A digital line output allows us to generate data to a single digital line.

18. Run the *Digital Output.task.* Toggle *Line 0* between low and high states and notice that the LEDs on the NI CompactDAQ demo box and the NI 9472 module change states in response to your input.

19. Close and save the task. We can use this hardware configuration on our block diagram to output a digital high when the temperature falls out the defined range.

20. Return to the block diagram of *3-Analysis and Output.gvi*.

21. To write to a digital line, we can use the NI-DAQmx Write function. Navigate on the Functions Palette **Hardware Interfaces » NI-DAQmx** and select **NI-DAQmx Write** to place on the block diagram.

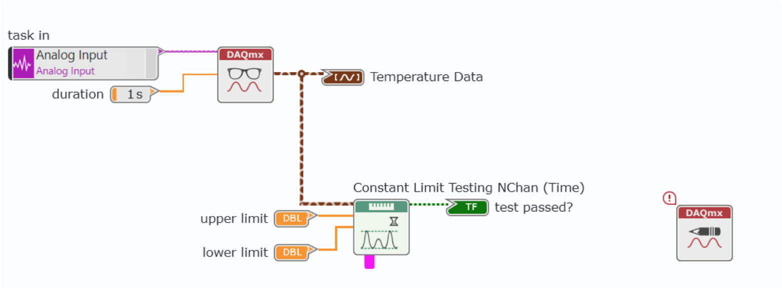
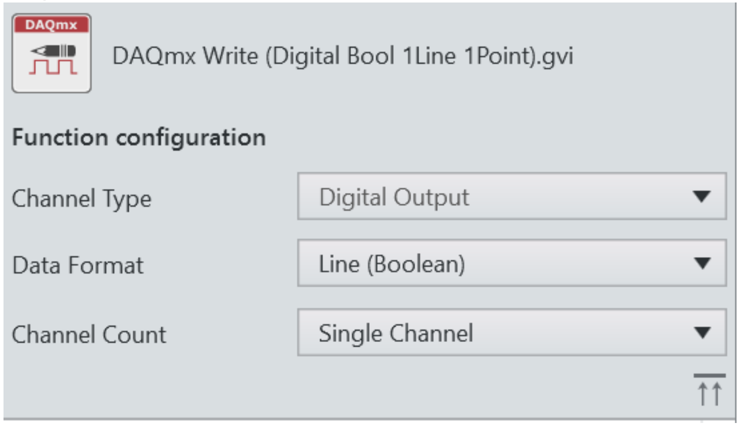


Figure 3-28: When the NI-DAQmx Write function is placed in a While Loop, it will continuously output a digital signal to the specified channel.

22. Use the Configuration Panel to modify the settings for this function to generate digital signal on a single line and channel.



DAQmx Write (Digital Bool 1Line 1Point).gvi

Function configuration

Channel Type: Digital Output

Data Format: Line (Boolean)

Channel Count: Single Channel

Figure 3-29: This function is configured to write to a single digital line.

23. Select **Create constant** in the Configuration Pane for the *task in* terminal. This will create a task constant wired to the NI-DAQmx Write function. From the task constant drop-down box, select **Digital Output.**

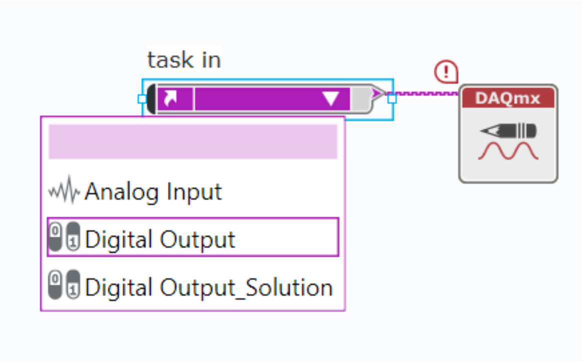


Figure 3-30: All hardware tasks that are connected to the project are available to be used in this VI.

24. Wire the output of the Limit Testing VI to the data terminal of the NI-DAQmx Write function. The NI-DAQmx Write function will change the state on the digital line depending if the temperature is outside of the user-defined limits.

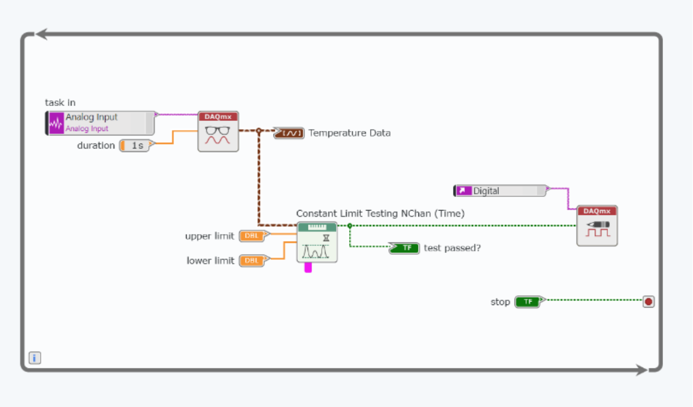


Figure 3-31: The Limit Testing result will be output to the digital module.

25. Press the **Run** button. Notice that the LED bank on the CompactDAQ 9472 module turns on and off to match *test passed?* value on the front panel.

26. Save the VI.

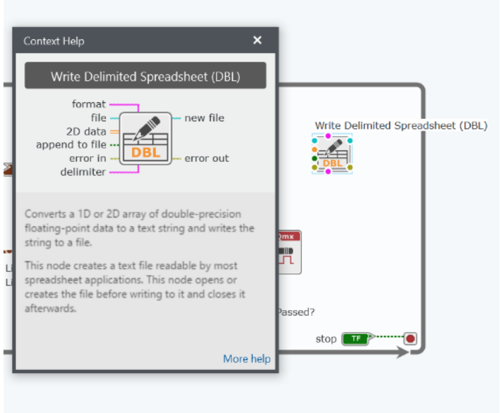
Part D

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.gvi* should still be open from the previous exercise. If not, open it now from the *Exercises* folder in the *Temperature* project.

2. Navigate to the Storage folder on the Functions palette and select **Write Delimited Spreadsheet.** Place the function inside the While Loop on the block diagram.

3. Use Context Help (CTRL + H) to learn more about how this function writes data to a measurement file.



Converts a 1D or 2D array of double-precision floating-point data to a text string and writes the string to a file.

This node creates a text file readable by most spreadsheet applications. This node opens or creates the file before writing to it and closes it afterwards.

Figure 3-32: The Write Delimited Spreadsheet VI is the simplest method of writing data to a file.

4. Right-click the **file** terminal of the *Write Delimited Spreadsheet* function to create a constant.

5. Find the file path of *Measurement Data.txt* that is located in the Measurement Data folder in the Project Viewer by right-clicking on *measurement data.txt* and selecting **Locate item in Windows Explorer.** Copy directory into the file path *constant* created in the previous step.

6. Dock the constant to the Write Delimited Spreadsheet function by right-clicking and selecting **dock to node**.

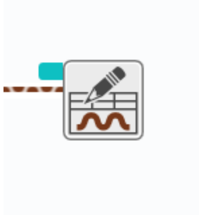


Figure 3-33: Docking a constant increases readability of your code by eliminating wire clutter.

7. Wire the data output from NI-DAQmx Read function to the data input of the Write Delimited Spreadsheet function.

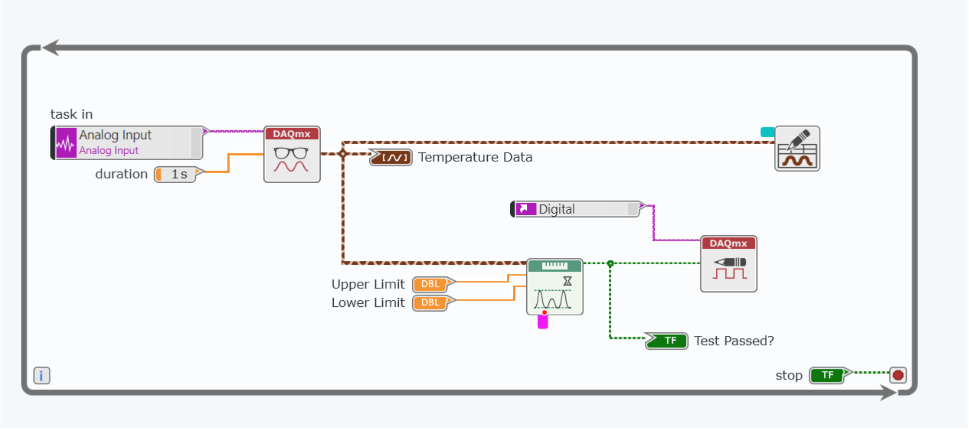
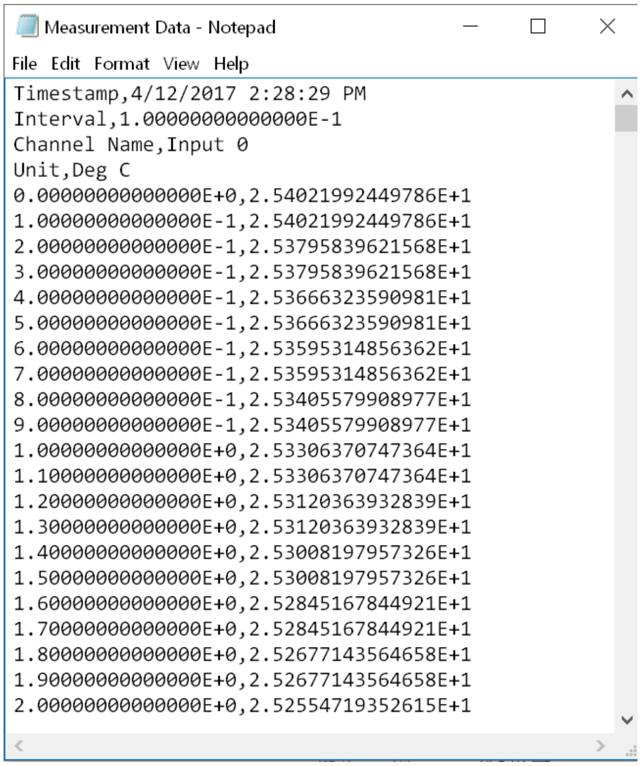


Figure 3-34: This completed application acquires temperature data, outputs a digital signal if the temperature exceeds a threshold, and saves all of the data to file.

8. Return to the Front Panel and run the VI.

9. Right-click on the *Measurement Data.txt* file in the Project Viewer to open the file.



```
Timestamp,4/12/2017 2:28:29 PM
Interval,1.00000000000000E-1
Channel Name,Input 0
Unit,Deg C
0.00000000000000E+0,2.54021992449786E+1
1.00000000000000E-1,2.54021992449786E+1
2.00000000000000E-1,2.53795839621568E+1
3.00000000000000E-1,2.53795839621568E+1
4.00000000000000E-1,2.53666323590981E+1
5.00000000000000E-1,2.53666323590981E+1
6.00000000000000E-1,2.53595314856362E+1
7.00000000000000E-1,2.53595314856362E+1
8.00000000000000E-1,2.53405579908977E+1
9.00000000000000E-1,2.53405579908977E+1
1.00000000000000E+0,2.53306370747364E+1
1.10000000000000E+0,2.53306370747364E+1
1.20000000000000E+0,2.53120363932839E+1
1.30000000000000E+0,2.53120363932839E+1
1.40000000000000E+0,2.53008197957326E+1
1.50000000000000E+0,2.53008197957326E+1
1.60000000000000E+0,2.52845167844921E+1
1.70000000000000E+0,2.52845167844921E+1
1.80000000000000E+0,2.52677143564658E+1
1.90000000000000E+0,2.52677143564658E+1
2.00000000000000E+0,2.52554719352615E+1
```

Figure 3-35: The resulting temperature data file that was saved from the completed LabVIEW NXG 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>

Chapter 4 Combining Measurements from Light and Temperature

Goals

- In this exercise, you will acquire light and temperature signals in the same hardware task using a Measurement Panel.
- In the second part of this exercise, you will modify code to scale the data and measure the ambient light in the Measurements Demo Box

Part A Open and Run Our Starting Point

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 **Windows » LabVIEW NXG**.
2. Open the Temperature and Light project.
    a. In the LabVIEW NXG welcome screen, select **Launch a Project**.
    b. Select **Browse** to *C:\Seminars\cDAQ + LabVIEW NXG HO\Exercises\4 – Temperature and Light.*
    c. Select and open **Temperature and Light.lvproj**.
3. Open and run the starting point application.
    a. In the Navigation Pane, double-click to select **Exercises » 1 – Write to File (Solution).gvi**.

b. Run the application by clicking the **Run** arrow (▷) on the front panel.

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 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 front panel.

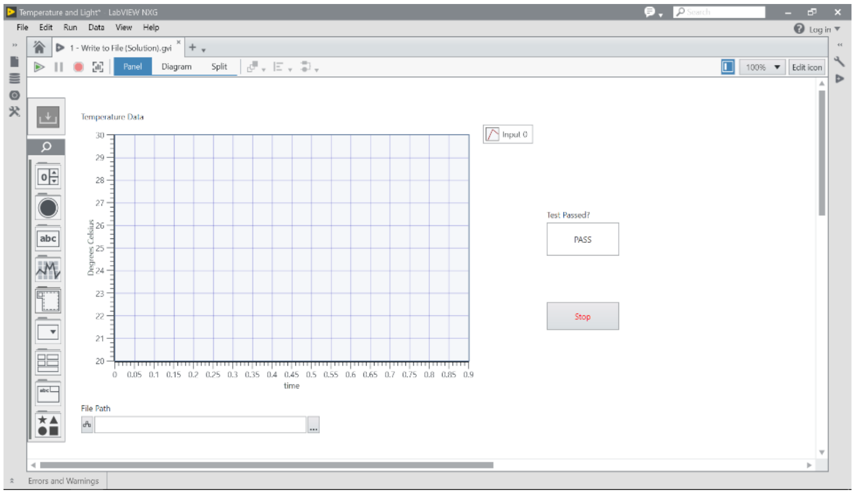


Figure 4-1: Use the Run arrow to run the application

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

   This VI was created by dragging and dropping the *Temperature.task* onto the block diagram. This method allows you to reuse your existing hardware configurations as a starting point when automating a measurement task. For further customization and control over how you acquire your data, use the NI-DAQmx API. This lower level programming method opens up the capabilities of the device.

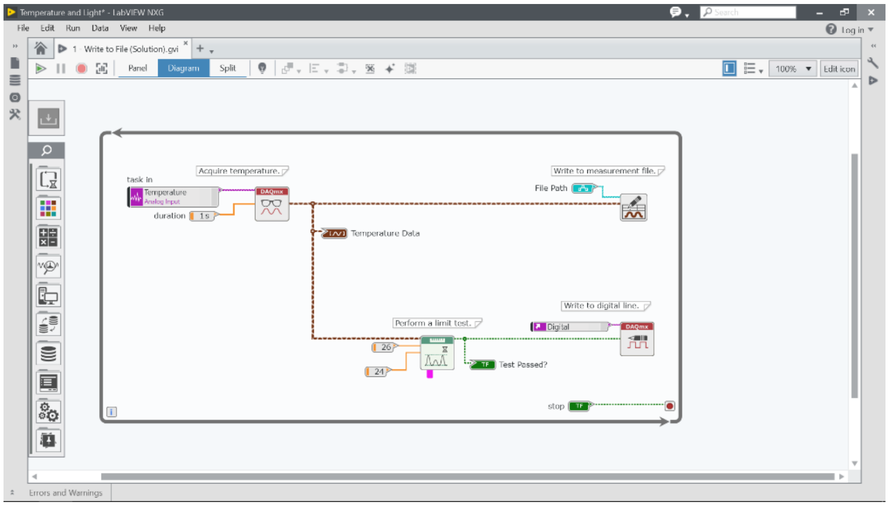


Figure 4-2: Dragging and dropping the hardware configuration is a quick method to begin to automate your data acquisition application.

5. **Close** the VI.

Part B Use the Low-Level NI-DAQmx API

Reusing the measurement task is a quick way to begin automating your data acquisition; however, to have more control over your task use the NI-DAQmx VI. This exercise will create our measurement task programmatically and accomplish the equivalent measurements using the more flexible lower-level NI-DAQmx VIs.

1. In the LabVIEW Project, double-click to select **Exercises » 2 – Use the NI-DAQmx VIs.gvi**.
2. Examine the lower-level NI-DAQmx code.

    a. Select **Diagram** or press **<Ctrl+E>** to examine the block diagram.

    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,

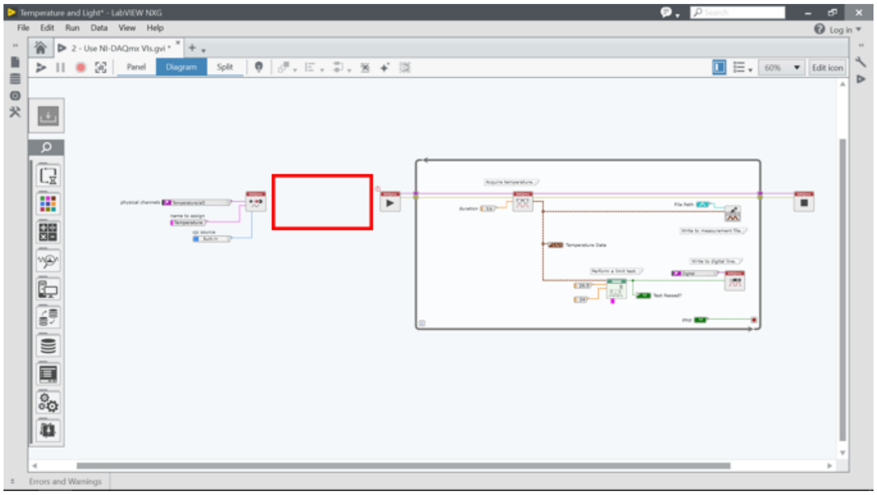
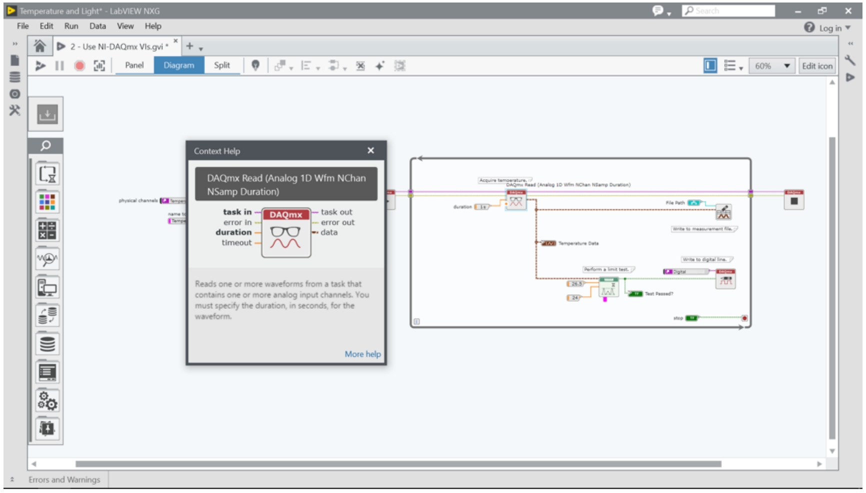


Figure 4-3: The lower-level NI-DAQmx API follows a logical pattern.

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



Reads one or more waveforms from a task that contains one or more analog input channels. You must specify the duration, in seconds, for the waveform.

Figure 4-4: 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.

e. Navigate to **Hardware Interfaces » NI-DAQmx** and select **DAQmx Timing.gvi**.

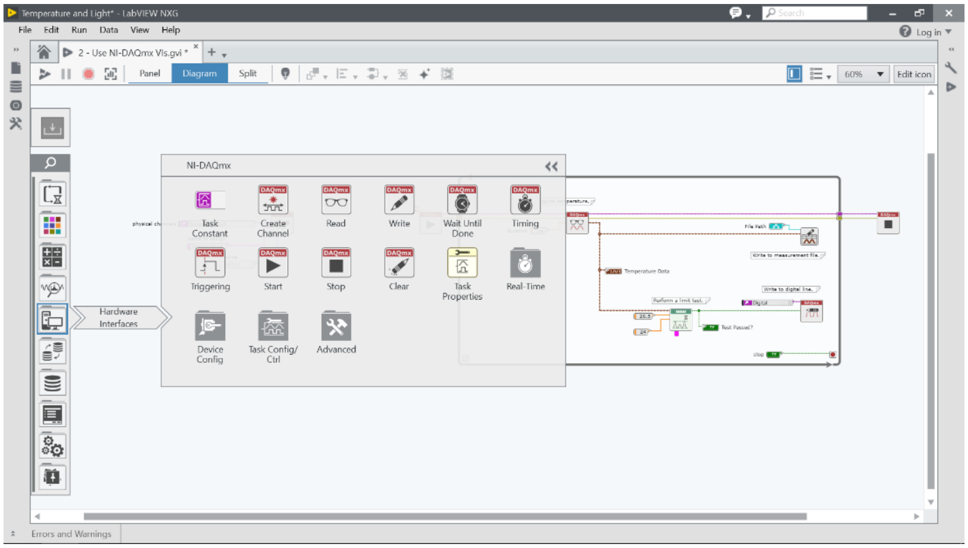


Figure 4-5: The DAQmx palette houses all the data acquisition functions built available with the NI-DAQmx driver.

f. Left-click to place the DAQmx Timing.gvi between the *DAQmx Create Channel VI* and the *DAQmx Start Task VI*.

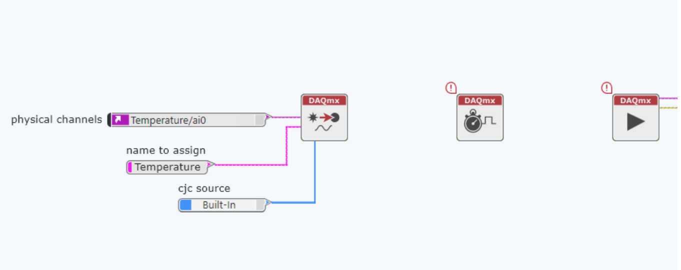
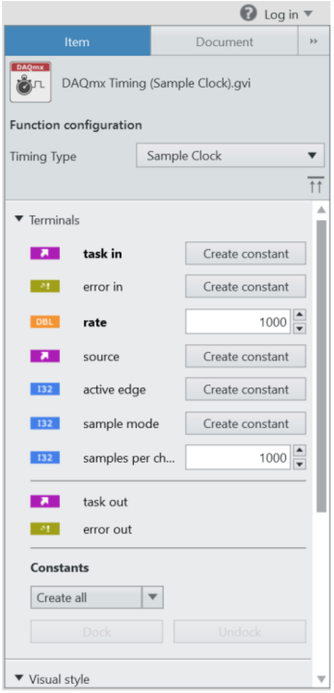


Figure 4-6: The NI-DAQmx Timing VI configures the timing for hardware-timed data acquisition operations.

g. Use the Configuration Pane to the right to modify the DAQmx Timing VI function configuration. Select **Sample Clock** as the timing type for this VI.


Log in

Item | Document

DAQmx Timing (Sample Clock).gvi

Function configuration

Timing Type: Sample Clock

Terminals

task in	Create constant
error in	Create constant
rate	1000
source	Create constant
active edge	Create constant
sample mode	Create constant
samples per ch...	1000
task out	
error out	

Constants

Create all

Dock | Undock

Visual style

Figure 4-7: Use the Configuration Pane to define the inputs for the NI-DAQmx Timing VI.

h. Select **Create Constant** for the *sample mode* in the configuration pane. This will place a constant on the block diagram wired into the *sample mode* input. Click the dropdown menu to change the value to **Continuous Samples.**

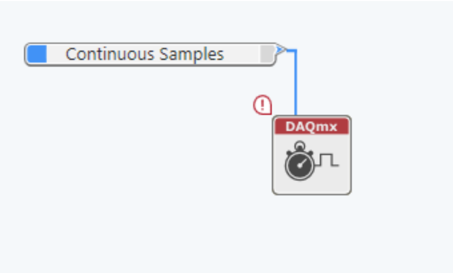


Figure 4-8: After defining the sample mode in the Configuration Pane, a constant is automatically connected to the Timing VI.

i. Mouse over the left side of the DAQmx Timing function to see the input terminals. Right-click on the *Rate* input and select **Create » Constant** from the right-click context menu. Leave the default rate value of 1000 Hz.

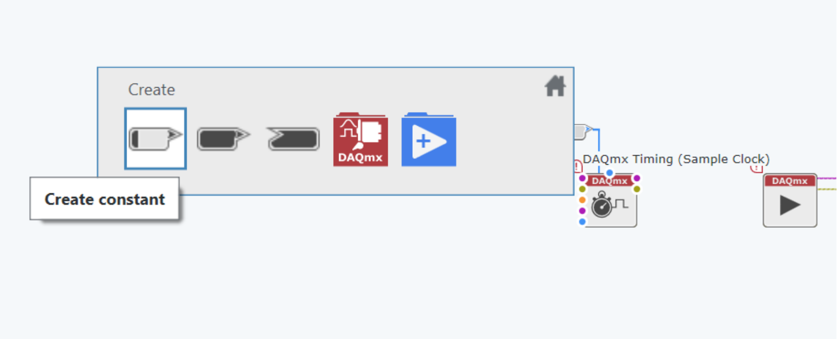


Figure 4-9: By right-clicking on a VI's terminal, you have the option to have the input be a constant or a control that is defined by the user on the front panel.

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

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

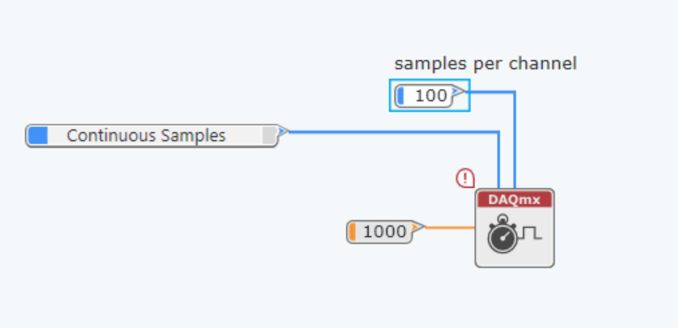


Figure 4-10: The samples per channel equals the number of samples read when the function executes.

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

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

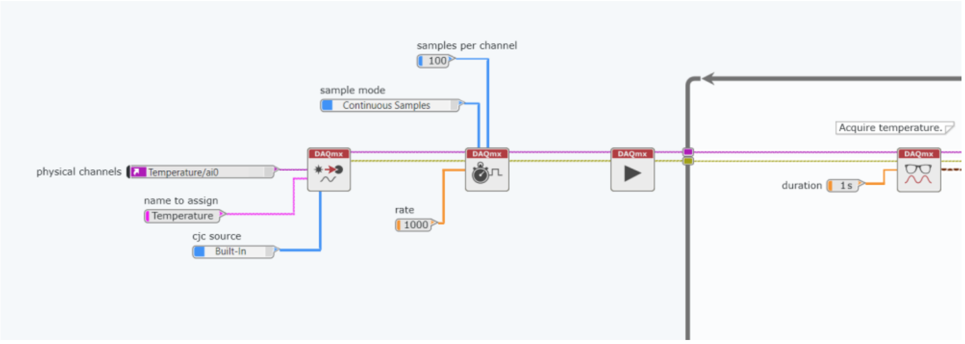


Figure 4-11: Complete the configuration of the DAQmx Timing VI as shown.

4. Run the VI.

n. Using the **Run** arrow (▶), run the VI and ensure that it behaves with the lower-level NI-DAQmx VIs exactly as it did when we reused the hardware configuration on the Measurement Panel.

o. Stop the VI by selecting the **Stop button** on the 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

The lower-level NI-DAQmx VIs offer the greatest flexibility in programming NI LabVIEW NXG 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 dragging and dropping a measurement task on the block diagram (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.gvi**.

    b. Right-click on **2 – NI-DAQmx VIs.gvi** and select **Duplicate**

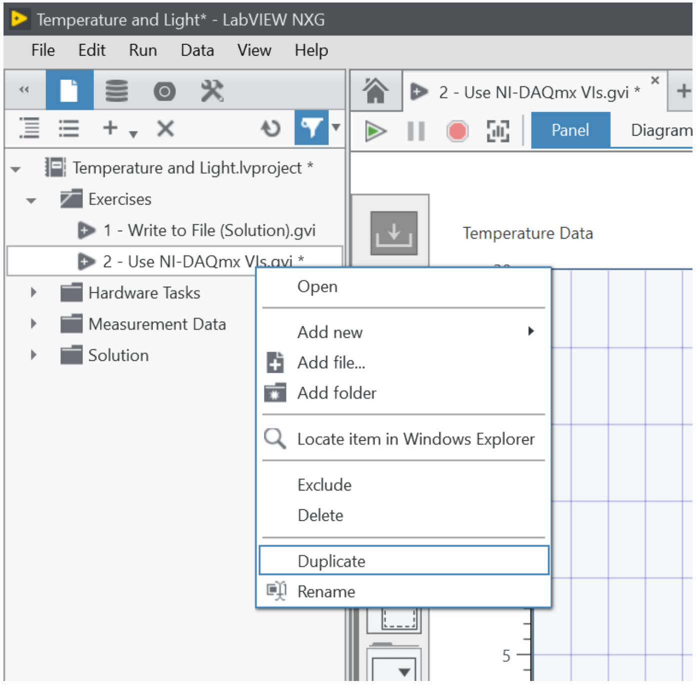
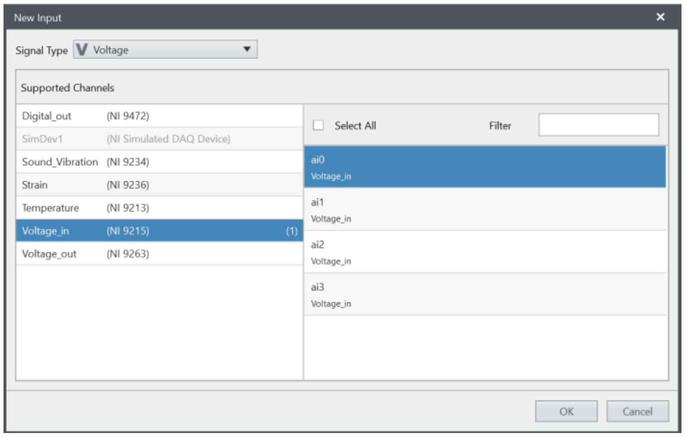

Temperature and Light* - LabVIEW NXG
File
Edit
Run
Data
View
Help
2 - Use NI-DAQmx VIs.gvi *
Panel
Diagram
Temperature and Light.lvproject *
Exercises
1 - Write to File (Solution).gvi
2 - Use NI-DAQmx VIs.gvi *
Hardware Tasks
Measurement Data
Solution
Temperature Data
Open
Add new
Add file...
Add folder
Locate item in Windows Explorer
Exclude
Delete
Duplicate
Rename
5

Figure 4-12: Duplicating the VI gives you code as a starting point when completing this exercise.

    c. Rename the duplicated VI as **3 – Use an NI-DAQmx Task.gvi**.

2. Right-click on *Temperature.task* in the Navigation Pane and select **Duplicate.** Rename this task *Temperature and Light*

a. Select **Create New Input.** This will allow you to create another analog input measurement in this task.



Signal Type: V Voltage

Supported Channels

Digital_out	(NI 9472)	
SimDev1	(NI Simulated DAQ Device)	
Sound_Vibration	(NI 9234)	
Strain	(NI 9236)	
Temperature	(NI 9213)	
Voltage_in	(NI 9215)	(1)
Voltage_out	(NI 9263)	

□ Select All Filter

- ai0 Voltage_in
- ai1 Voltage_in
- ai2 Voltage_in
- ai3 Voltage_in

Figure 4-13: Include new inputs to this task by selecting additional hardware channels.

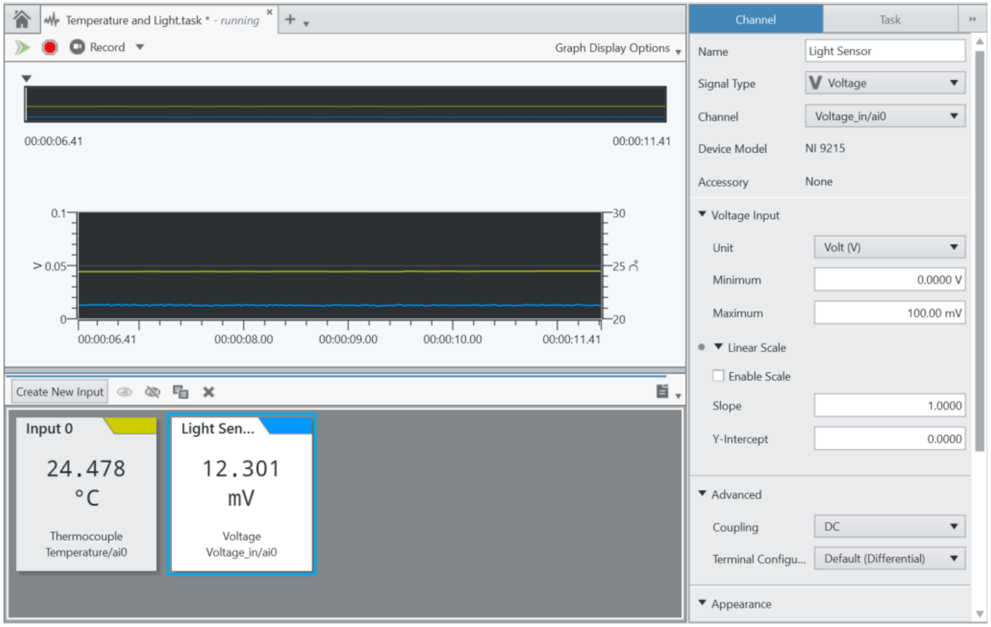
b. Select **Voltage** as the new Signal Type.

c. Choose **Voltage_in** and select the **ai0** channel.

3. Configure the Light Sensor channel in the task to acquire data from the light sensor attached to the box that holds the NI CompactDAQ chassis.

a. In the configuration pane, rename the channel *Light Sensor.*

b. Because the voltage readings are small, change the **minimum value** to 0 mV and the **maximum value** to 100 mV. Run the task to see temperature and light data on the same graph.



Input 0: 24.478 °C, Thermocouple, Temperature/ai0

Light Sen...: 12.301 mV, Voltage, Voltage_in/ai0

Name	Light Sensor
Signal Type	V Voltage
Channel	Voltage_in/ai0
Device Model	NI 9215
Accessory	None

▾ Voltage Input

Unit	Volt (V)
Minimum	0.0000 V
Maximum	100.00 mV

▾ Linear Scale

□ Enable Scale

Slope	1.0000
Y-Intercept	0.0000

▾ Advanced

Coupling	DC
Terminal Configu...	Default (Differential)

▾ Appearance

Figure 4-14: In the Measurement Panel, you can view both temperature and light signals on one graph.

4. Use the Temperature and Light task on the Block Diagram

   a. In the **3 – Use a NI-DAQmx Task.gvi,** click and drag to highlight all VIs and input parameters to the left of the *DAQmx Start Task.gvi.*
   b. Using your keyboard, press **Delete** to delete the code to the left of the DAQmx Start Task.gvi.

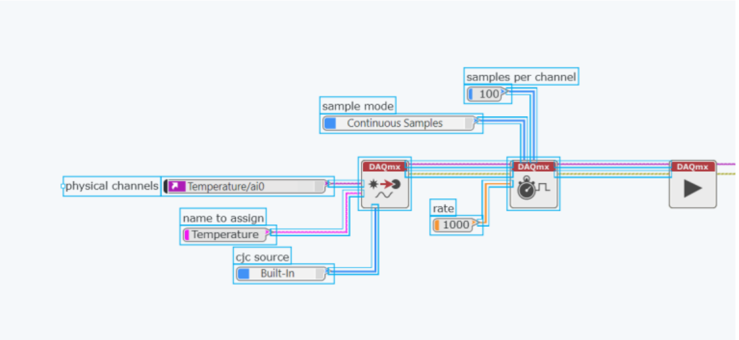
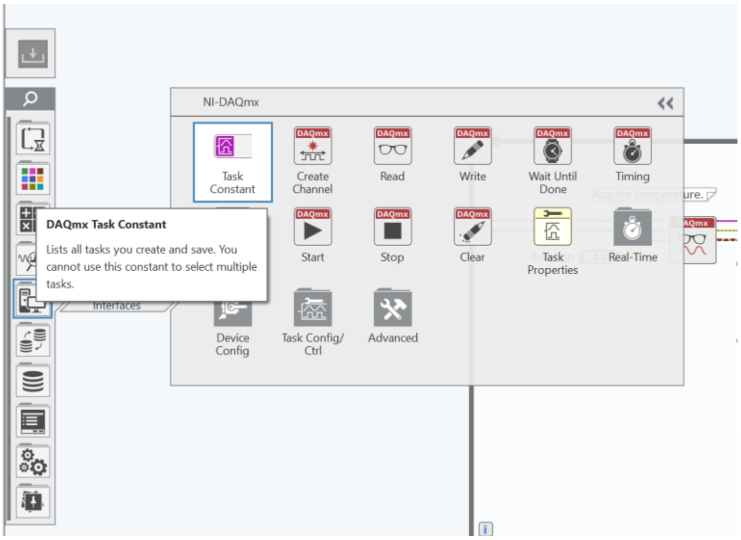


Figure 4-15: This section of code will be replaced using the configuration from the Temperature and Light.task.

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

   d. Navigate on the Functions Palette to Hardware **Interfaces » NI-DAQmx** and select **DAQmx Task Name Constant**.

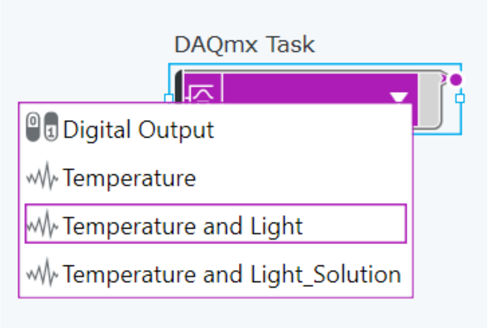


DAQmx Task Constant

Lists all tasks you create and save. You cannot use this constant to select multiple tasks.

Figure 4-16: Select the NI-DAQmx Task Name Constant from the DAQmx functions palette.

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


DAQmx Task

Digital Output

Temperature

Temperature and Light

Temperature and Light_Solution

Figure 4-17: All the hardware tasks saved in the project are available to use.

f. Using the drop-down, select the **Temperature and Light** task that you configured in the Measurement Panel.

5. Set-up a Custom NI-DAQmx Scale for the Light Sensor.

In this section, we will scale our solar cell data to make it easier to view alongside our temperature data. Recall that our task includes a thermocouple and a solar cell – we will be applying this scaling only to the solar cell, by using a **DAQmx Task Property Node**.

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 degree Celsius (°C) scale between 0 and 100.

Applying our custom scale to the solar cell measurements will map the solar values into the same range of values being represented by the thermocouple channel at approximately room temperature. We will set the lower bound (no light) at 20 and our upper bound at 40.

a. Navigate on the Functions Palette to **Hardware Interfaces » NI-DAQmx » Advanced » Scale Setup** and select **Create Scale**.

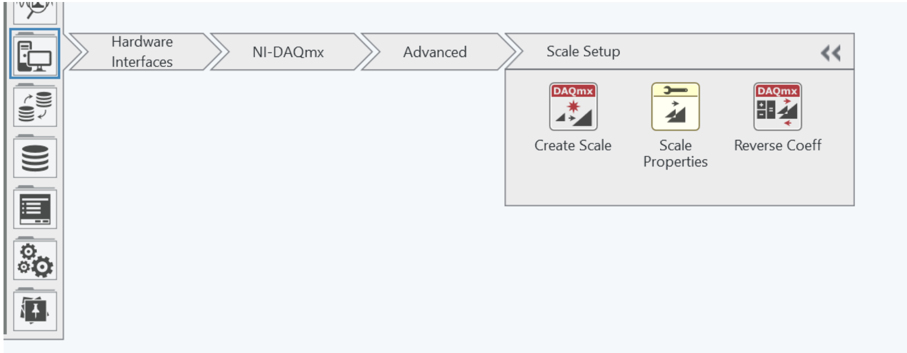


Figure 4-18: Select the NI-DAQmx Create Scale VI in the NI-DAQmx functions palette.

b. Left-click on the DAQmx Create Scale to drop the VI under the Task Constant.

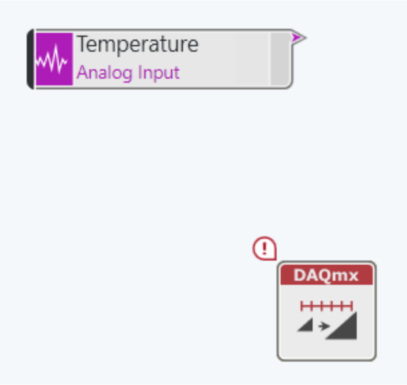
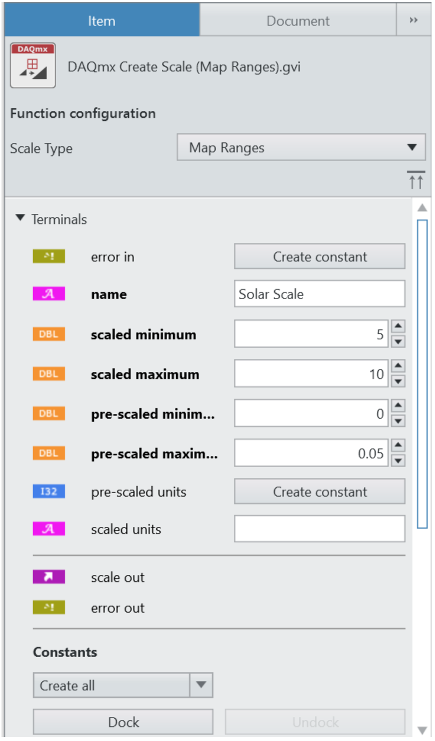


Figure 4-19: Use the DAQmx Create Scale function to define the parameters of the scale that will be applied to the signal acquired from the Light Sensor channel.

c. In the Configuration Pane, select the scale type to **Map Ranges.**

d. Under Terminals, assign the following values for each parameter. Notice constants will be created for each of the inputs for the Create Scale VI.



DAQmx Create Scale (Map Ranges).gvi

Function configuration

Scale Type: Map Ranges

Terminals

	error in	Create constant
	name	Solar Scale
DBL	**scaled minimum**	5
DBL	**scaled maximum**	10
DBL	**pre-scaled minim...**	0
DBL	**pre-scaled maxim...**	0.05
I32	pre-scaled units	Create constant
	scaled units	
	scale out	
	error out	

Constants

Create all

Dock
Undock

Figure 4-20: Match the settings displayed in the Configuration Pane for the Create Scale VI.

e. To clean-up the Block Diagram, select **Dock** to dock your constants and limit the number of wires coming out of the Create Scale VI.

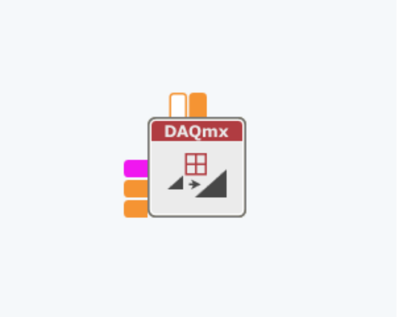


Figure 4-21: Docking constants cleans-up the block diagram. To view the value of each constant, hover over each docked constant.

f. Navigate on the Functions Palette to **Hardware Interfaces » NI-DAQmx** and select **Task Properties**. Left-click to place the NI-DAQmx Task Property node on the Block Diagram.

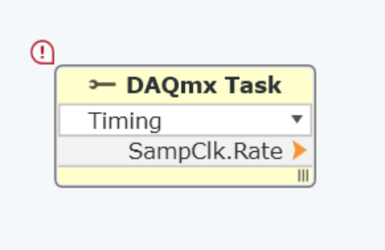


Figure 4-22: A property node is used to modify properties of the DAQmx task and setting to write will allow you to apply the custom scale.

g. In the first drop-down menu of the NI-DAQmx Task Property node, select **Channel.** In the second drop-down menu, select **Wire on Diagram.**

h. Next, expand the DAQmx Task properties node by three items.

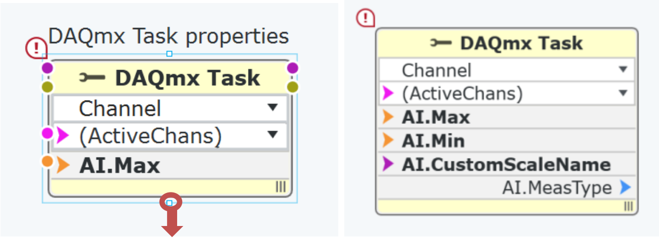


Figure 4-23: You can write to or read from multiple properties in one node.

i. Click the last drop-down menu (**AI.MeasType**) and navigate to **Analog Input > Voltage > Units.** Right-click on the task and select **Set All to Write.** Your property node should look like the one below:

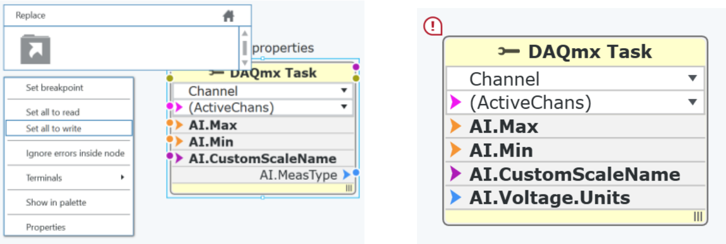


Figure 4-24: Setting all to write will enable you to change the properties of the Temperature and Light task.

j. Click on the DAQmx Task property node. In the configuration pane, set **AI.Max** to *40* and **AI.Min** to *20*. Click on **ActiveChans** and type *Light Sensor.* This is the channel in the *Temperature and Light.task* we will apply the scale to.

k. Right click on the input terminal for **AI.Voltage.Units** and create a constant ( ). This creates a dropdown menu called an Enum. Click on the dropdown menu and select *From Custom Scale*.

l. From the DAQmx Create Scale VI, wire the output of *scale out* to the **AI.Custom.ScaleName** input of the DAQmx Task Property node and wire the output of *error out* to the **Error in** input of the DAQmx Task Property node.

m. Wire the task output of the DAQmx Task Property node to the task input of the DAQmx Start Task VI and wire the error out from the DAQmx Task Property node to the error in.

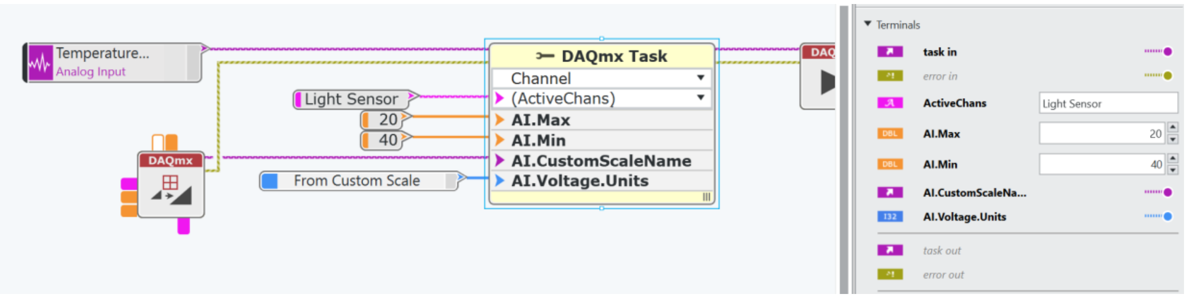


Figure 4-25: The completed code to apply the custom scale to the light sensor channel.

6. Run the VI.

a. Using the **Run** arrow ( ), run the VI. Note that you can see both thermocouple and light sensor data on one graph.

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.

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

7. Close the VI.

8. If time remains, continue to modify your VI and expand its functionality. Try customizing your graph axes and data sets. Try creating a new **Test Passed?** LED to respond to your solar cell measurements.

Chapter 5 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 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

Acquiring strain data is one of the most common tasks in any physical 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.

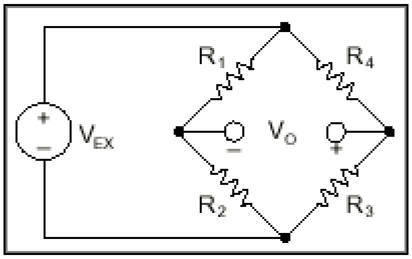
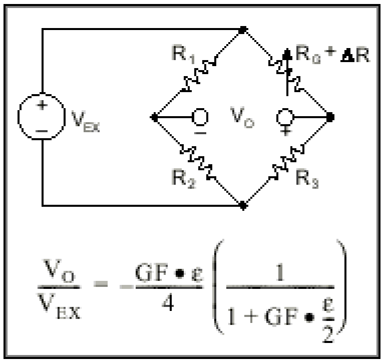


Figure 5-1 A Wheatstone Bridge enables precise measurement of small resistance changes.

In a Wheatstone Bridge, four resistors are used in the configuration seen in Figure 5-1 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 your hardware configuration for this seminar, 350 Ohms. As the strain gage flexes, the resistance changes and, consequently, V_o. Knowing this, we can derive the proportional change, arriving at the following equation:



$$\frac{V_O}{V_{EX}} = -\frac{GF \bullet \varepsilon}{4}\left(\frac{1}{1 + GF \bullet \frac{\varepsilon}{2}}\right)$$

Figure 5-2 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 that you can acquire basic data from the channel connected to the strain gage.

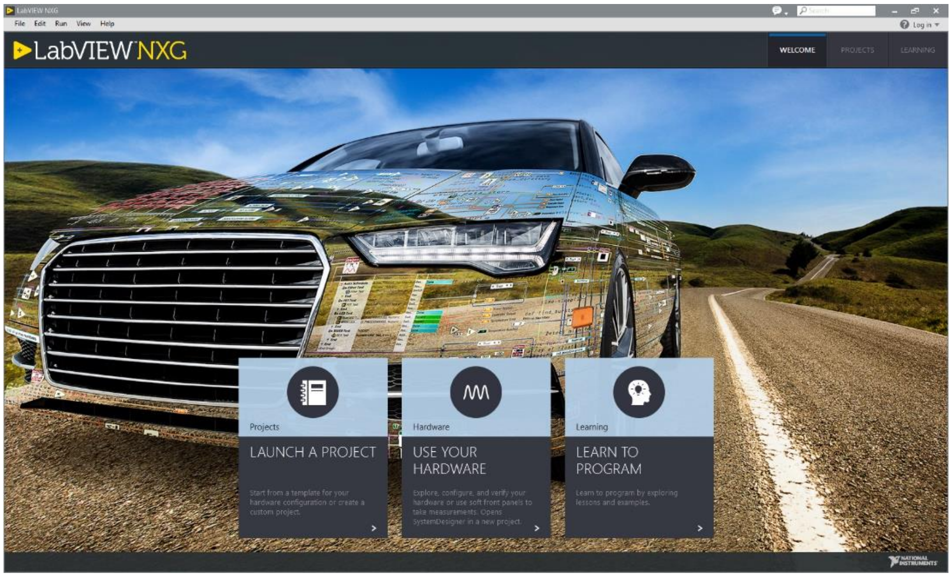
Note: Steps 1-14 below test the function of your Strain Gauge. If you have already configured and tested your hardware, skip to step 15.

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 securely connected to the spring terminals.

3. If it isn't already open, launch LabVIEW NXG by selecting **Windows » LabVIEW NXG**

4. Return to the home 🏠 screen if needed, and click on **Use Your Hardware**.



LAUNCH A PROJECT

Start from a template for your hardware configuration or create a custom project.

USE YOUR HARDWARE

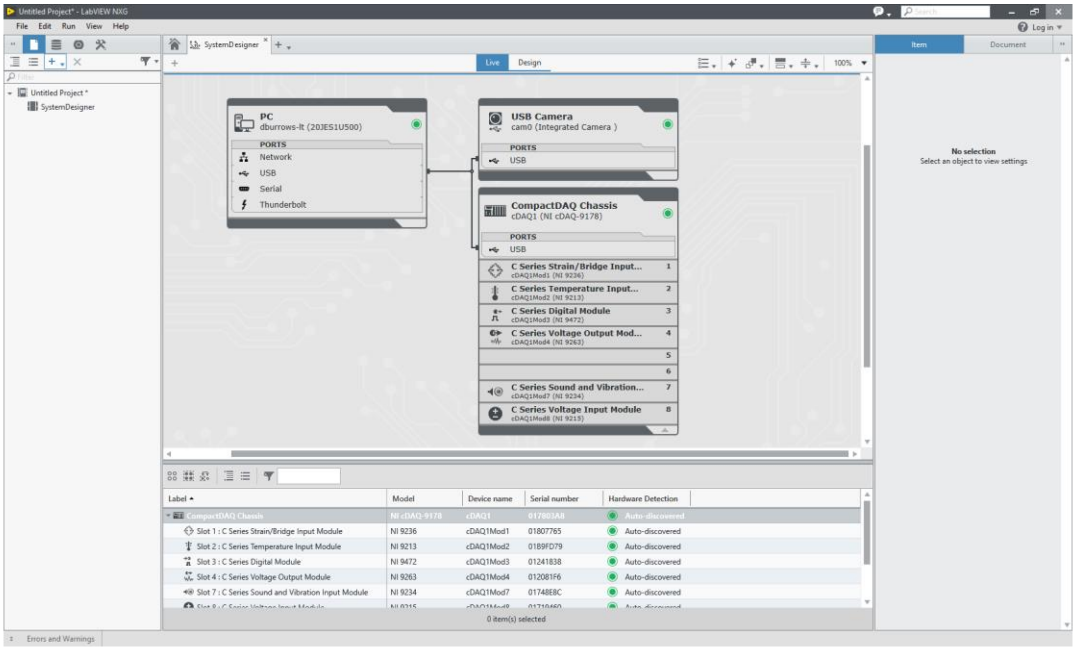
Explore, configure, and verify your hardware or use soft front panels to take measurements. Opens SystemDesigner in a new project.

LEARN TO PROGRAM

Learn to program by exploring lessons and examples.

Figure 5-3: Click Use Your Hardware to configure a simple measurement for checking signal connectivity.

5. In the SystemDesigner, find your CompactDAQ Chassis.

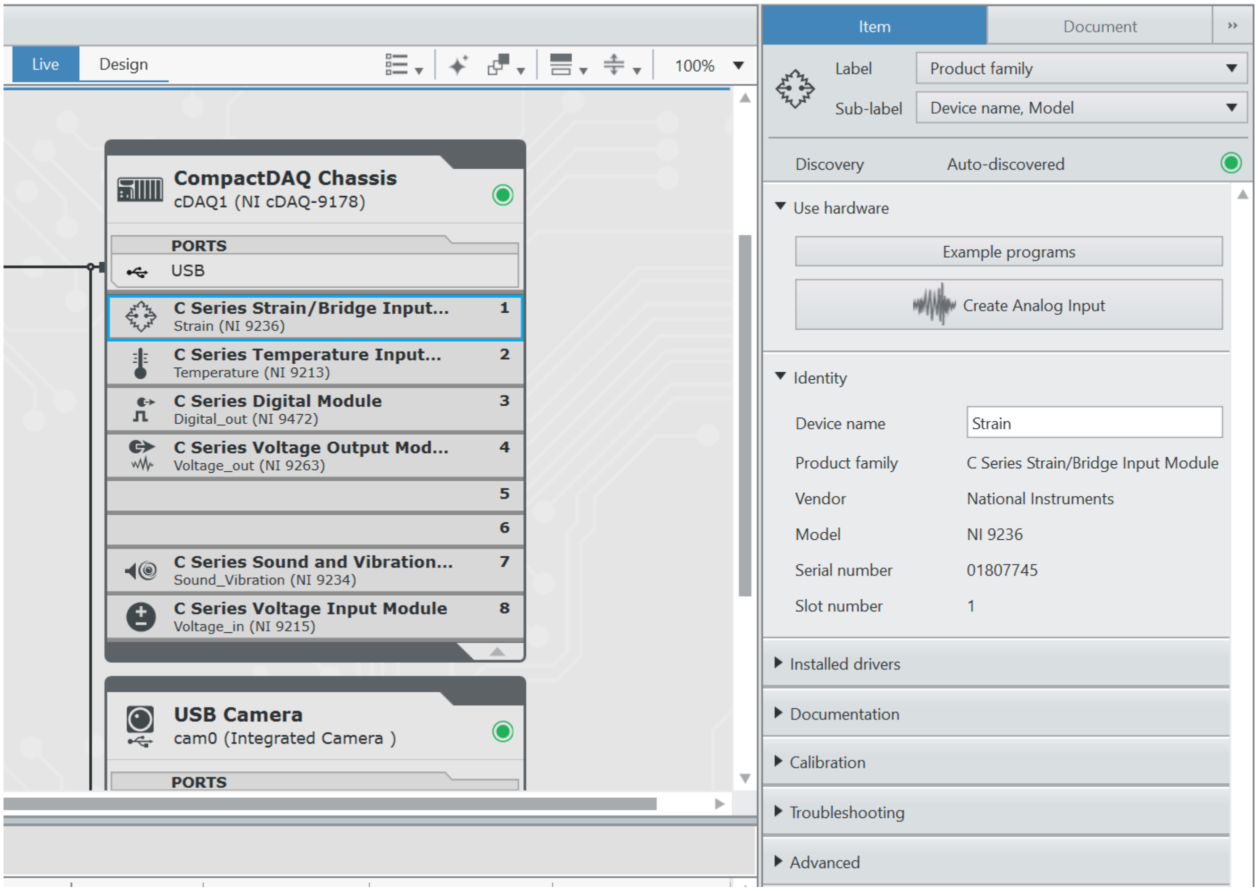


Label ▲	Model	Device name	Serial number	Hardware Detection
CompactDAQ Chassis	NI cDAQ-9178	cDAQ1	017803A8	● Auto-discovered
Slot 1 : C Series Strain/Bridge Input Module	NI 9236	cDAQ1Mod1	01807765	● Auto-discovered
Slot 2 : C Series Temperature Input Module	NI 9213	cDAQ1Mod2	01B9FD79	● Auto-discovered
Slot 3 : C Series Digital Module	NI 9472	cDAQ1Mod3	01241838	● Auto-discovered
Slot 4 : C Series Voltage Output Module	NI 9263	cDAQ1Mod4	012081F6	● Auto-discovered
Slot 7 : C Series Sound and Vibration Input Module	NI 9234	cDAQ1Mod7	01748E8C	● Auto-discovered
Slot 8 : C Series Voltage Input Module	NI 9215	cDAQ1Mod8	01719460	● Auto-discovered

0 item(s) selected

Figure 5-4: Display of all connected hardware with appropriate drivers installed.

6. Click on the C Series Strain/Bridge Input Module named Strain in your CompactDAQ chassis. The configuration pane to the right of the screen will populate with relevant information.



Label: Product family

Sub-label: Device name, Model

Discovery: Auto-discovered ●

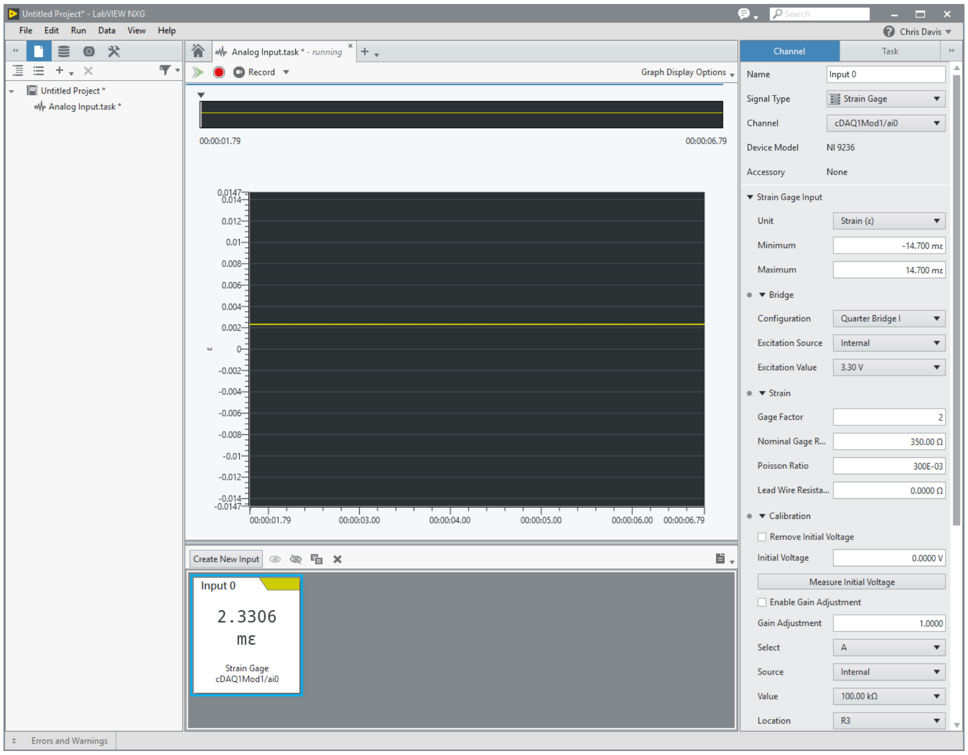
Device name	Strain
Product family	C Series Strain/Bridge Input Module
Vendor	National Instruments
Model	NI 9236
Serial number	01807745
Slot number	1

Figure 5-5: C Series module configuration window

7. Click **Create Analog Input** to begin acquiring a strain measurement in a measurement panel.

8. In the configuration pane on the right, switch to the **Task** page at the top, deselect **Auto Managed Timing**, and change the **Desired Sample Rate** to 1 kHz.

9. Data should automatically be acquired and displayed on the graph provided. The graph should look like Figure 5-6.



Name: Input 0
Signal Type: Strain Gage
Channel: cDAQ1Mod1/ai0
Device Model: NI 9236
Accessory: None

▼ Strain Gage Input
Unit: Strain (ε)
Minimum: -14.700 mε
Maximum: 14.700 mε

▼ Bridge
Configuration: Quarter Bridge I
Excitation Source: Internal
Excitation Value: 3.30 V

▼ Strain
Gage Factor: 2
Nominal Gage R...: 350.00 Ω
Poisson Ratio: 300E-03
Lead Wire Resista...: 0.0000 Ω

▼ Calibration
□ Remove Initial Voltage
Initial Voltage: 0.0000 V
Measure Initial Voltage
□ Enable Gain Adjustment
Gain Adjustment: 1.0000
Select: A
Source: Internal
Value: 100.00 kΩ
Location: R3

Figure 5-6: The configured measurement is used to confirm electrical connections and quickly acquire data.

10. 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.

11. Once you have confirmed that the strain gage is working, click the stop icon (●) to stop acquisition.

12. You can quickly capture measurement data using the measurement panels in LabVIEW NXG. To do so:

    a) Click **Record** Record to begin acquiring data
    b) Gently press down on the strain bar in your demo box to observe a change in measured strain
    c) Click **stop recording** ■ 0:00:18.

NOTE: 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. LabVIEW NXG's interactive measurements pane provides the insight into your system and setup to help you eliminate mistakes early in your development process.

13. The Navigation Pane to the left of your screen will switch to the Captured Data view and display your saved data file. Explore the captured data by double clicking on the data file, named **Recording 1** by default. Take a minute to familiarize yourself with the functionality available while viewing captured data.

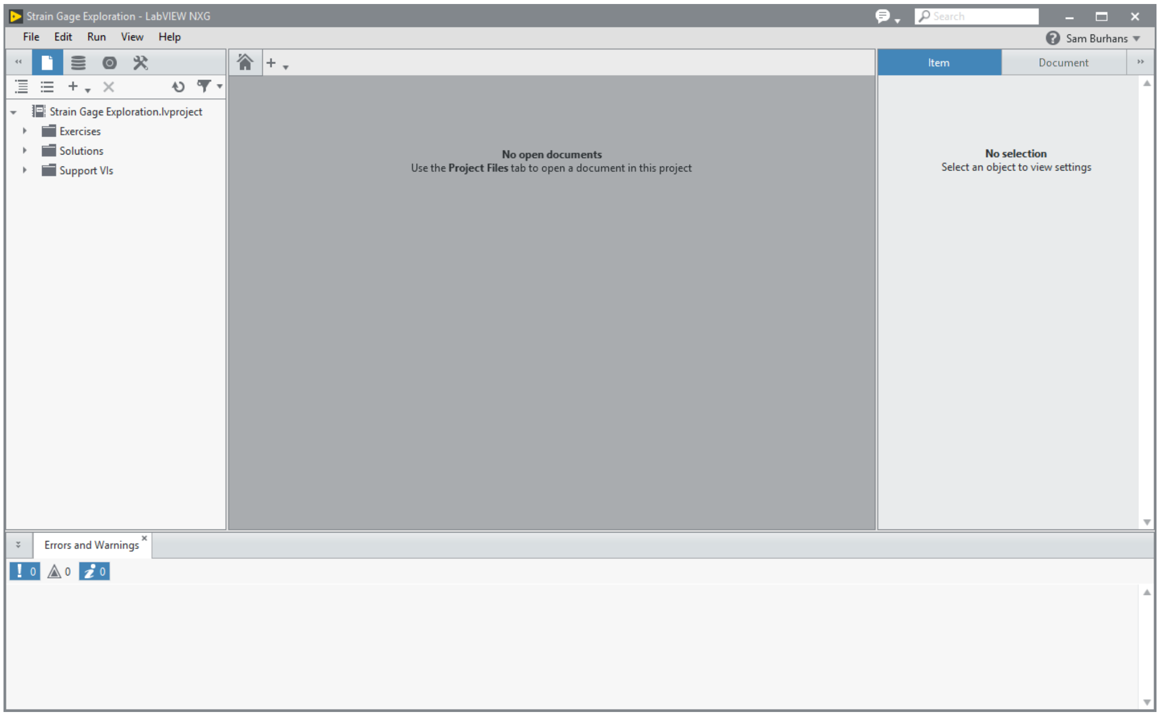
> **NOTE**: You can see a reminder of what panes are currently visible by clicking on the question mark icon ❓ at the top right of any window.

a) Browse the waveform data using your mouse to view individual data points.
b) Adjust the **cursors** ⛉ above the waveform chart to highlight points of interest.
c) Use **Ctrl+Mouse Scroll Wheel** to zoom in to a smaller section of the recorded data. Notice that the macro view at the top of the data viewer window shows the area you are currently viewing and allows you to further adjust as needed.
d) Right click the graph to capture a new data set equal to whatever is currently visible, if desired.

14. When finished, close LabVIEW NXG and discard changes if prompted.

15. Navigate to *C:\Seminars\cDAQ + LabVIEW NXG HO\Exercises\5 – Strain*.

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



No open documents
Use the **Project Files** tab to open a document in this project

No selection
Select an object to view settings

Figure 5-7 Use LabVIEW Projects to organize all your VIs.

With the LabVIEW Project like the one you've just opened; you can easily organize and visualize all the files that are important to your application. In this project, the folders like Exercises, Solutions, and Support VIs have been populated with the files within them. The file structure of your project should mirror the file structure in Windows Explorer.

17. Start by opening the solution. In the Navigation Pane, expand **Strain Gage Exploration.lvproject » Solutions** and double-click **Strain_w_User_Feedback_Solution.gvi**.

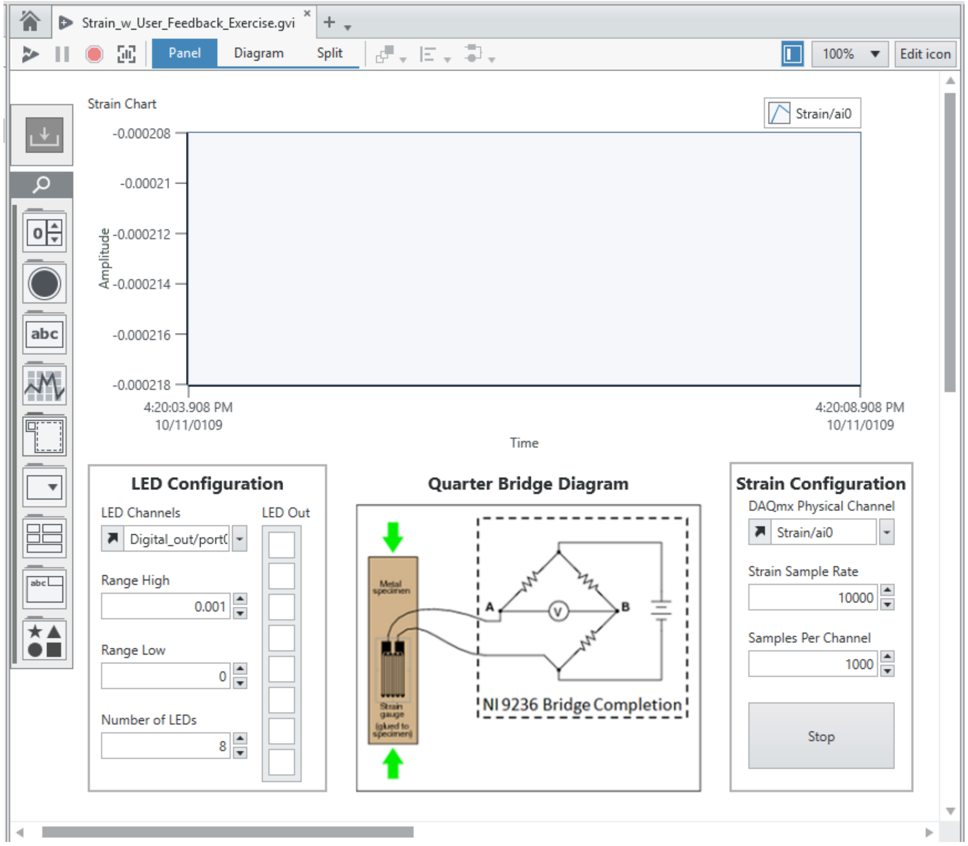
18. Click the **Run** arrow at the top-left of the front panel pane, and then push down on the strain bar – be careful not to bend the bar. Notice that the LEDs signal the amount of strain experienced on the strain bar.

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

20. To start this exercise, use the Navigation Pane to navigate to the Exercises folder and double-click **Strain_w_User_Feedback_Exercise.gvi**. The front panel has already been created for you.

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

NOTE: 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 wiring the block diagram to support that functionality.



LED Configuration

LED Channels: Digital_out/port(

Range High: 0.001

Range Low: 0

Number of LEDs: 8

Strain Configuration

DAQmx Physical Channel: Strain/ai0

Strain Sample Rate: 10000

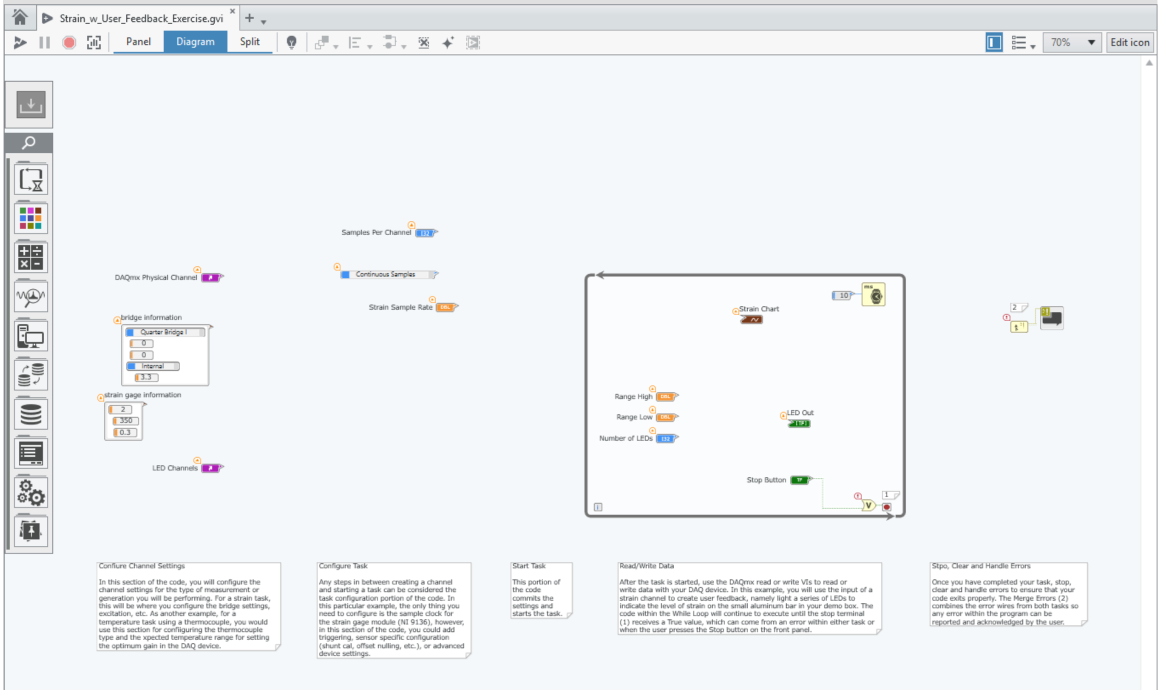
Samples Per Channel: 1000

Figure 5-8 The 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 left corner is broken . This is because the code contains errors, or in this case, is not in a running state. LabVIEW NXG continuously compiles in the editing environment, so you will always know if your code can 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 NXG will even highlight the section of code containing the error.

21. Close the Error Dialog and select **Diagram** at the top of the VI. The keyboard shortcut for this is **Ctrl+E**. Use this shortcut to toggle back and forth between the Front Panel and Block Diagram. Alternatively, you can use the **Split** view to display both the Panel and Diagram at the same time. When the Block Diagram opens, you will see the following code:



Confiure Channel Settings

In this section of the code, you will configure the channel settings for the type of measurement or generation you will be performing. For a strain task, this will be where you configure the bridge settings, excitation, etc. As another example, for a temperature task using a thermocouple, you would use this section for confiiguring the thermocouple type and the xpected temperature range for setting the optimum gain in the DAQ device.

Configure Task

Any steps in between creating a channel and starting a task can be considered the task configuration portion of the code. In this particular example, the only thing you need to configure is the sample clock for the strain gage module (NI 9136), however, in this section of the code, you could add triggering, sensor specific configuration (shunt cal, offset nulling, etc.), or advanced device settings.

Start Task

This portion of the code commits the settings and starts the task.

Read/Write Data

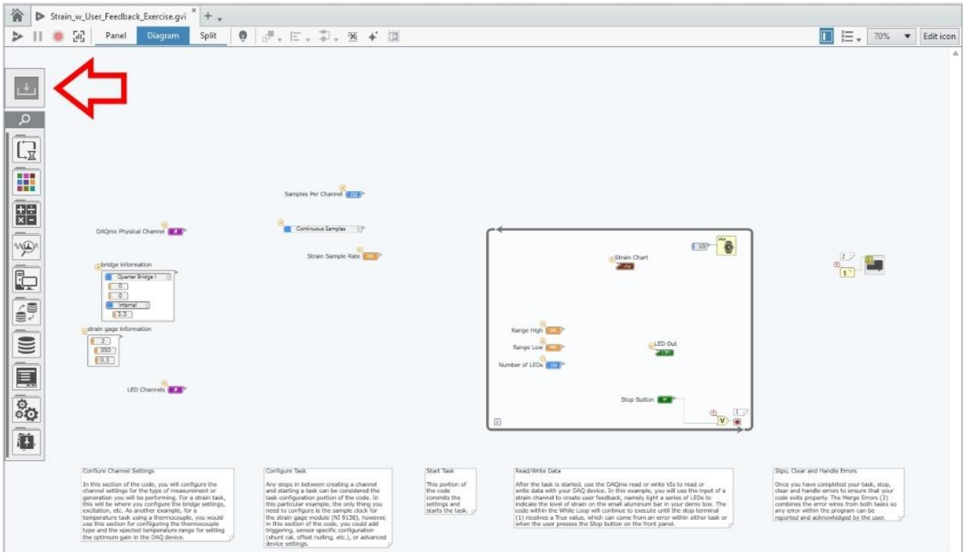
After the task is started, use the DAQmx read or write VIs to read or write data with your DAQ device. In this example, you will use the input of a strain channel to create user feedback, namely light a series of LEDs to indicate the level of strain on the small aluminum bar in your demo box. The code within the While Loop will continue to execute until the stop terminal (1) receives a True value, which can come from an error within either task or when the user presses the Stop button on the front panel.

Stpo, Clear and Handle Errors

Once you have completed your task, stop, clear and handle errors to ensure that your code exits properly. The Merge Errors (2) combines the error wires from both tasks so any error within the program can be reported and acknowledged by the user.

Figure 5-9 The Block Diagram does the work of your program. This is where you will write the code to control the Front Panel.

This Block Diagram includes some basic elements that have been provided for you. 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. As you add new elements to the front panel, their corresponding block diagram element will be held in the **Unplaced Items** section of the **Palette** (arrow in Figure 5-10).



Confiure Channel Settings

In this section of the code, you will configure the channel settings for the type of measurement or generation you will be performing. For a strain task, this will be where you configure the bridge settings, excitation, etc. As another example, for a temperature task using a thermocouple, you would use this section for confiiguring the thermocouple type and the xpected temperature range for setting the optimum gain in the DAQ device.

Configure Task

Any steps in between creating a channel and starting a task can be considered the task configuration portion of the code. In this particular example, the only thing you need to configure is the sample clock for the strain gage module (NI 9136), however, in this section of the code, you could add triggering, sensor specific configuration (shunt cal, offset nulling, etc.), or advanced device settings.

Start Task

This portion of the code commits the settings and starts the task.

Read/Write Data

After the task is started, use the DAQmx read or write VIs to read or write data with your DAQ device. In this example, you will use the input of a strain channel to create user feedback, namely light a series of LEDs to indicate the level of strain on the small aluminum bar in your demo box. The code within the While Loop will continue to execute until the stop terminal (1) receives a True value, which can come from an error within either task or when the user presses the Stop button on the front panel.

Stpo, Clear and Handle Errors

Once you have completed your task, stop, clear and handle errors to ensure that your code exits properly. The Merge Errors (2) combines the error wires from both tasks so any error within the program can be reported and acknowledged by the user.

Figure 5-10 New elements are stored in the Unplaced Items palette for easy access as you add functionality to your programs.

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 grey text 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 or someone else can understand what is going on.

REMEMBER: you can zoom in and out on the Front Panel and Block Diagram by holding the Ctrl key and scrolling the mouse scroll wheel.

22. 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.

23. For this exercise, a large part of the coding will come from the DAQmx palette. To access this palette click the **Hardware Interfaces** menu of palette, then select NI-DAQmx.

24. Click on the **double arrow** « to keep this palette visible (arrow in Figure 5-11).

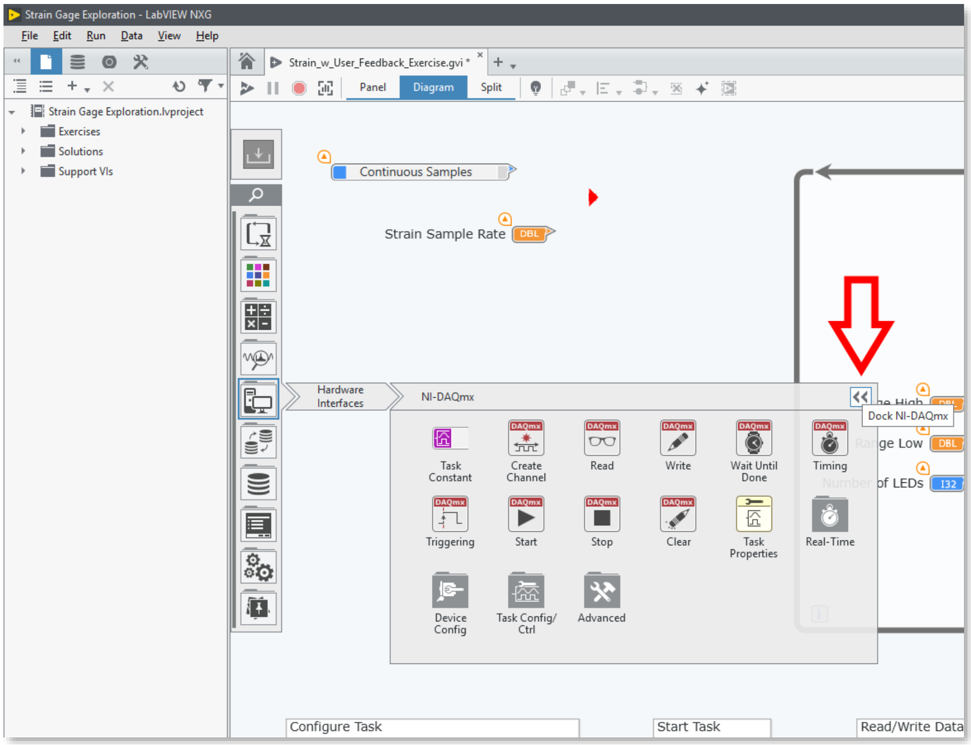
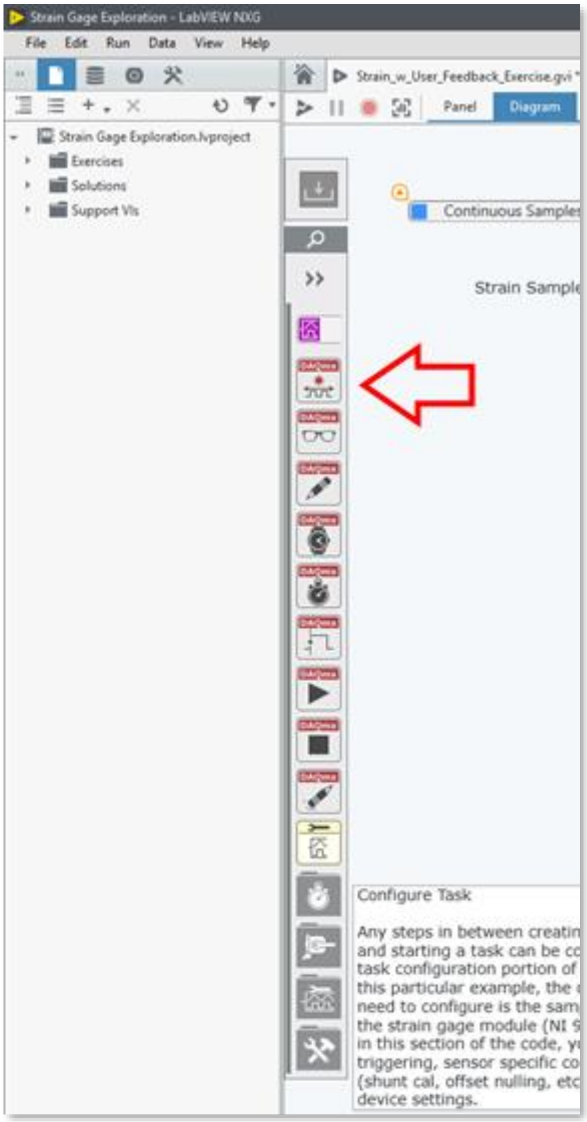


Figure 5-11 The DAQmx palette includes all the required VIs for communicating with your NI CompactDAQ chassis.

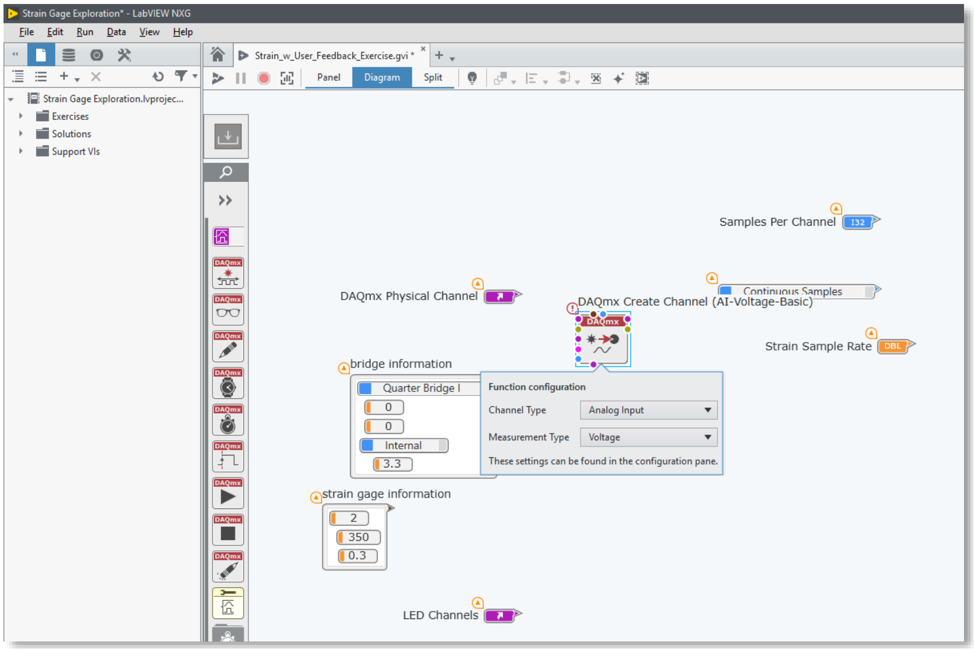

Strain Gage Exploration - LabVIEW NXG
File Edit Run Data View Help
Strain Gage Exploration.lvproject
Exercises
Solutions
Support VIs
Strain_w_User_Feedback_Exercise.gvi *
Panel
Diagram
Continuous Sample
Strain Sample

Configure Task

Any steps in between creatin
and starting a task can be co
task configuration portion of
this particular example, the
need to configure is the sam
the strain gage module (NI 9
in this section of the code, y
triggering, sensor specific co
(shunt cal, offset nulling, etc
device settings.

Figure 5-12 Pinning a palette enables you to click elsewhere in your code with the palette still visible.

25. To start, drag the **NI-DAQmx Create Virtual Channel** (arrow in Figure 5-12) and place it in the position shown in Figure 5-13.



Function configuration

Channel Type: Analog Input

Measurement Type: Voltage

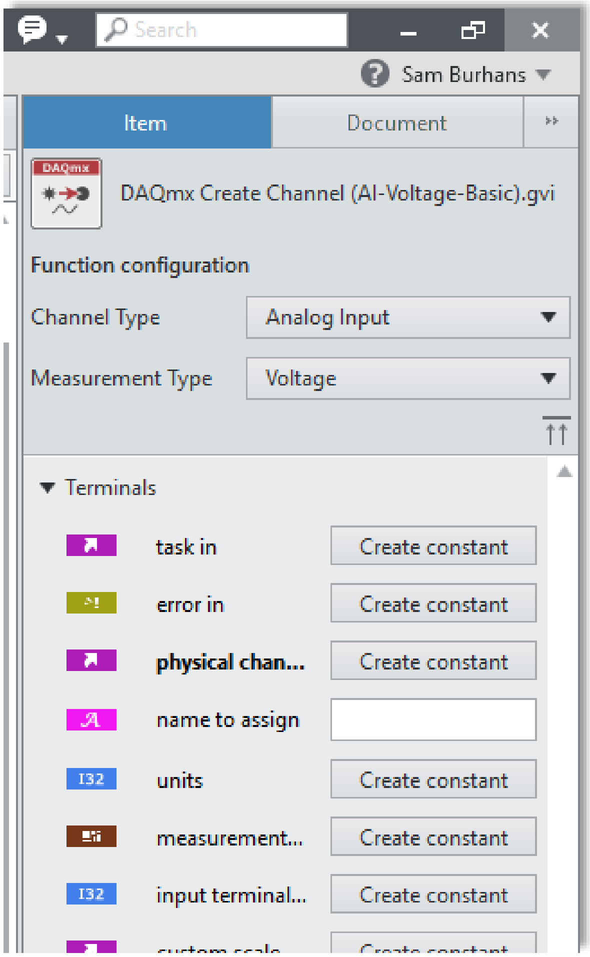
These settings can be found in the configuration pane.

Figure 5-13 The DAQmx Create Channel VI establishes communication with your DAQ device.

26. When this VI is placed on the diagram, a configuration menu opens in the right-hand configuration pane to specify the desired Channel and Measurement Type as shown in Figure 5-13. By default, the VI is configured to create a simple analog input voltage. To configure this for strain, click on the **Measurement Type** drop down arrow and select **Strain.** The Sensor Type field that appears should be configured for Strain Gage by default.

    If you make an incorrect selection, or otherwise need to adjust the configuration of this VI, you can do so by selecting the VI on the Block Diagram to view all available configuration settings in the **Configuration Pane** to the right.

NOTE: You can see a reminder of what panes are currently visible by clicking on the question mark icon ? at the top right of any window.



DAQmx Create Channel (AI-Voltage-Basic).gvi

Function configuration

Channel Type: Analog Input

Measurement Type: Voltage

Figure 5-14 The DAQmx Create Channel VI is called polymorphic because it changes to adapt to the selected measurement type. The Configuration of polymorphic VIs can be changed in the configuration pane.

27. 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 5-15 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.

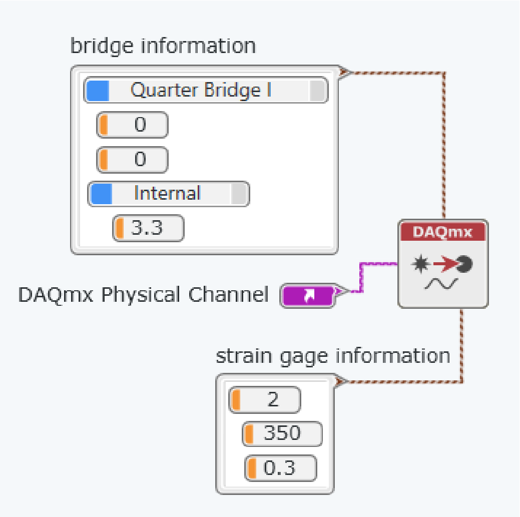


Figure 5-15 Wire up the inputs to the DAQmx Create Channel VI.

28. Drag the following VIs from the DAQmx palette and place them as shown in Figure 5-16.
    a) DAQmx Timing
    b) DAQmx Start Task
    c) DAQmx Read
    d) DAQmx Stop Task
    e) DAQmx Clear Task

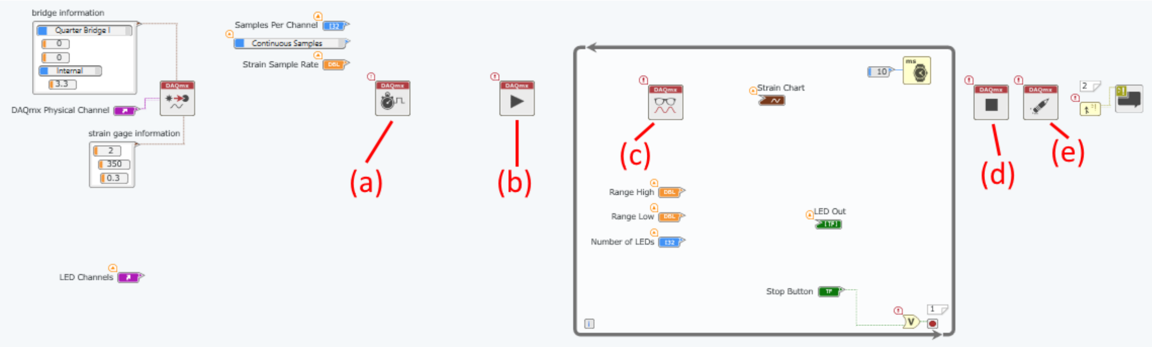


Figure 5-16 The flow of DAQmx code almost always follows the same pattern.

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.

29. 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:

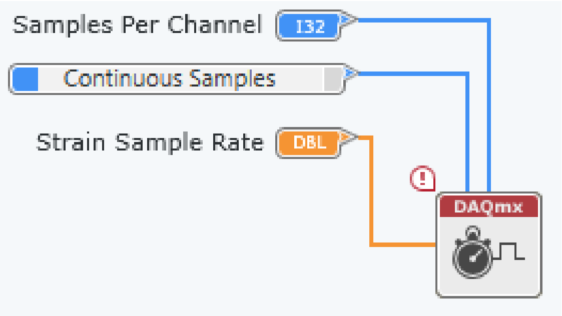
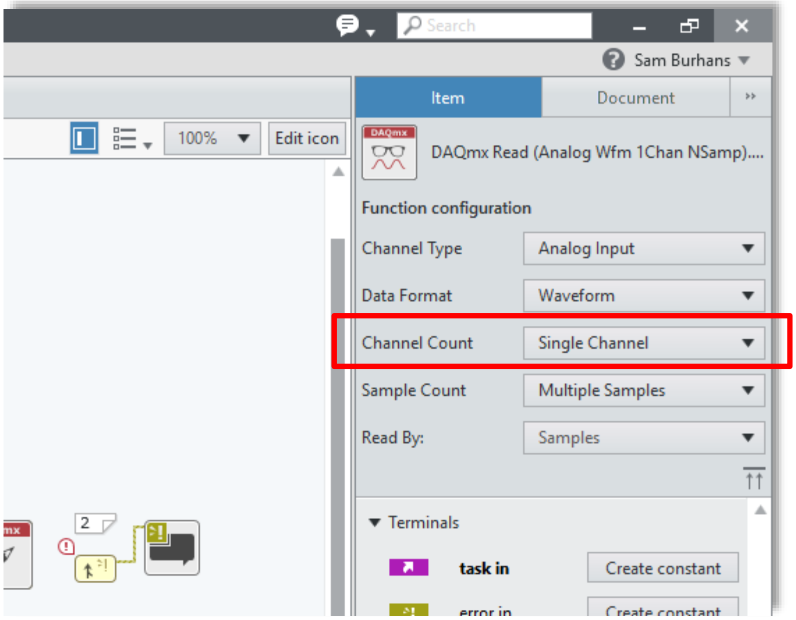


Figure 5-17 The DAQmx Timing VI configures your clock parameters.

30. 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.

31. 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 VI and observe the settings available in the **Configuration Pane**. Set the function configuration settings as shown in Figure 5-18.



DAQmx Read (Analog Wfm 1Chan NSamp)....

Function configuration

Channel Type: Analog Input

Data Format: Waveform

Channel Count: Single Channel

Sample Count: Multiple Samples

Read By: Samples

Terminals

task in — Create constant

error in — Create constant

Figure 5-18 Select a single channel of waveform measurement to read from the supplied strain gage.

32. 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.

33. 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.

34. Wire the purple task and yellow error wires through the code as shown in Figure 5-19. This will pass the task values and any errors that may arise through the code.

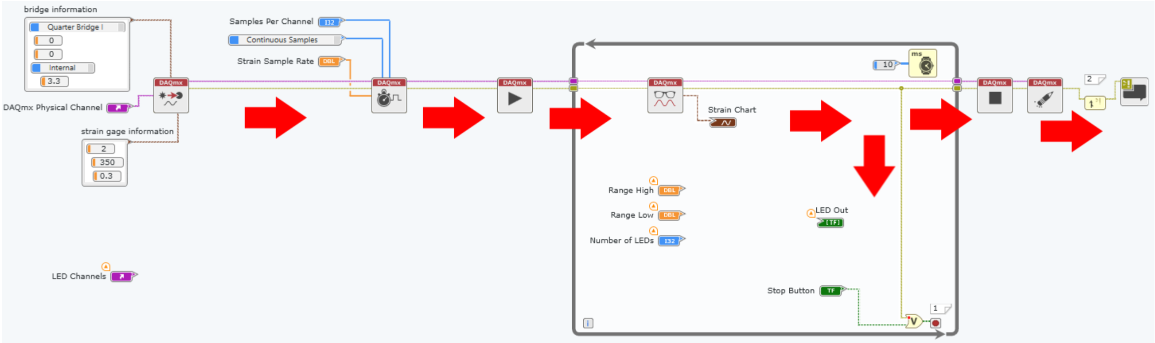


Figure 5-19 To finish the strain acquisition code, wire the task and error wires through the code.

35. Wire the error wire from the DAQmx Read VI to the input of the OR Node. 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.

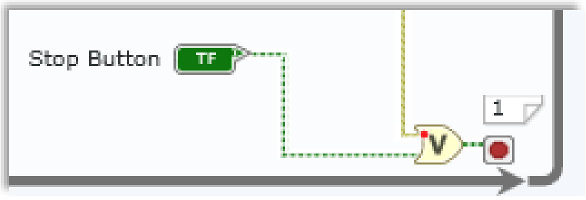


Figure 5-20 The Compound Arithmetic ORs the inputs, stopping the While Loop if the code has an error or the user presses stop.

NOTE: 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.

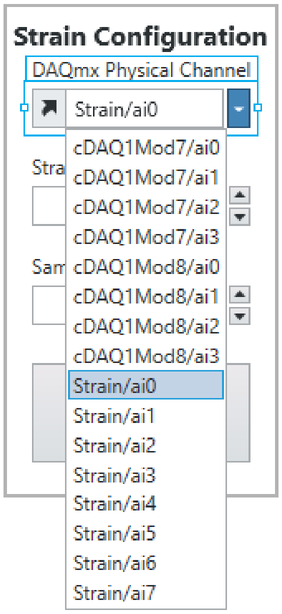
Wiring the error wire thought the code ensures that any errors in the code will pass through the code and be reported to the user.

NOTE: To create a branch from an existing wire, right click on the wire and select **Create branch**, or click a completed wire while holding **Ctrl**.

36. Once you wire the error wire to the OR node, your code should be ready to run. Check to see that the Run Arrow is intact ▷.

37. Press **Ctrl+E** to toggle to the Front Panel.

38. 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**.



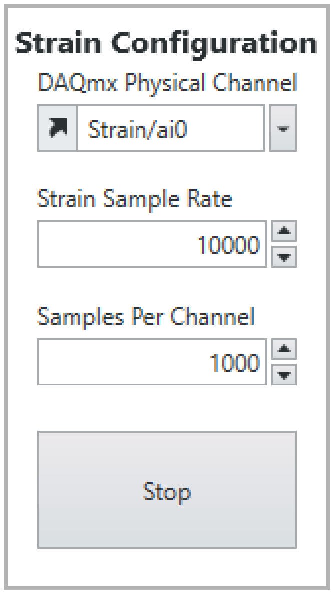
Strain Configuration

DAQmx Physical Channel

Strain/ai0

Figure 5-21 Select the channel that your strain gage is attached to.

39. Set the *Strain Sample Rate* to 10000 Hz and the *Samples Per Channel* to 1000, like in Figure 5-22.



Strain Configuration

DAQmx Physical Channel

Strain/ai0

Strain Sample Rate

10000

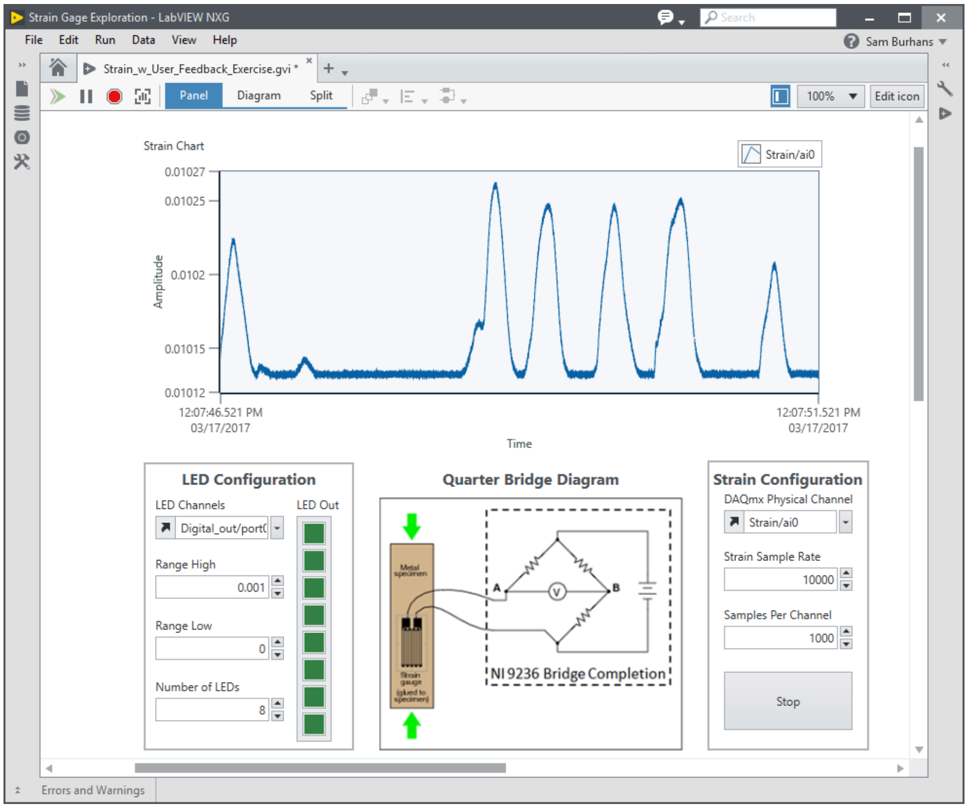
Samples Per Channel

1000

Figure 5-22 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* in the in-product help search at the top right of the window.

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



LED Configuration

LED Channels: Digital_out/port(

LED Out

Range High: 0.001

Range Low: 0

Number of LEDs: 8

Strain Configuration

DAQmx Physical Channel: Strain/ai0

Strain Sample Rate: 10000

Samples Per Channel: 1000

Figure 5-23 You have created a program to visualize strain gage data. This data can be analyzed, saved to file or used to control an output.

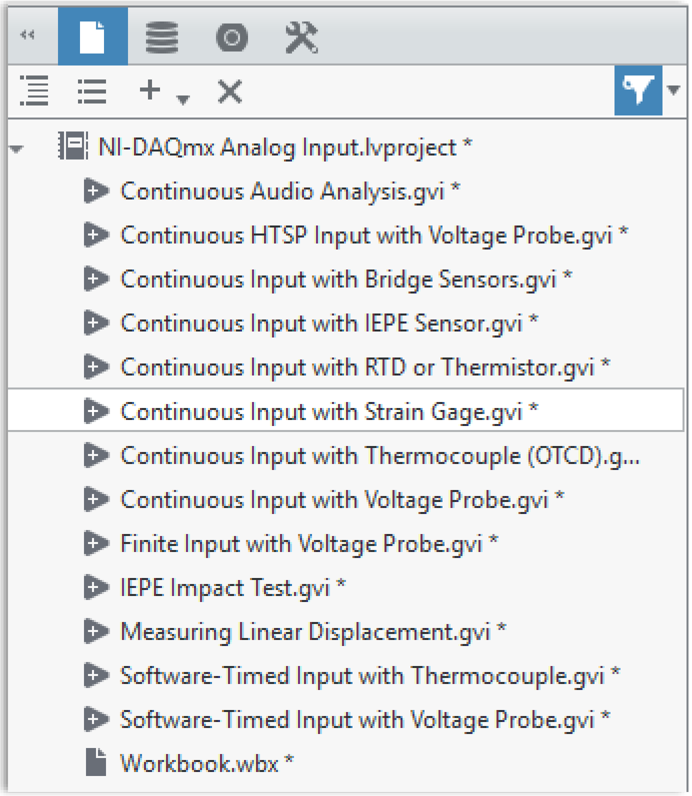
41. Press **Stop**.

42. Save this VI, as you will use it in the next part of the exercise.

43. To finish out this portion of the exercise, return to the home screen of LabVIEW NXG by selecting the **Home Icon** 🏠 at the top left of the window. Then, enter the Learning section by clicking **Learning** at the top right of the home screen.

44. Navigate to **Examples » Hardware Input and Output » NI-DAQmx » Programmatic Configuration » NI-DAQmx Analog Input.** Save the file with any file name.

45. This example is a project with a variety of pre-built analog input applications for different measurement types. You can see all the included VIs in the Navigation Pane, as shown in Figure 5-24.



NI-DAQmx Analog Input.lvproject *

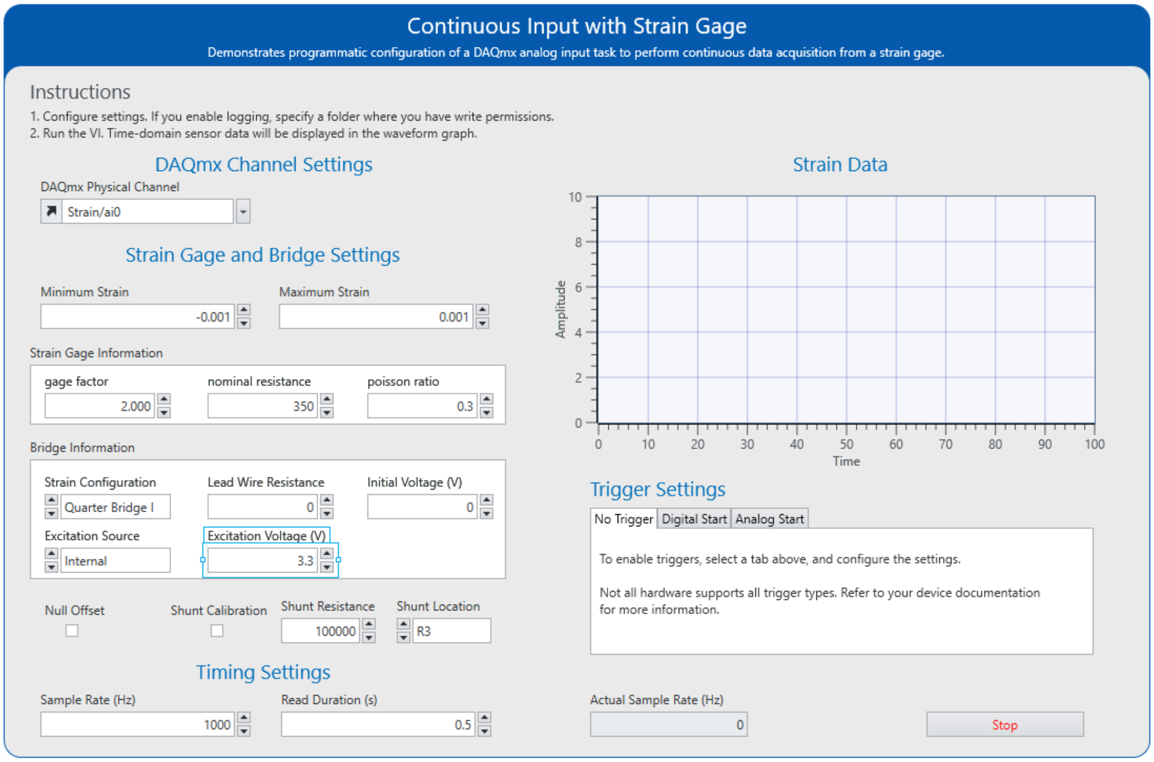
- Continuous Audio Analysis.gvi *
- Continuous HTSP Input with Voltage Probe.gvi *
- Continuous Input with Bridge Sensors.gvi *
- Continuous Input with IEPE Sensor.gvi *
- Continuous Input with RTD or Thermistor.gvi *
- Continuous Input with Strain Gage.gvi *
- Continuous Input with Thermocouple (OTCD).g...
- Continuous Input with Voltage Probe.gvi *
- Finite Input with Voltage Probe.gvi *
- IEPE Impact Test.gvi *
- Measuring Linear Displacement.gvi *
- Software-Timed Input with Thermocouple.gvi *
- Software-Timed Input with Voltage Probe.gvi *
- Workbook.wbx *

Figure 5-24 Example programs are pre-built and ready to run. They serve as a great starting point for your measurement applications.

46. Double-click on the Continuous Input with Strain Gage.gvi to open the Front Panel.

47. Change the *DAQmx 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 5-25 for reference.

NOTE: Examples and Lessons in LabVIEW NXG often include Workbooks that will open when the project is opened. Workbooks will explain the content of each lesson in an interactive, step-by-step guide.



Continuous Input with Strain Gage

Demonstrates programmatic configuration of a DAQmx analog input task to perform continuous data acquisition from a strain gage.

Instructions

1. Configure settings. If you enable logging, specify a folder where you have write permissions.
2. Run the VI. Time-domain sensor data will be displayed in the waveform graph.

DAQmx Channel Settings

DAQmx Physical Channel: Strain/ai0

Strain Gage and Bridge Settings

Minimum Strain: -0.001

Maximum Strain: 0.001

Strain Gage Information

gage factor: 2.000

nominal resistance: 350

poisson ratio: 0.3

Bridge Information

Strain Configuration: Quarter Bridge I

Lead Wire Resistance: 0

Initial Voltage (V): 0

Excitation Source: Internal

Excitation Voltage (V): 3.3

Null Offset: □

Shunt Calibration: □

Shunt Resistance: 100000

Shunt Location: R3

Timing Settings

Sample Rate (Hz): 1000

Read Duration (s): 0.5

Trigger Settings

No Trigger | Digital Start | Analog Start

To enable triggers, select a tab above, and configure the settings.

Not all hardware supports all trigger types. Refer to your device documentation for more information.

Actual Sample Rate (Hz): 0

Figure 5-25 The DAQmx examples are built to give you a solid starting point for your programs. Use them as a starting point.

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

49. 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 triggering functionality. This could easily be added to your code by adding the appropriate DAQmx Trigger VI.

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

Part B Controlling the Colored LEDs for User Feedback

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 Panels.

1. If needed, re-open **Strain Gage Exploration.lvproj** from *C:\Seminars\cDAQ + LabVIEW NXG HO\Exercises\5 – Strain*.

2. Click the **Add File** icon + in the tab toolbar at the top of the window, or navigate to **File » New,** and select **Digital Output**, as in Figure 5-26. This will create a new digital output Measurement Panel.

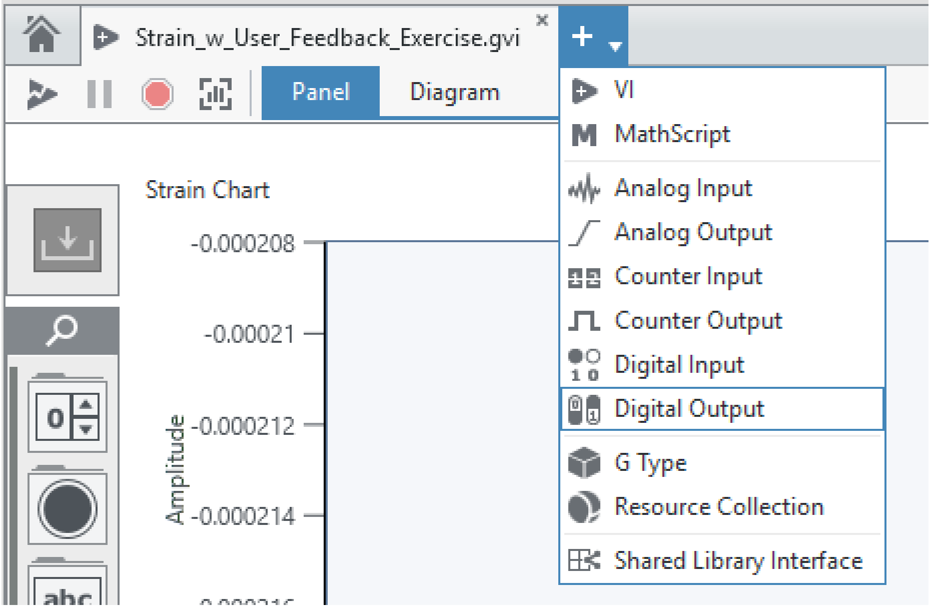
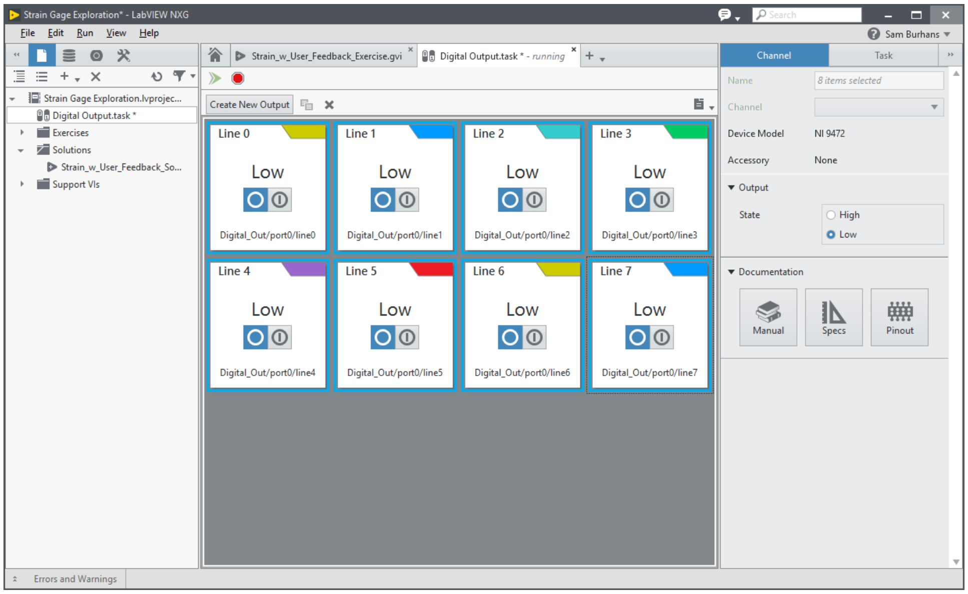


Figure 5-26 Starting with Measurement Panels ensures that your sensors are working and you will not have hardware problems.

3. Click **Create New Output**, check **Select All** and click **OK**. This will add all available lines to the Measurement Panel:



Name: 8 items selected

Channel:

Device Model: NI 9472

Accessory: None

State: ○ High ● Low

Figure 5-27 With the Digital Output Measurement Panel, you can control the digital output lines to ensure they are operating properly.

4. Press **Run** ▷.

5. For this test, select all lines using **Shift+Click**. Once selected, the line state can be changed in the Channel tab of the Configuration Pane. Set all lines to **High**.

6. You should see all the LEDs on your demo box light up. Notice that the LEDs on the CompactDAQ NI 9474 module also light up. You can control each LED individually, but for testing operability, it is not essential.

7. Press **Stop** and close the Measurement Panel. Discard Changes.

8. Open your copy of **Strain_w_User_Feedback_Exercise.gvi** from **Exercises** folder of the Project Files view in the Navigation pane.

9. 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.

10. From the **Hardware Interfaces** menu of the palette, drag the **NI-DAQmx Create Channel** (arrow in **Error! Reference source not found.**) and place it in the position shown in Figure 5-28.

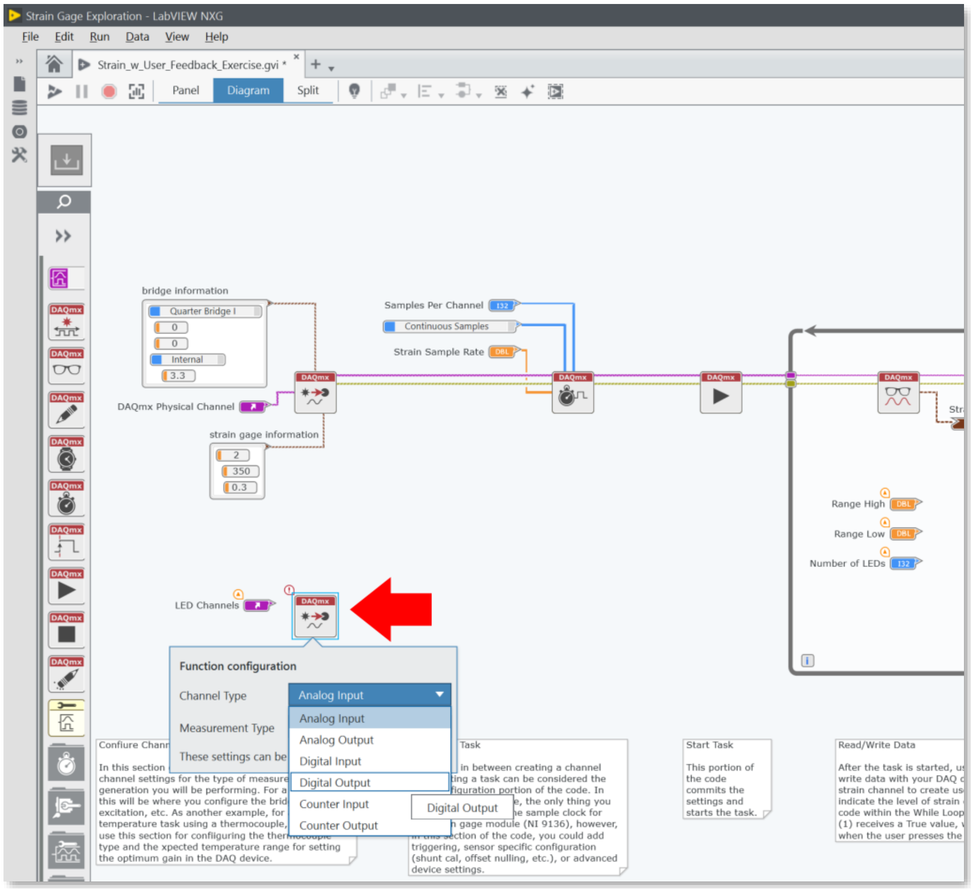


Figure 5-28 The digital output code will run parallel to the strain task.

11. When this VI is placed on the diagram, a configuration menu opens to specify the desired Channel and Measurement Type as shown in Figure 5-13. By default, the VI is configured to create a simple analog input voltage. To configure this for digital output, click on the Measurement Type drop down arrow and select **Digital Output.**

    If you make an incorrect selection, or otherwise need to adjust the configuration of this VI, you can do so by selecting the VI on the Block Diagram to view all available configuration settings in the **Configuration Pane** to the right.

12. Wire the LED Channels control to the Physical Channels 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 5-29.
    a) DAQmx Start Task
    b) DAQmx Write
    c) DAQmx Stop Task
    d) DAQmx Clear Task

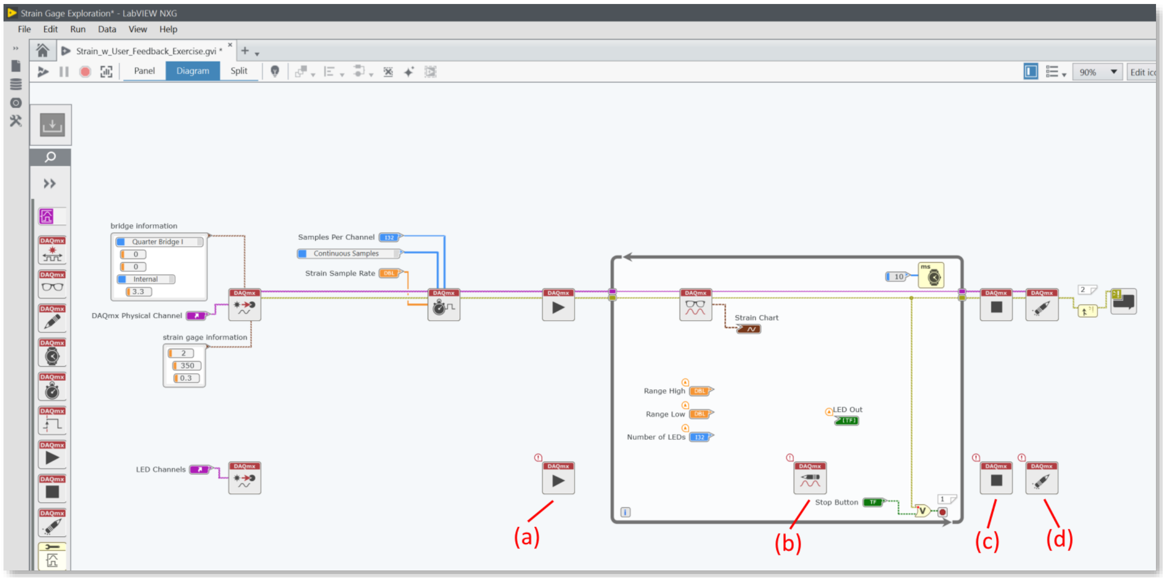
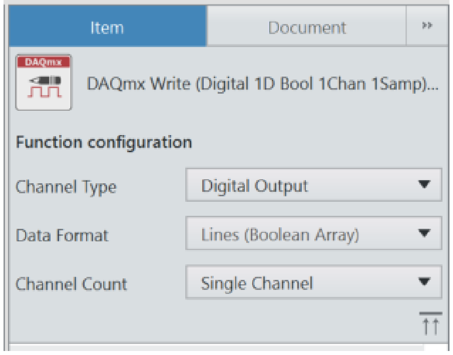


Figure 5-29 These VIs should look familiar – nearly all DAQ programs will follow this pattern.

14. Click on the new **DAQmx Write VI** and change the Function Configuration in the Item tab of the Configuration Pane as shown in Figure 5-30.



DAQmx Write (Digital 1D Bool 1Chan 1Samp)...

Function configuration

Channel Type: Digital Output

Data Format: Lines (Boolean Array)

Channel Count: Single Channel

Figure 5-30 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 5-31.

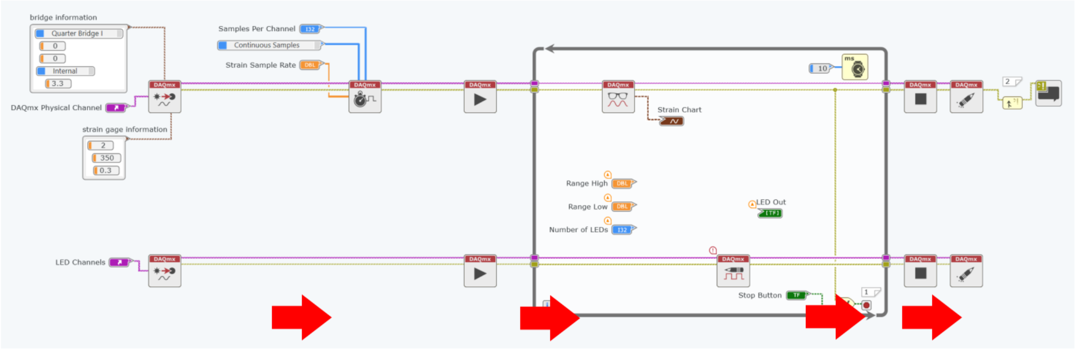


Figure 5-31 Wiring the task and error wires through dictates the flow of your code.

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 OR function to accept three inputs. Hover over the VI and drag the handle up to expose another input.

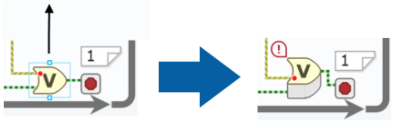


Figure 5-32 Expanding the OR for additional inputs.

17. Wire the error wire from the output of the DAQmx Write to the new compound OR input.

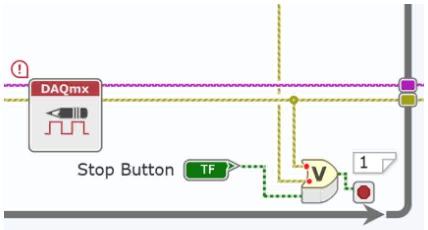


Figure 5-33 The Compound OR now accounts for all the errors inside the While Loop.

NOTE: To create a branch from an existing wire, click a completed wire while holding **Ctrl** , or right click on the wire and select **Create branch**.

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. The Retain First Error function merges multiple error clusters into a single error, looking for errors from top to bottom beginning with the first error in parameter.

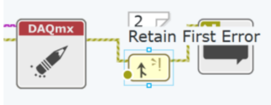


Figure 5-34 Combine all the error wires in your code with the Retain First Error VI.

19. Wire the error wire from the output of the bottom DAQmx Clear Task VI into the second input of the Retain First Error function. Your code should look like Figure 5-35.

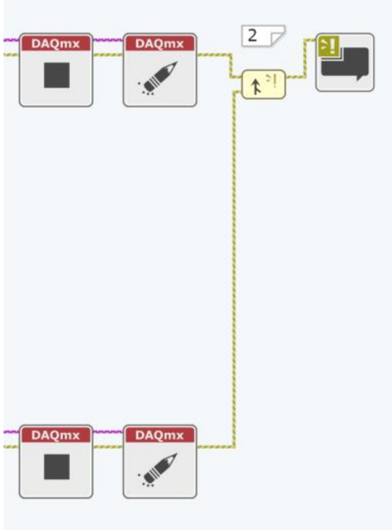


Figure 5-35 All your error wires are combines so the Simple Error Handler can display the errors to the user.

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, view the Project Files within the Navigation Pane. Under the Support Vis folder, find the **LED_Scaled_Output VI**. Drag and drop this VI on to your block diagram as shown in Figure 5-36.

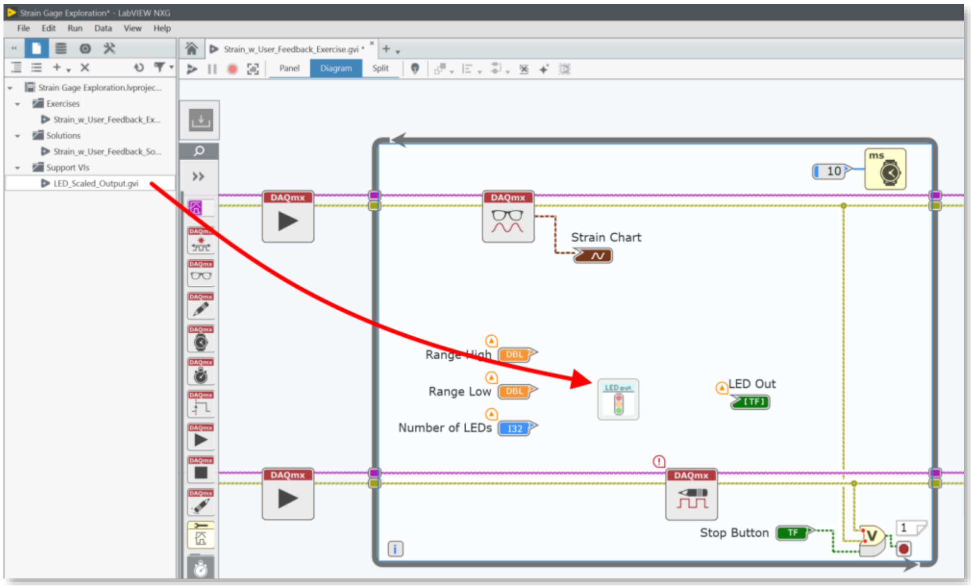


Figure 5-36 The LED_Scaled_Output VI is a subVI to scale the strain input to an LED output.

20. Organizing your code into subVIs is 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.

21. Close the subVI.

22. Wire the call to the subVI like in Figure 5-37.

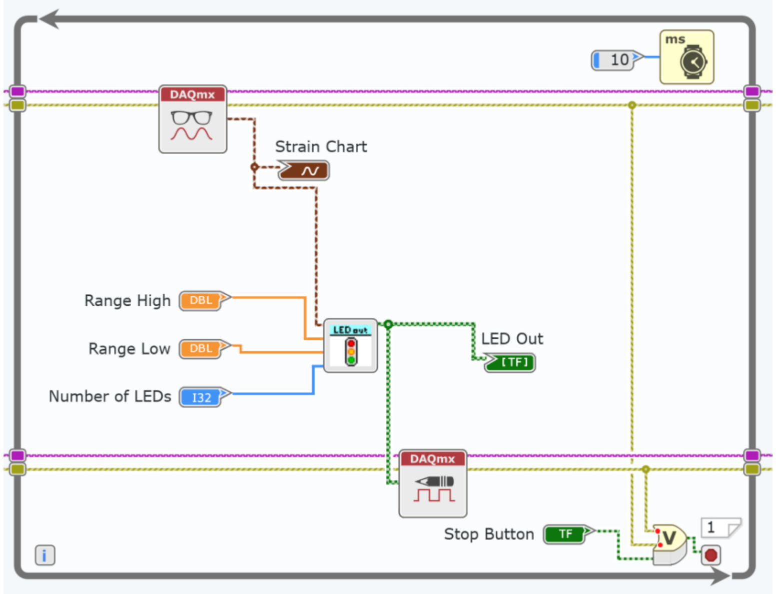


Figure 5-37 This subVI outputs a Boolean array that can then be output by the DAQmx Write.

23. 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.

24. 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, click on the wire shown in Figure 5-38 and press delete.

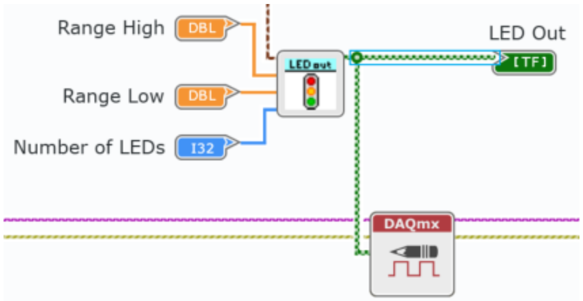
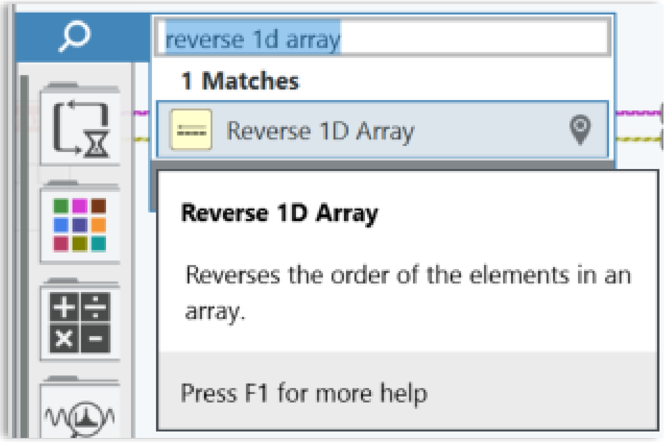


Figure 5-38 Delete the wire between the LED subVI and the LED indicator.

25. To switch the order of a 1D array, use the Reverse 1D Array function. To quickly find and place a function that you know the name of, use the quick drop utility. Press **Ctrl+Space** to open the quick drop search, and type **Reverse 1D Array**.



reverse 1d array

1 Matches

Reverse 1D Array

Reverse 1D Array

Reverses the order of the elements in an array.

Press F1 for more help

Figure 5-39 The quick drop is the fastest way to find a function in the palette by name.

26. Place the VI in between the LED out subVI and the LED out indicator and wire it up like in .

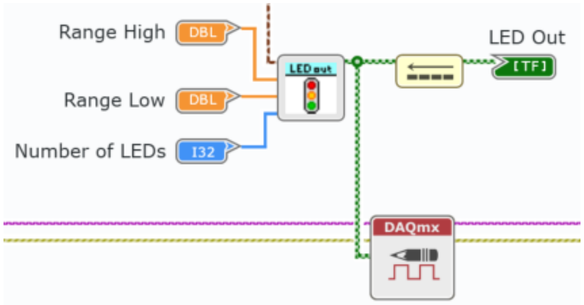


Figure 5-40 Use Reverse 1D Array to swap the order that the Boolean array is displayed.

<End of Exercise>

Chapter 6 Audio Equalizer – Audio Input and Output with Analysis

Exercise 1: Acquire and Log 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.

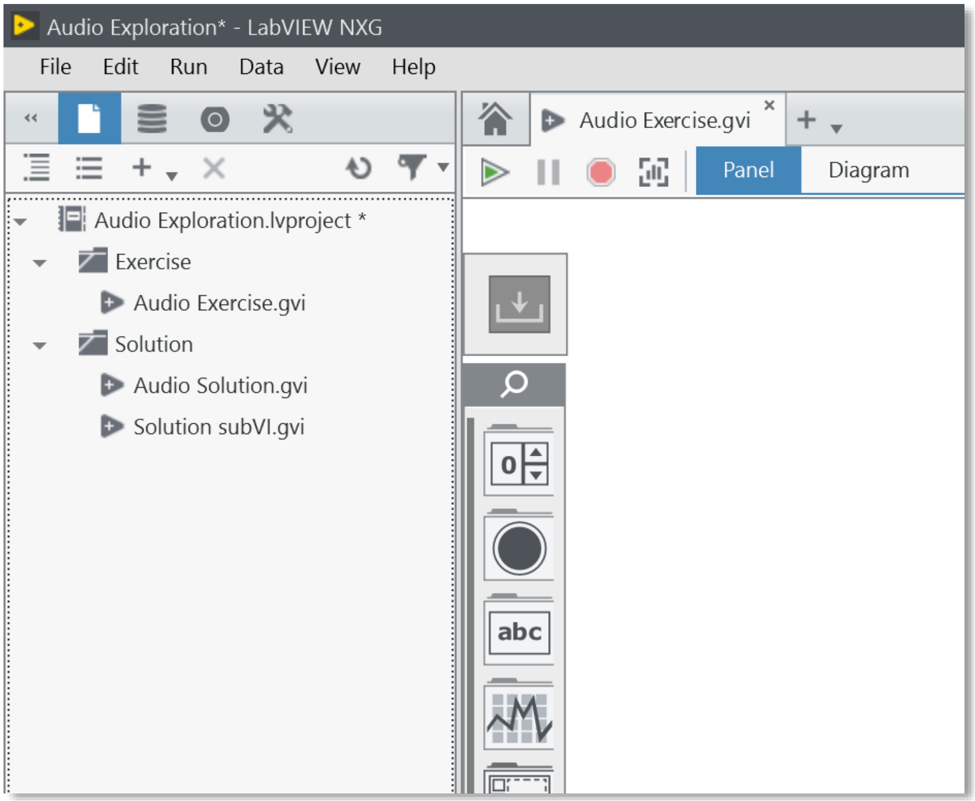
Goals

- Continuously acquire 40000 data point arrays from the audio source connected to the NI 9215 (or similar Analog Input) module.
- Add data logging to the DAQmx task.

1. Launch LabVIEW NXG by navigating to **Windows » LabVIEW NXG.**

2. Open the provided LabVIEW Project by selecting the **File » Open Project » Audio Equalizer Exploration.lvproject** from *C:\Seminars\cDAQ + LabVIEW NXG HO\Exercises\Audio\* option from the Getting Started Window.

3. Within the **project files** view of the **navigation pane**, you have been provided with a few LabVIEW Virtual Instruments (VIs).



4. Open Audio Exercise.gvi by double-clicking on it in the project files pane.

5. Note that some front panel (user interface) elements have been provided for you. It is often a good idea to start with the user interface to provide a framework around which to build your application.

6. Navigate to the Block Diagram window of Audio Exercise.gvi. If you are on the Front Panel, press **Ctrl + E** to toggle between Block Diagram and Front Panel, or use the **Panel** and **Diagram** buttons at the top of the VI.

   Take a second to familiarize yourself with the block diagram interface. This is where you will write graphical code to interface with DAQ hardware and develop custom user interfaces to interface with your user.

   The palette along the left of the VI is where functions and user interface elements reside. You can manually navigate the palette menus to browse for a function or element, or use the search function at the top of the palette to find an item by name.

   The items available in the palette will change when looking at the diagram versus when looking at the panel. In the diagram view, you will see programming functions. In the panel view, you will see user interface elements like graphs, numeric controls, and boolean buttons.

   Try finding a few common structures or functions in the palette; things like a While Loop or an add function.

7. All the elements from the front panel will have a corresponding block diagram element. These elements, called **controls** and **indicators** are how you pass data between the user interface and the code. The provided block diagram is currently empty because these elements have not yet been placed. Click on the **Unplaced Items** menu to view them. They do not need to be placed now.

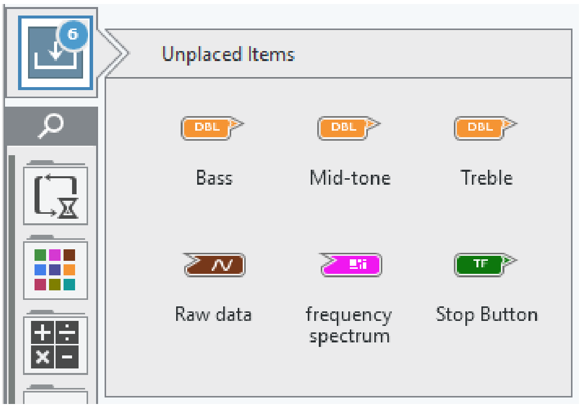


Figure 6-1 The Unplaced Items menu stores elements that are automatically created when placing front panel objects.

8. First, we must create an NI-DAQmx **Task** for our audio measurement. A task can be created interactively using measurement panels - which will be used in this exercise - or programmatically using the NI-DAQmx API. Generally, tasks contain all the configuration information necessary to perform a measurement or generation with DAQ hardware.

9. Click the **plus icon** next to the Audio Exercise.gvi tab at the top of the VI window, and select **Analog Input**, as the image below:

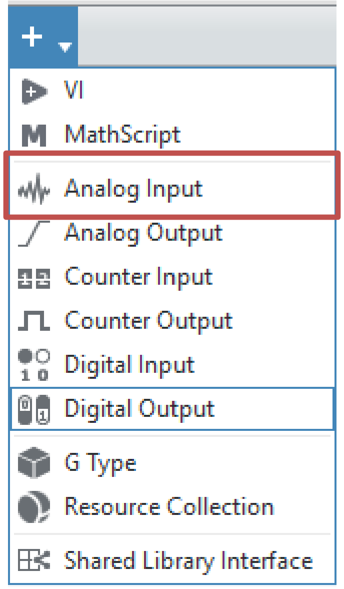
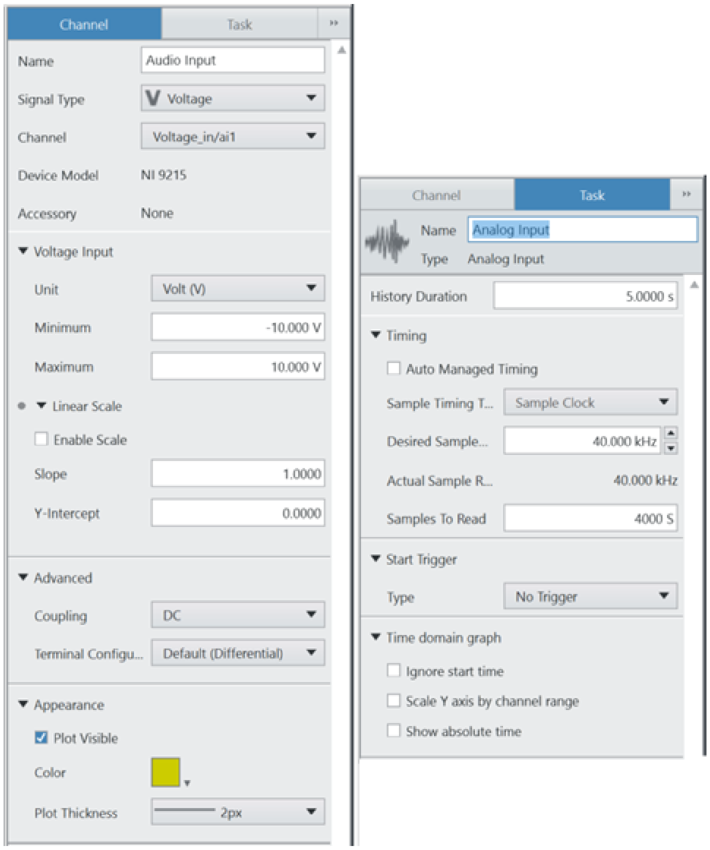


Figure 6-2 Quickly add a new file to your project using the plus icon.

The resulting display is called a Measurement Panel. Measurement panels provide you immediate access to the I/O of your hardware. You can adjust measurement or generation configuration settings in the configuration pane to the right, and observe the measured data in real time in the graph at the center. You can also capture data from this view using the record function at the top left of the panel.

10. Configure the Channel and Task settings in the configuration pane of the measurement panel as shown in Figure 6-3.



Channel | Task

Name: Audio Input

Signal Type: Voltage

Channel: Voltage_in/ai1

Device Model: NI 9215

Accessory: None

Voltage Input

Unit: Volt (V)

Minimum: -10.000 V

Maximum: 10.000 V

Linear Scale

□ Enable Scale

Slope: 1.0000

Y-Intercept: 0.0000

Advanced

Coupling: DC

Terminal Configu...: Default (Differential)

Appearance

☑ Plot Visible

Color: 🟨

Plot Thickness: 2px

Channel | Task

Name: Analog Input

Type: Analog Input

History Duration: 5.0000 s

Timing

□ Auto Managed Timing

Sample Timing T...: Sample Clock

Desired Sample...: 40.000 kHz

Actual Sample R...: 40.000 kHz

Samples To Read: 4000 S

Start Trigger

Type: No Trigger

Time domain graph

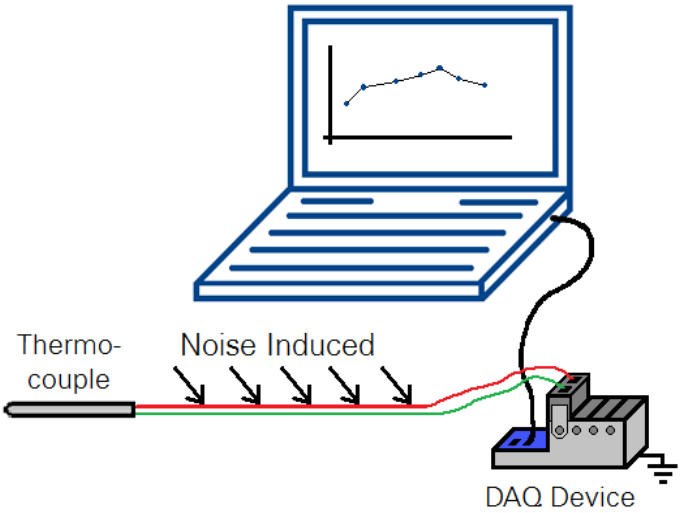
□ Ignore start time

□ Scale Y axis by channel range

□ Show absolute time

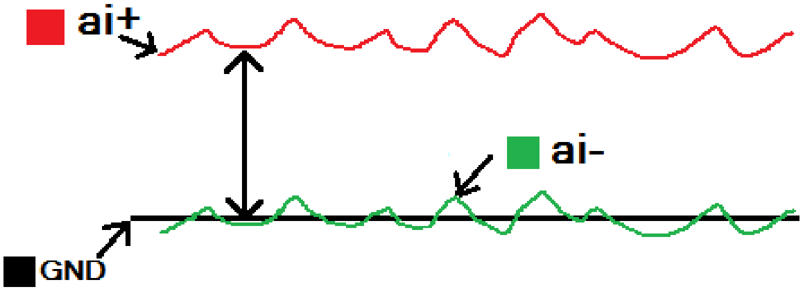
Figure 6-3 The configuration pane contains all of the relevant channel and task settings for the type of task in the open measurement panel.

NOTE: Choosing the Right Input Configuration



Noise and other signals can enter 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 effect.

The terminal input configuration is the easiest way to eliminate a lot of noise. Single-ended measurements (RSE and NRSE) only take the positive sensor signal (ai+) and reference it 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 to both the positive and negative wires.

Always use this configuration if available for sensor acquisition.

11. If it is not running automatically, run the measurement panel by clicking on the run arrow at the top left ▶. Run an MP3 source into the AUDIO IN terminal of the demo box and observe the acquired data – this can be your cell phone or the laptop provided. If using the laptop provided, launch Windows Media Player and use one of the sample songs provided. Be sure the volume on your source device is adequately high to ensure the best possible acquisition.

12. After a few seconds, stop the task by clicking the **stop** button ●.

13. Save the task using **Ctrl + S** or by navigating to **File » Save Analog Input.task**.

We have now successfully configured a measurement task and taken an interactive measurement. To use this measurement task alongside other operations (like output, analysis, logging, etc.) we must incorporate the measurement into our Audio Exercise VI.

14. Switch to the Audio Exercise.gvi tab and open the block diagram by clicking the Diagram button

.

15. Start creating code by dragging **Analog Input.task** from the Project Files onto the block diagram, as shown in Figure 6-4. Note that, along with a constant specifying which task to run, you have also been provided with:
    a. DAQmx Read function
    b. Duration constant
    c. Data indicator.

    This is all the necessary code to run this measurement, however we will need to add to and delete from this code to achieve our desired measurement.

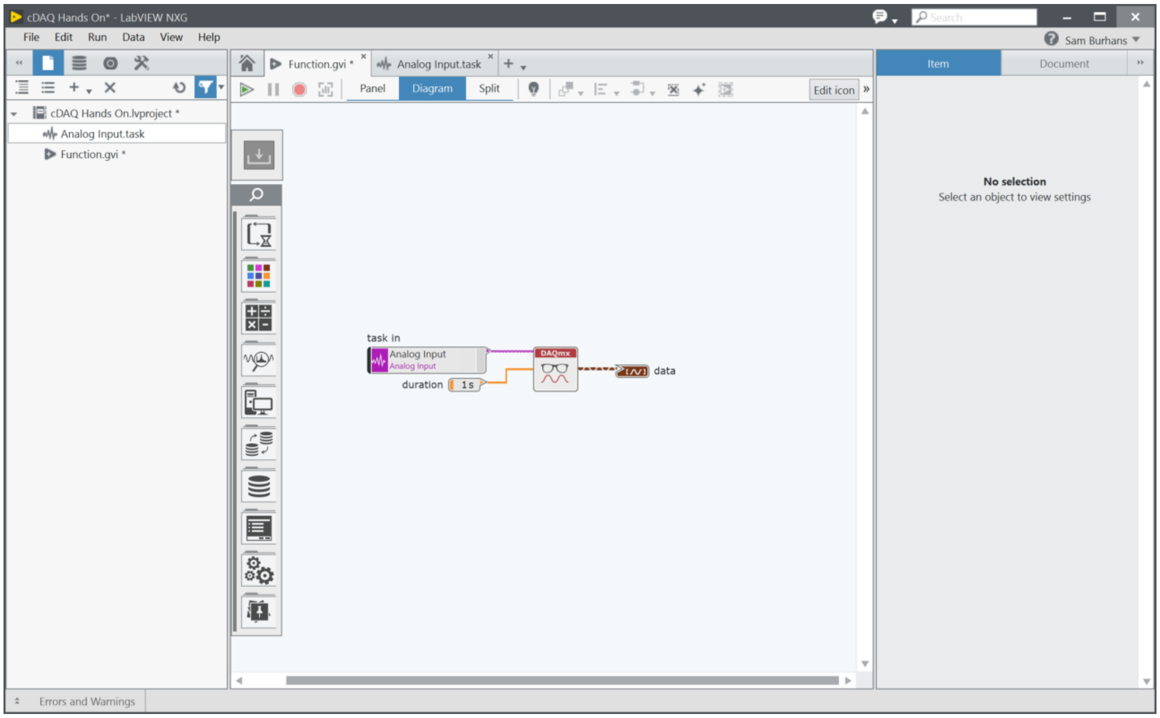


Figure 6-4 Dragging a task onto the block diagram automatically generates all of the code necessary to run that task.

16. The following steps will guide you in building the rest of your DAQmx Analog Input measurement VI:
    a. Select the DAQmx Read function, note that the configuration pane on the right populates with all the settings that can be adjusted for this function.

    b. Ensure the function configuration for DAQmx Read is as follows:
        i. Channel Type = Analog Input
        ii. Data Format = Waveform
        iii. Channel Count = Single Channel
        iv. Sample Count = Multiple Samples
        v. Read By = Time

    c. Move the **task in** control to the left to create some space. You can also hold Ctrl and click and drag the mouse on the block diagram to dynamically create and remove space on the block diagram.

    d. Navigate to the palette to find the While Loop structure inside the Program Flow sub-palette.

    e. If one is not created automatically, create a while loop: select **while loop** from the palette, then click and drag around the code you would like to iterate during the loop's operation. In this case, drag the loop around the DAQmx read function and its inputs and outputs, leaving the task input outside.

    f. Once the measurement operation completes and we exit the while loop, we must clear and dispose of any references to hardware to free them for use by other processes or tasks. To do so, navigate or search the palette for the DAQmx Clear Task function (**Hardware Interfaces » NI-DAQmx**), and place it to the right of the while loop.

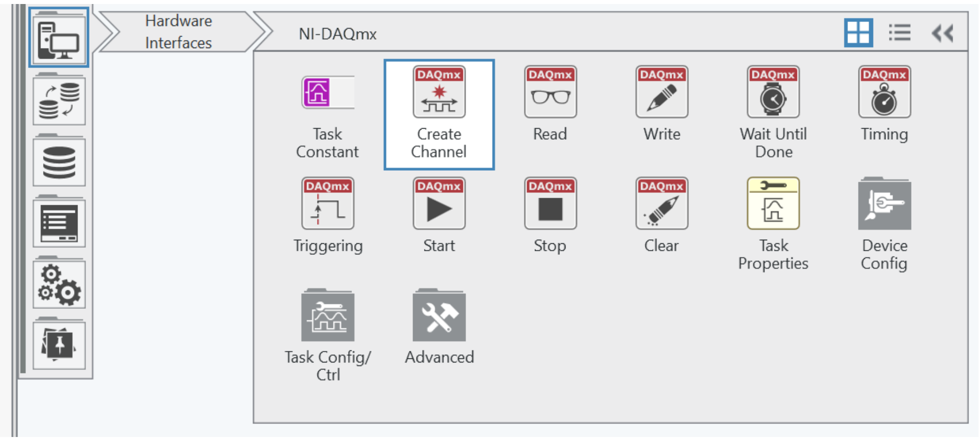


Figure 6-5 Find DAQmx functions under the Hardware Interfaces section of the palette.

g. Properly informing your user of errors generated during operation is important, but should be done in a controlled and deliberate manner. After clearing DAQmx tasks, we will combine any error wires, and display any errors to the user using two functions: **Retain First Error** and **Display Error**. Navigate or search the palette for these functions and place them to the right of DAQmx Clear Task.

h. Now that all functions have been placed, wire them together using the purple Task and yellow Error wires through the Task In/Out and Error In/Out terminals, respectively. This ensures that errors and task configuration information propagate from function to function. Your code should now look like Figure 6-6.

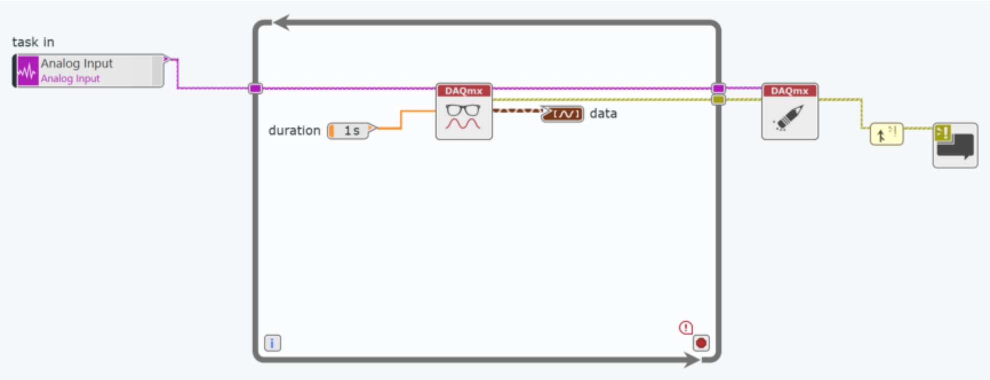


Figure 6-6 DAQmx task code will always follow the same pattern of create/configure channel information, start, acquire or generate, clear, and close.

17. Attempt to run your newly created program. Notice that the run arrow is broken (). LabVIEW operates with an always-on compiler that is constantly compiling your block diagram code. The run icon informs you as to whether there is currently an error preventing the code from compiling properly to run. Clicking on the broken run arrow will open the Errors and Warnings pane and inform you of potential issues.
    a. Click the broken run arrow

b. Note the error in this case: “The required input terminal is unwired.” This means a required input to one of the place functions is not properly connected. In this case, the problem is the while loop’s condition terminal, which dictates when the while loop is to stop iterating. In its currently state, the while loop would run infinitely. To correct this, we should create a stop button to allow the user to stop the code from operating.

c. Double-click on the error to highlight the problem area on the block diagram.

d. Open the **Unplaced Items** menu discussed previously, and retrieve the Stop Button control that has been provided. Place this control near the conditional terminal of the while loop.

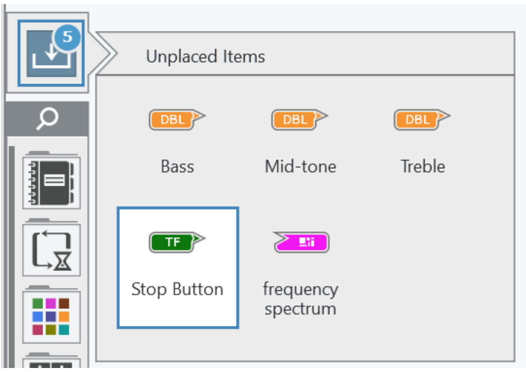


Figure 6-6 The Unplaced Items menu stores controls and indicators you have yet to place on the panel or diagram.

e. If our DAQmx analog input task has an error, it would be ideal if the loop would also stop and notify the user of the error. This can easily be accomplished by adding some simple Boolean logic.

f. From the palette, find the **OR** function under **Data Types » Boolean**. Place this function near the condition terminal.

g. Delete the wire between the stop control and the condition terminal. Wire the stop control into the bottom of the newly placed OR function.

h. Next, create a branch from the error wire coming out of the DAQmx Read function (right click on the wire and select **Create branch**). Connect the error wire branch to the top input of the OR function.

i. The code you just wrote (pictured below) will automatically command the while loop to stop iterating in the event of an error from the DAQmx function or the user clicking the stop button.

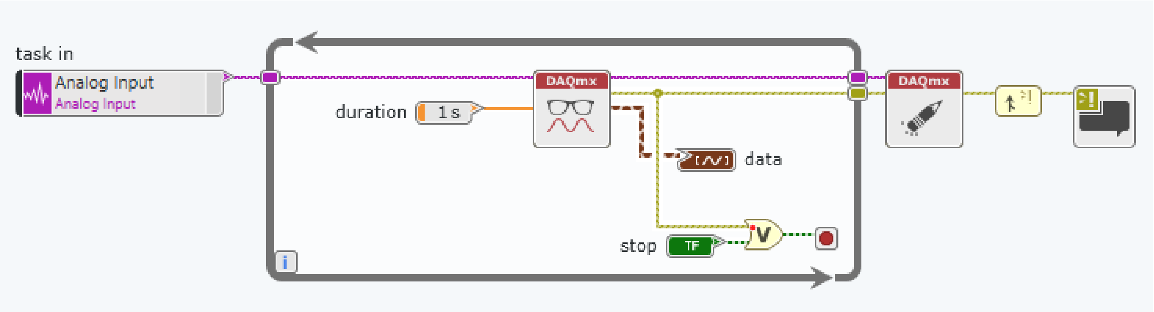


Figure 6-7 An OR function can be used to command a while loop to stop in the event of an error or a user pressing the stop button on the front panel.

18. To ensure that the DAQmx Read VI receives the Task and Error values from one iteration to the next, you will need to use shift registers within your while loop.

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

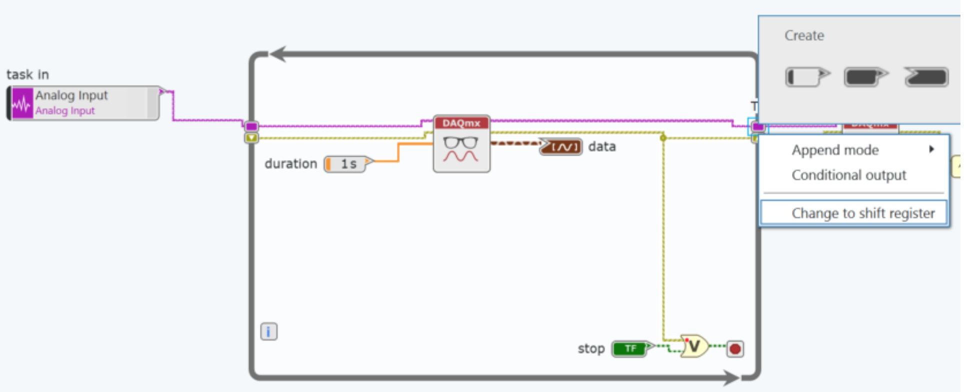


Figure 6-8 Tunnels on the border of a while loops can be changed to a shift register to pass data from one iteration of the loop to the next.

    b. Repeat for error wire tunnels. This will create a new input terminal on the left side of the while loop that you will need to wire to the error input of DAQmx Read (See Figure 6-8).

19. Switch to the front panel of the VI by pressing Ctrl + E or using the Panel button at the top of the VI window. As you create controls and indicators on the block diagram, their corresponding front panel elements are stored in the **Unplaced items** bin, just above the palette, as with the block diagram. A blue circle with a number indicated how many unplaced items are available.

20. From the **unplaced items** bin, click and place the **data** indicator (a graph) on the front panel above the provided graph – you may need to resize the graph to fit.

21. Run the VI by clicking on the run arrow at the top left ▷. Run an MP3 source (your cell phone, or the laptop) into the AUDIO IN terminal of the demo box and observe the acquired data on your front panel indicator. You should see data like in Figure 6-9.

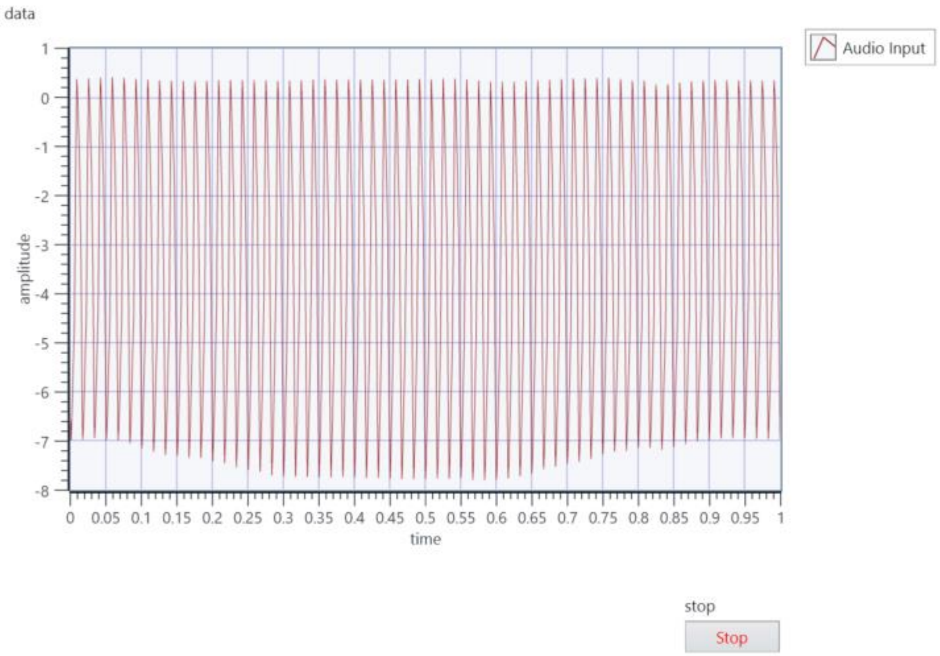


Figure 6-9 Front panel controls and indicators give the user a means to view data and interact with the program.

22. Use the stop button to stop the code when you are done. Save the VI.

23. Return to the block diagram using Ctrl+E.

24. Make some space to the left of the while loop by holding Ctrl, clicking and dragging your cursor in the area you'd like to expand, as in Figure 6-10.

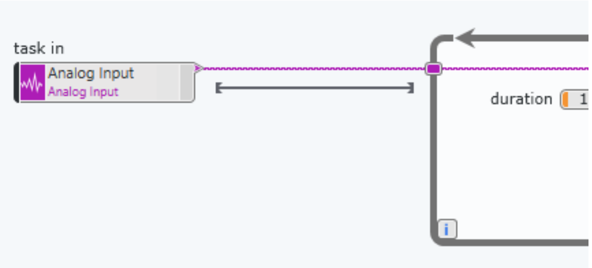


Figure 6-10 You can dynamically create space on a block diagram by holding control while clicking and dragging your cursor.

25. Navigate to the DAQmx API palette (**hardware interface » NI-DAQmx » Task Config/Ctrl** and select the **DAQmx Configure Logging** VI. Place it to the right of DAQmx task constant, and wire the Task and Error wires through the function (note that you will need to break/delete the existing wires to do so). Your code should look like:

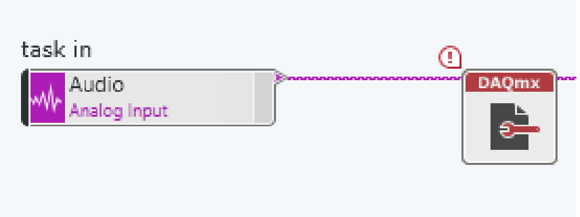
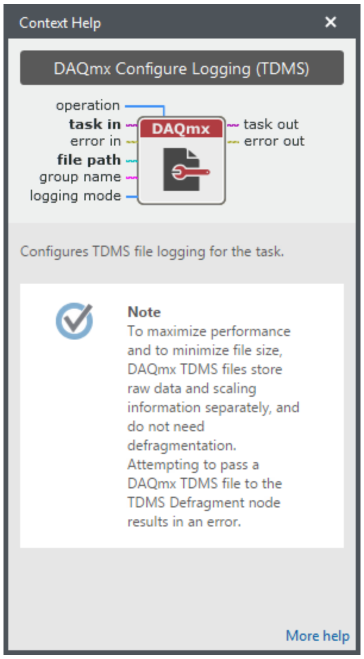


Figure 6-11 DAQmx Configure Logging is the easiest method to incorporate logging into your DAQ task.

26. Note that the configure logging VI still has an exclamation mark icon (!) on the top left. This means there is a required input to the function that is not currently wired. To determine which input is required, press Ctrl+H to open context help, then click on the DAQmx Configure Logging function. The context window will provide additional information about panel and diagram elements as you hover over them, as in Figure 6-12. The **bold** terminals are required, and the code will not be able to compile without creating and wiring a control or constant to these terminals.



Configures TDMS file logging for the task.

Note
To maximize performance and to minimize file size, DAQmx TDMS files store raw data and scaling information separately, and do not need defragmentation. Attempting to pass a DAQmx TDMS file to the TDMS Defragment node results in an error.

Figure 6-12 Context Help can be toggled using the shortcut Ctrl + H, and provides detail on the inputs, outputs, and functionality of each element of the program.

27. In this case, File Path is required. There are two options to address this:
    a. Create a file path constant that will only be visible on the block diagram. This is a good option if you will consistently write to the same path every time you run your code.
    b. Create a file path control that will be visible on the front panel. This is a good option if you would like to give your end user (or yourself) the ability to change the file path at run time without having to look at the code.
28. For this exercise, we will use a control. Right click on the **File Path** input terminal of DAQmx Configure Logging, and select **Create control**, as in Figure 6-13.

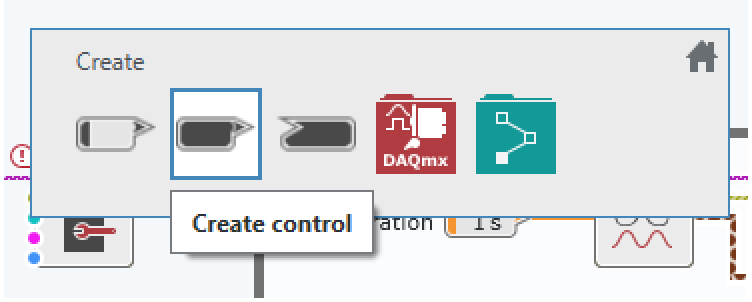


Figure 6-13 Create controls, constants, and indicators by right-clicking on the terminals of a block diagram element.

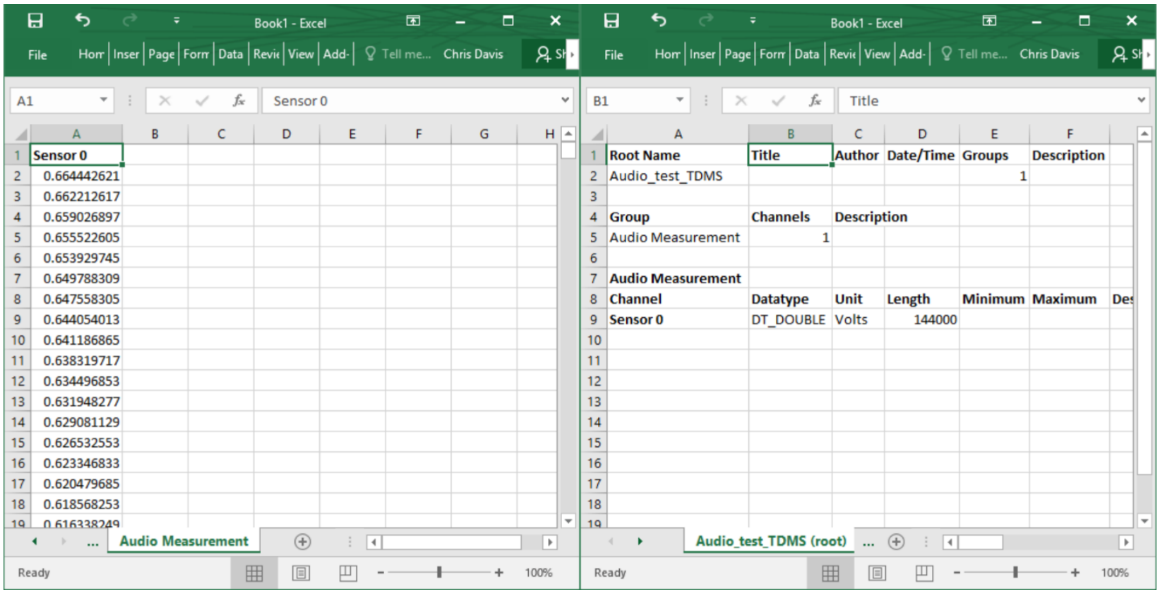
29. Return to the front panel (Ctrl+E). The control you just created should be in the Unplace Items palette. Retrieve it and place it to the right of the chart.

30. Populate the new control with a file path where you would like to log your audio measurement data. Enter the following string:
    C:\Seminars\cDAQ + LabVIEW NXG HO\Exercises\Audio\Audio_Data.tdms

    Optional: You can also browse to your own path by pressing (…). If you do not have a TDMS file to point to, click the address bar to copy the path of the file, manually paste the path into the **file path** control. Then, manually add **\Audio_Data.tdms**.

31. Use the provided auxiliary cable to provide some audio input to your DAQ hardware. Run your program for a few seconds, then stop it.

32. Navigate to the saved data file and double-click it to open the file in Excel. If Excel is not the default file option, right click the file and choose **Open with > Excel Importer**. Observe the saved data format, as in Figure 6-14.



A1 | Sensor 0

	A	B	C	D	E	F	G	H
1	**Sensor 0**							
2	0.664442621							
3	0.662212617							
4	0.659026897							
5	0.655522605							
6	0.653929745							
7	0.649788309							
8	0.647558305							
9	0.644054013							
10	0.641186865							
11	0.638319717							
12	0.634496853							
13	0.631948277							
14	0.629081129							
15	0.626532553							
16	0.623346833							
17	0.620479685							
18	0.618568253							
19	0.616338249							

B1 | Title

	A	B	C	D	E	F	
1	**Root Name**	**Title**	**Author**	**Date/Time**	**Groups**	**Description**	
2	Audio_test_TDMS				1		
3							
4	**Group**	**Channels**	**Description**				
5	Audio Measurement	1					
6							
7	**Audio Measurement**						
8	**Channel**	**Datatype**	**Unit**	**Length**	**Minimum**	**Maximum**	**Des**
9	**Sensor 0**	DT_DOUBLE	Volts	144000			
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							

Figure 6-14 The TDMS file format is specifically designed for storing technical data and includes all relevant metadata that might be useful for a DAQ measurement application.

33. The DAQmx Configure Logging function uses the TDMS file format for logging data with metadata efficiently. TDMS was specifically designed by National Instruments for effectively logging measurement data in a fast, scalable way. There is also a free Excel Importer tool for opening TDMS data files in Microsoft Excel, even without any other NI Software installed. To learn more about the format, review the whitepaper at www.ni.com/tdms.

34. When finished, close the data file, save your VI, and move on to the next section.

Exercise 2: Filtering and FFT

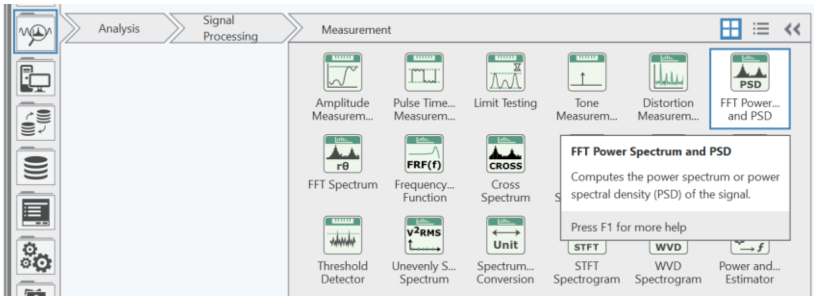
Goals

- Add a Fast Fourier Transform (FFT) to the audio signal to view the signal in the frequency spectrum
- Interactively configure a low-pass filter to remove the high-frequencies from the audio signal. Use that configuration in your code.
- 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 the signal processing

Part A

1. Before placing the signal processing functions, first create empty space within the while loop for these functions using Ctrl + click-and-drag, as before.

2. Open the Functions Palette and navigate to **Analysis » Signal Processing » Measurement** and select **FFT Power Spectrum and PSD**. Place this VI to the right and below the DAQmx Read VI, as in Figure 6-16.



FFT Power Spectrum and PSD

Computes the power spectrum or power spectral density (PSD) of the signal.

Press F1 for more help

Figure 6-15: As you explore palettes, notice the numerous analysis functions available in LabVIEW

a. When placed, this VI will pop-up a configuration window to specify the type of data and desired analysis. Leave the default settings: Power Spectrum with Continuous data.

b. Wire the DAQmx Read **data** output into the **signal** input of the Power Spectrum. You can right click and select create branch to branch the existing data wire, or save time by holding Ctrl and clicking the wire to create a branch.

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

d. Open the **Unplaced Items** palette menu and retrieve the **Frequency Spectrum** indicator. Place it just after the Power Spectrum/PSD. Wire the **Power Spectrum for 1 Chan** output into the input of the provided Frequency Spectrum graph.

e. Wire the Error out of the DAQmx Read into the Error In of the Power Spectrum function, and then connect the Error Output into the Shift Register and the OR function.

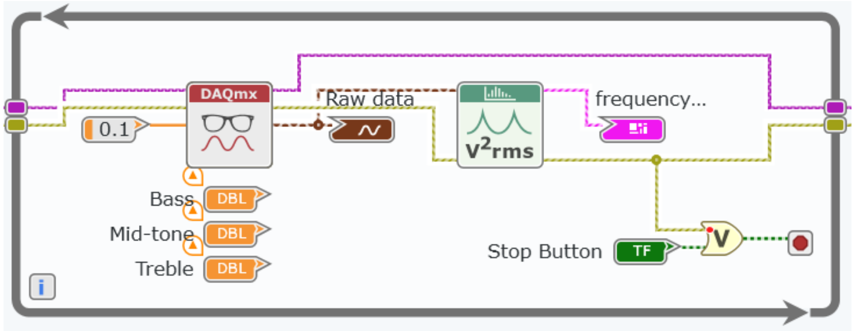
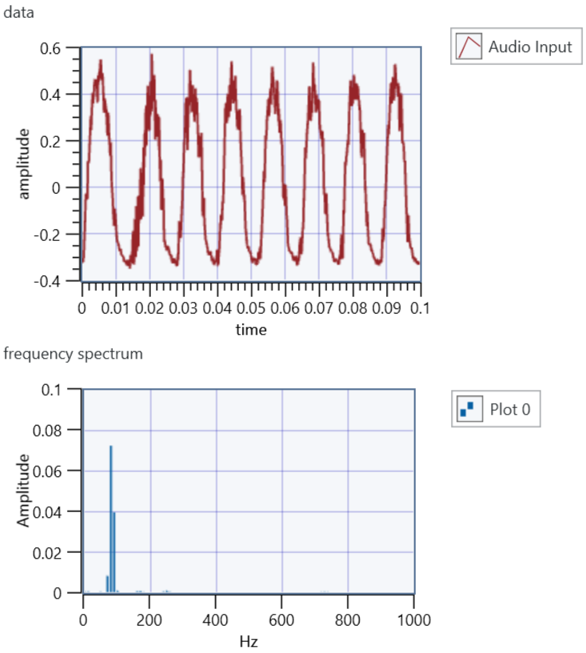


Figure 6-16 FFT analysis functions accept time-domain measurement data and return frequency-domain analysis

3. Save the VI.

4. Run the VI to observe the Frequency spectrum of the audio signal, shown below:



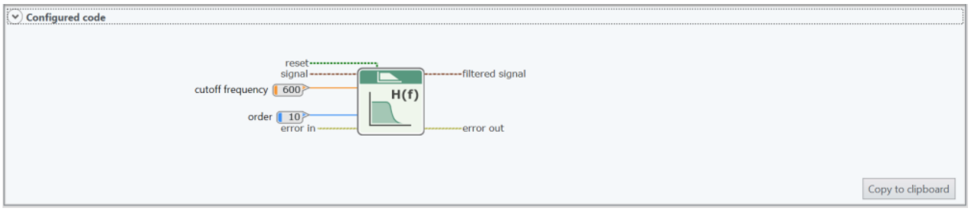
Part B

1. Click the **Captured Data** icon In the Navigation Pane. Import the TDMS file created previously by clicking the **import** button and navigating to your saved file.

2. Once imported, double-click on the file to open the data set in the Data Viewer. The Data Viewer allows you to interact with the dataset and capture subsets of data that may contain events you are interested in exploring further.

3. In the Data Viewer, select **Filter** from the Interactive Analysis section of the Configuration Pane. This creates a filter analysis pane with the data set you were previously viewing. Here you can play with different filtering settings and view the resulting data set in real time. Take a minute to explore this functionality. When finished, configure a **Lowpass Butterworth** filter with the following settings:
   a. Cutoff frequency = 600
   b. Order = 10

4. Once configured, click **Configured Code** at the bottom of the analysis pane to show the block diagram code necessary to implement the filter you've just configured. From here, click the **Copy to Clipboard** button.



5. Return to your exercise block diagram, and paste the copied code. Position the pasted code between the DAQmx Read and Power Spectrum functions.

6. Wire accordingly for error and signal wires. Expand the while loop if you need more space. The resulting diagram should look like Figure 6-17.

7. Switch to the front panel and run the VI. Observe how the filter alters the frequency spectrum of your acquired data. Save the VI.

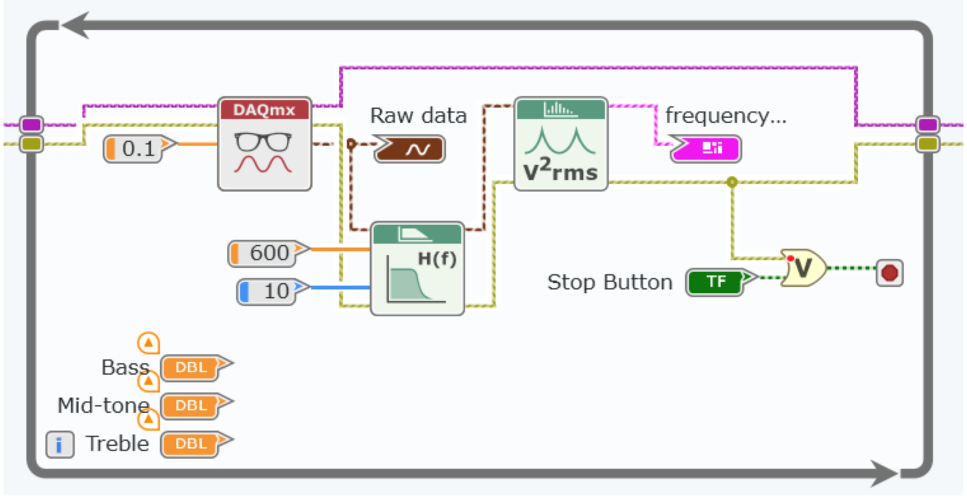


Figure 6-17 Applying a filter to your acquired data can focus the resulting data set on a particular spectrum of interest.

Part C

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

   a. Select the filter and its constant inputs by holding shift and clicking each item. Press **Ctrl + C** to copy the functions, and **Ctrl + V** to paste them lower in the while loop.

   b. Select the first copy of the filter code, and use the configuration pane to change the type to **bandpass**. Select the second, changing the type to **highpass**.

   c. 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.

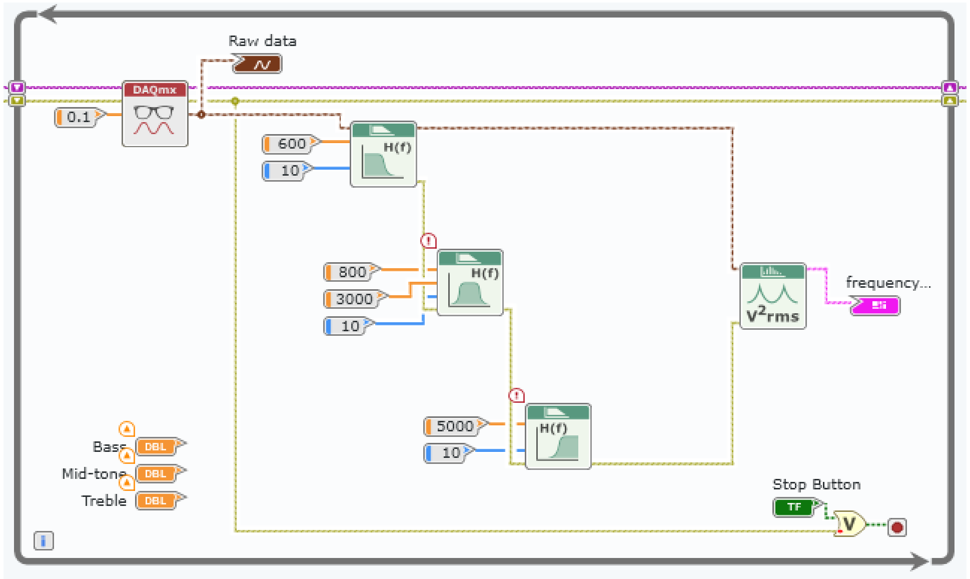


Figure 6-18 The error wire specifies the order of operations, following the principles of dataflow.

   d. The bandpass cutoff frequency must always be greater than zero and less than the upper cutoff frequency. Set Low cutoff frequency to **800**, add a new constant for the high cutoff frequency and set it to **3000**.

   e. For the highpass filter, set the cutoff frequency to **5000**.

   f. 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 from the data output from the DAQmx Read VI and wire this to the signal in terminal of each filter.

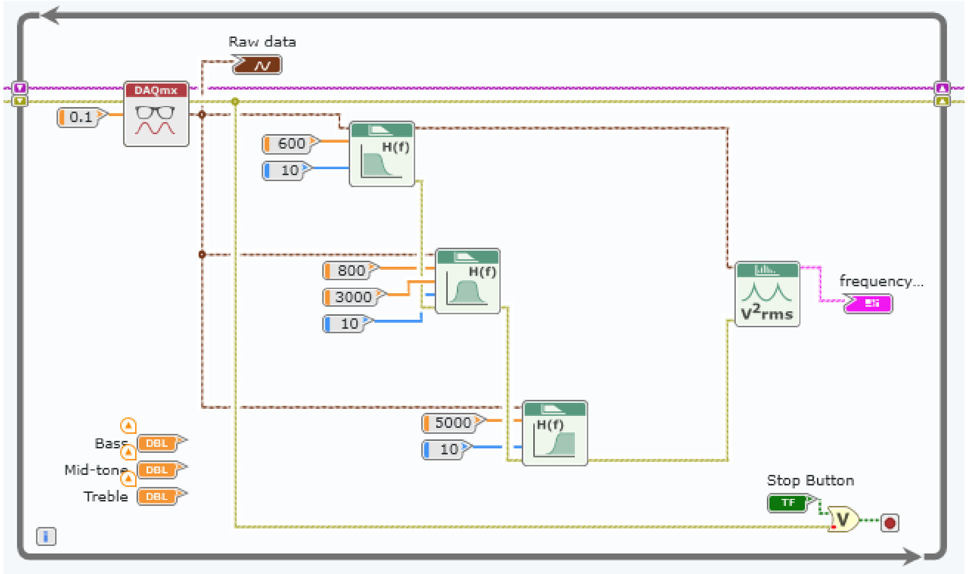


Figure 6-19 Each filter will operate on the same original data set acquired by the hardware.

2. 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 the **Waveform Properties** function for each of the three waveform clusters to extract the array of data, **Y**.

      i. Navigate to **Data Types » Waveform » Analog Waveform** on the Functions Palette.

      ii. Place 3 **Waveform Properties** functions on the diagram and wire the waveform output from each filter into the **Y** input. Since we only care about the array of data, you can shrink each waveform properties function down to a single element, as in Figure 6-20.

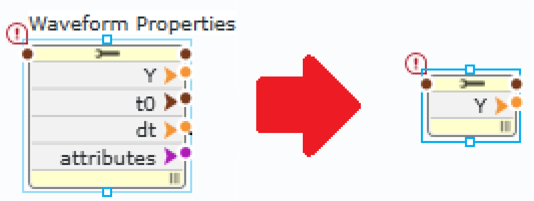


Figure 6-20 Resizing functions like Waveform Properties to display only the properties of interest keeps your code clean and readable.

   b. Place a multiply function for each array from the **Math » Numeric** functions Palette and connect each one of the three arrays to the top input of the function.

c. Drag each of the provided control terminals for Bass, Mid-tone, and Treble to the space before each multiply function, and wire them to the open input.

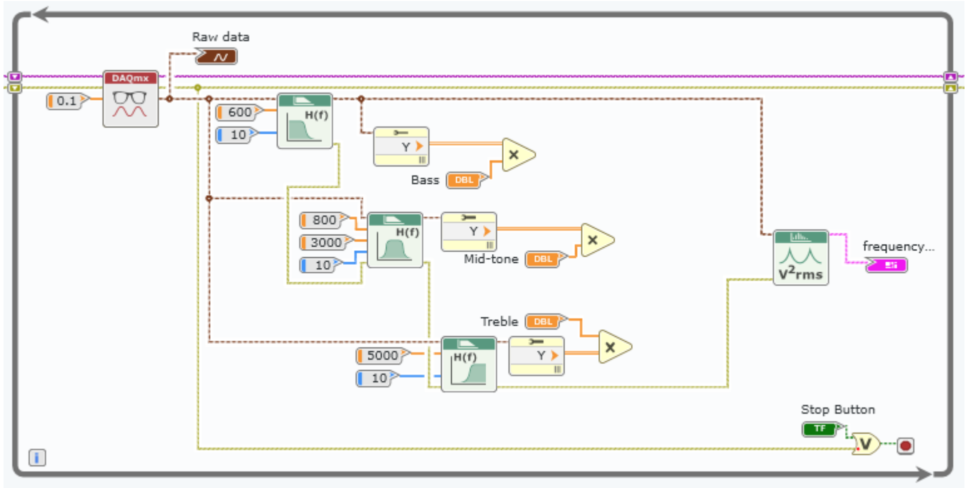


Figure 6-21 Adding amplification code provides the ability to amplify or attenuate each section of the frequency spectrum.

3. To add the arrays back together, use an **Add** function as you did to add the Boolean functions together for the Conditional terminal.

   a. Place it from the **Math » 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 **Waveform Properties** function and branch from the previous waveform to use as a reference for the **waveform in** input on the top of the function. You can copy one of the previously placed waveform properties functions, or add a new one from the palette. Either way, the default configuration is to read, not write, properties.

   d. Change the **Waveform Properties** function to write the **Y** data array by selecting the function and using the Configuration Pane to **Set All to Write**.

   e. Wire the output of the Add into the **Y** terminal of the Build Waveform.

   f. Delete the existing data input to the **Power Spectrum** VI, and wire the output of the waveform properties node in it's place. The Block Diagram code within the While Loop should appear as follows.

5

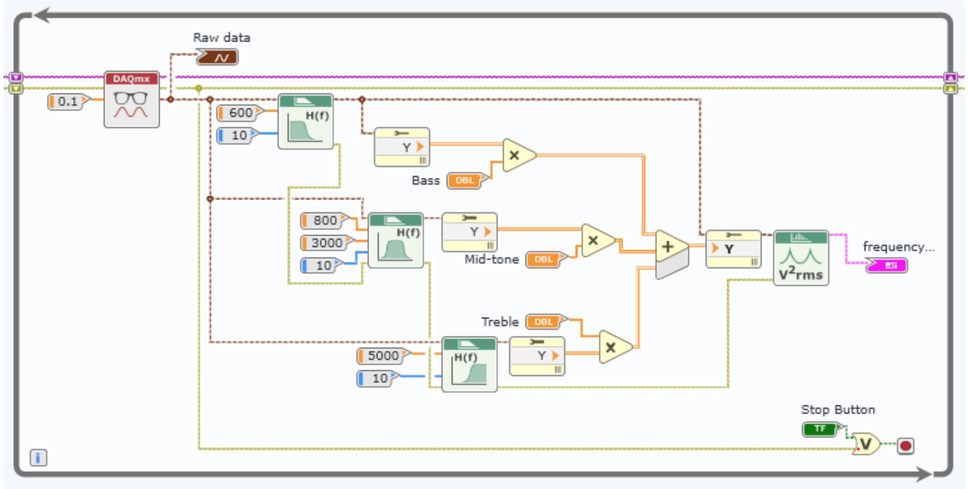


Figure 6-22 Rebuild the complete waveform by summing the bass, mid-tone, and treble component

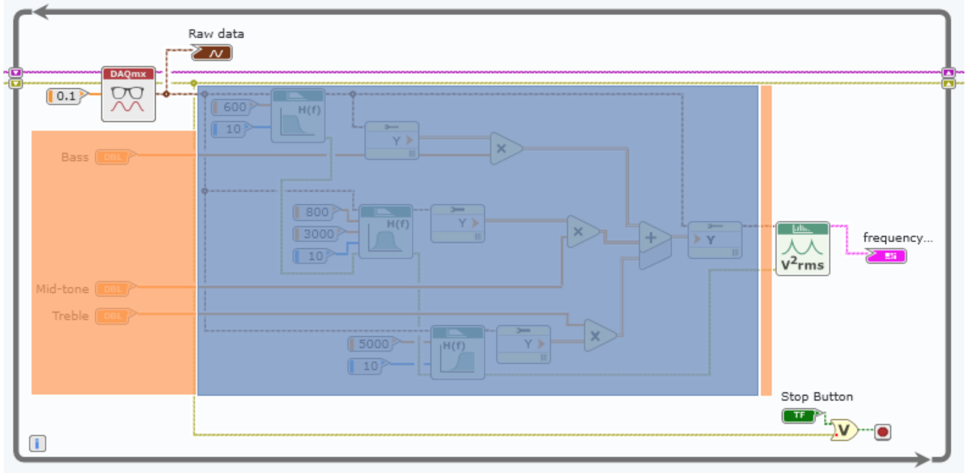
4. Save the VI.

5. 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 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:

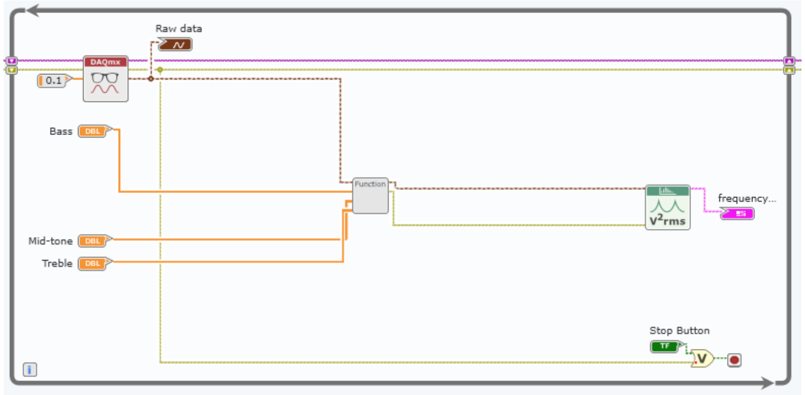


The blue section will be contained within the subVI, and the wires that are enveloped in the orange section will be inputs and outputs to the subVI function. If no terminals exist for either inputs or outputs, they will be created along with the subVI.

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 from Selection**.

   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 in a new tab.

   a. Note that all the necessary controls and indicators to reflect the function's inputs and outputs have been created for you, but they are in the Unplaced Items bin. You may wish to place them on the front panel, but this is not required for the subVI to function. A good rule of thumb is to place inputs on the left of the panel and outputs on the right.

   b. Next to the Panel and Diagram selector, you can click on Icon to create a custom icon for your subVI.

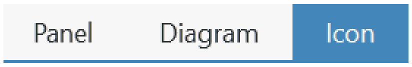


Figure 6-23 Editing the icon ensures that the functionality of your subVI is clear and that the connection terminals are efficiently located.

   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. The configuration pane and palette contain a variety of tools to customize your subVI icons with icons, color, pictures, and more.

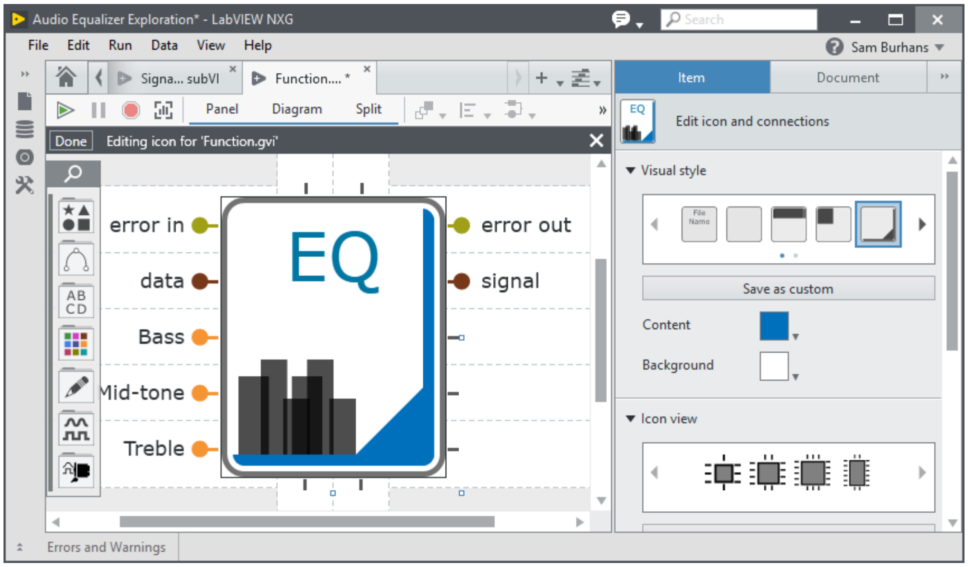


Figure 6-24 A good icon makes a subVIs functionality apparent in the top-level VI in which it is used.

d. See above for an example of an icon that you could use. Feel free to create your own.

e. Depending on how you selected the code to go into your subVI, you may wish to adjust the wiring configuration. 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. Figure 6-24 shows a good mapping of terminals for this exercise – adjust your icon to match.

f. Select **Done** once you are satisfied with the terminal mapping.

g. By default, the subVI is saved as **Function.gvi** in the project files view of the Navigation Pane. You may wish to rename this file to **Signal Processing subVI** by right-clicking the file name and selecting **rename**.

h. Close the subVI.

5. Open the Block Diagram of your main code and notice that the subVI has replaced the previously written code, and your icon is displayed. Your run arrow should be solid if all terminals of the subVI are properly mapped. Notice how much cleaner your block diagram looks, you may want to user Ctrl+click+drag to remove some of the free space.

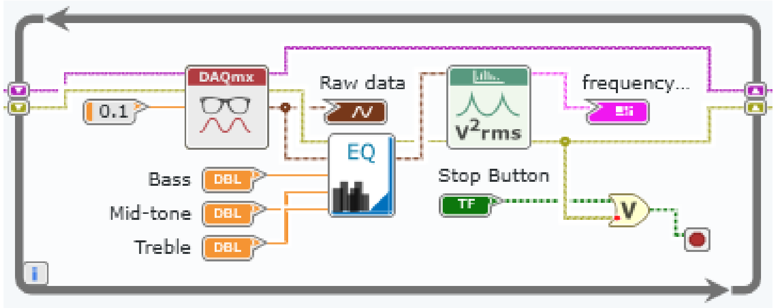


Figure 6-25 Proper use of subVIs can greatly simplify your top-level code.

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

Exercise 3: Output Processed Signal

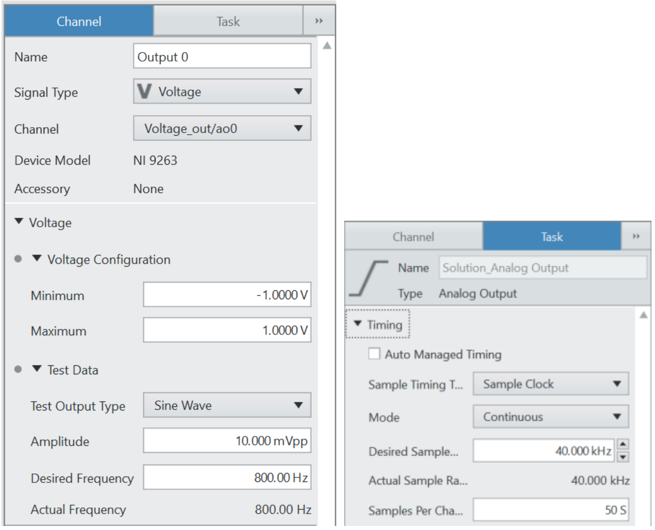
Goals

- Create an Analog Output Task using measurement panels 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 Analog Output measurement panel by selecting the **Add File…** icon in the tab toolbar and selecting **Analog Output**. This will open an analog output measurement panel.

2. A single output channel should be created for you. Configure this channel and task using the configuration pane to the right. Set the configuration as shown in Figure 6-26.



Name: Output 0
Signal Type: Voltage
Channel: Voltage_out/ao0
Device Model: NI 9263
Accessory: None
▼ Voltage
▼ Voltage Configuration
Minimum: -1.0000 V
Maximum: 1.0000 V
▼ Test Data
Test Output Type: Sine Wave
Amplitude: 10.000 mVpp
Desired Frequency: 800.00 Hz
Actual Frequency: 800.00 Hz

Name: Solution_Analog Output
Type: Analog Output
▼ Timing
□ Auto Managed Timing
Sample Timing T...: Sample Clock
Mode: Continuous
Desired Sample...: 40.000 kHz
Actual Sample Ra...: 40.000 kHz
Samples Per Cha...: 50 S

Figure 6-26 The frequency of a sine wave test signal in an analog output measurement panel is dictated by the desired sample rate and output buffer size.

3. Click the run button to run the task. This should play an 800 Hz tone through the speaker of your demo kit until the stop button is pressed.

Part B

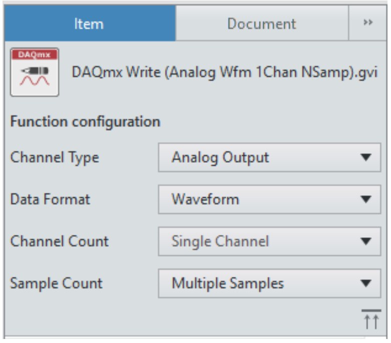
1. To update the output waveform while running, you can change the output buffer size in the Task configuration, but this isn't the most user-friendly experience. To address this, we can use the task to write a new VI with a proper user interface.

2. Save the analog output task to your project files, and create a new VI using the plus icon in the tab menu.

3. Switch to the block diagram, then drag and drop **Analog Output.task** from the project files on to the block diagram to automatically generate the output code.

4. As-is, the code is missing a few important pieces. Perform the following steps:

    a. For now, we'll only be using a single output channel. Change the configuration of the DAQmx Write VI to Single Channel in the configuration pane, as shown below:



DAQmx Write (Analog Wfm 1Chan NSamp).gvi

Function configuration

Channel Type	Analog Output
Data Format	Waveform
Channel Count	Single Channel
Sample Count	Multiple Samples

    b. Delete the **data** input constant and the **For Loop** and all code within it.

    c. We now need to generate the sine wave for our hardware to output. To do this, navigate the palette to **Analysis » Signal Processing » Generation**, and select the Wave Generator function.

    d. Create constants for **amplitude** and **sample rate**, and set their values as follows:
        i. Amplitude = 0.005
        ii. Sample rate = 40,000

    e. Branch the sample rate wire and connect it to the **samples** input as well.

    f. Create a control for the frequency input.

    g. Wire the **sine wave** output into the DAQmx Write **data** input terminal.

    h. Finally, place a **while loop** around the DAQmx Write function. Create a stop button on the input of the conditional terminal (right-click and create control).

5. Switch to the front panel and place all the controls and indicators from the **Unplaced Items** menu. Right click the frequency control and select **replace**. Choose a more user-friendly graphical control to adjust the frequency of the output, like a knob or slider. Double-click

the minimum and maximum of the scale of this new control and make the range from 1 to 20k. Your front panel and block diagram should look like Figure 6-27.

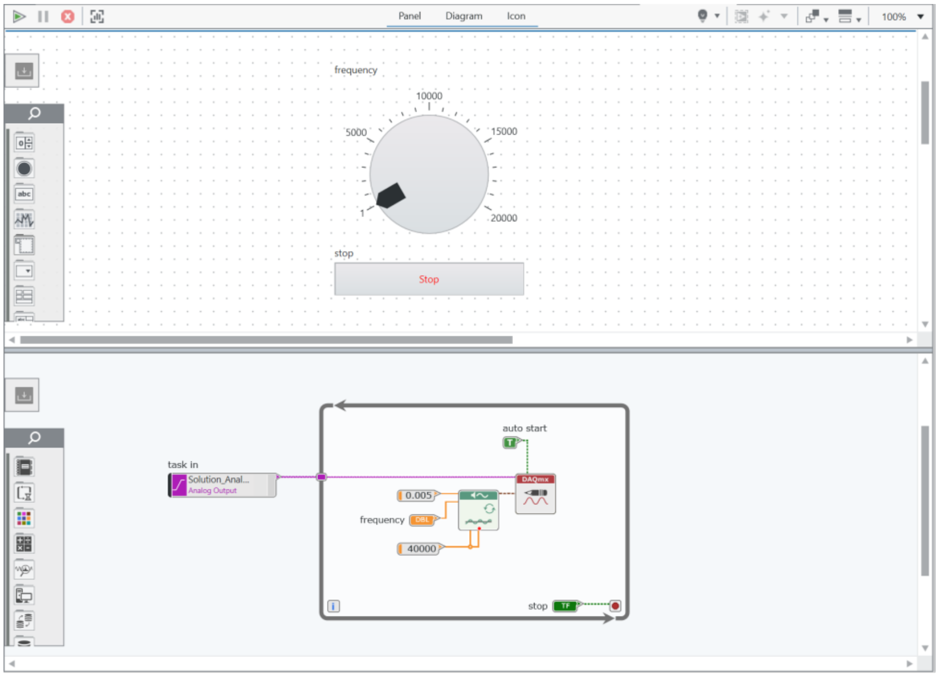


Figure 6-27 The Split view allows you to see both the front panel and block diagram at the same time. To access Split view, click the "Split Orientation" button:

6. Save the VI (you may choose to rename it in the Project Files pane if you wish). Run the VI and adjust the frequency control. Changing the frequency control should correlate to a change in the pitch of the tone being output from the speaker of your demo kit.

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.

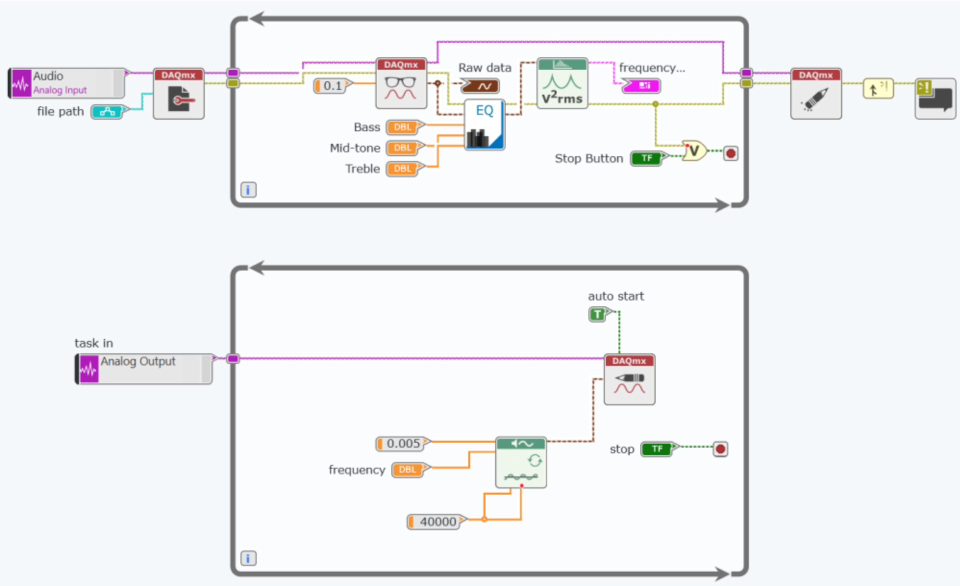


Figure 6-28 Analog output code added to the analog input block diagram.

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. To remove a loop without deleting the code contained within it, right click on the loop border and select **Delete 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 in Figure 6-29:

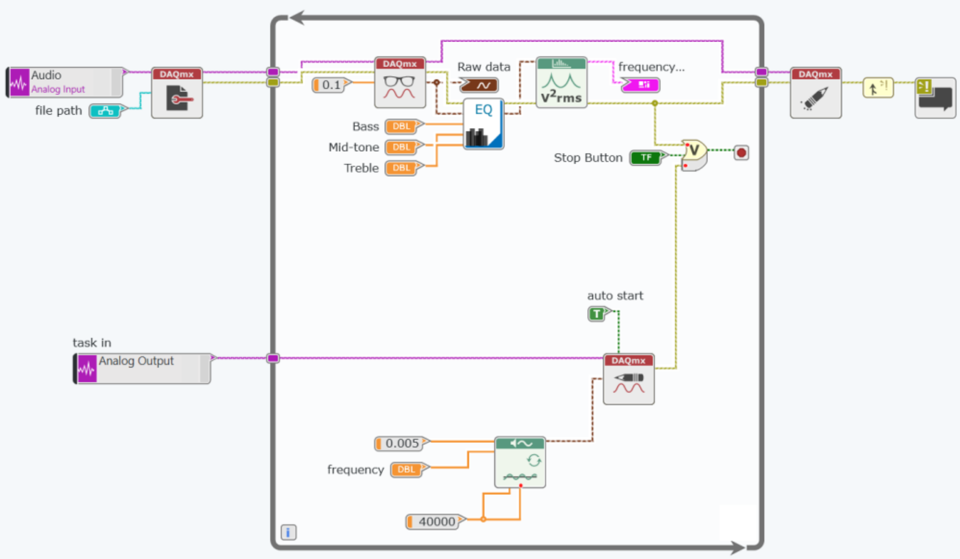


Figure 6-29 Expand the OR function to stop the while loop in the event of an error from either task or a user pressing the stop button.

3. Remove the Function Generator VI and all 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.

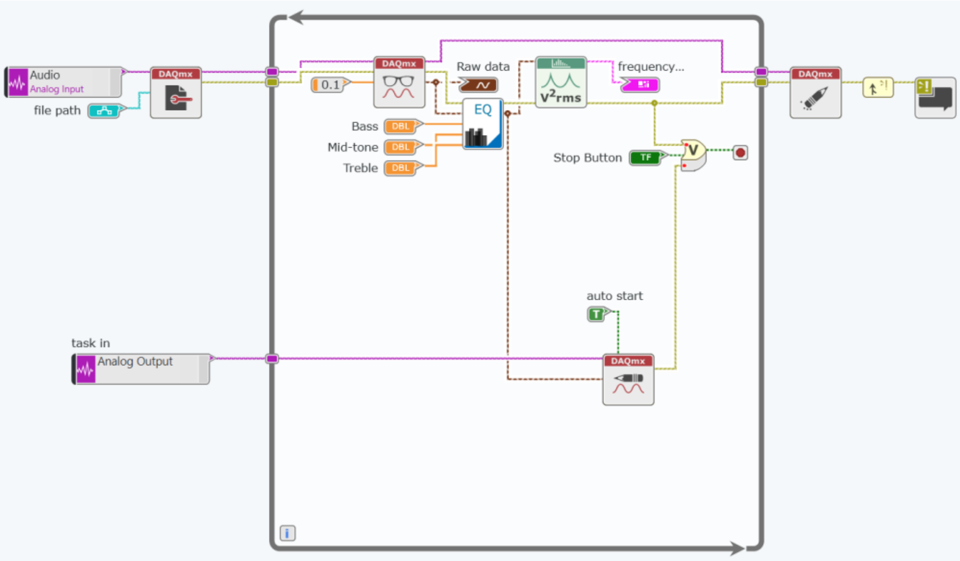


Figure 6-30 Wiring the output of the DAQmx Read to the input of the DAQmx Write means our acquired data will be looped back out to the connected speaker.

5. Now that there are multiple tasks in a single VI, it is best practice to explicitly handle your hardware references and error wires. Make the following adjustments to the code:

   a. Place a DAQmx Clear Task VI to the right of the while loop, below the existing DAQmx Clear Task.

   b. Wire the task wire from DAQmx Write to DAQmx Clear Task.

   c. Wire the error wires from DAQmx Write, through DAQmx Clear Task, and terminating in the Retain First Error function. This ensures errors from either task are properly elevated to the user.

   d. Your resulting code should look similar to Figure 6-31.

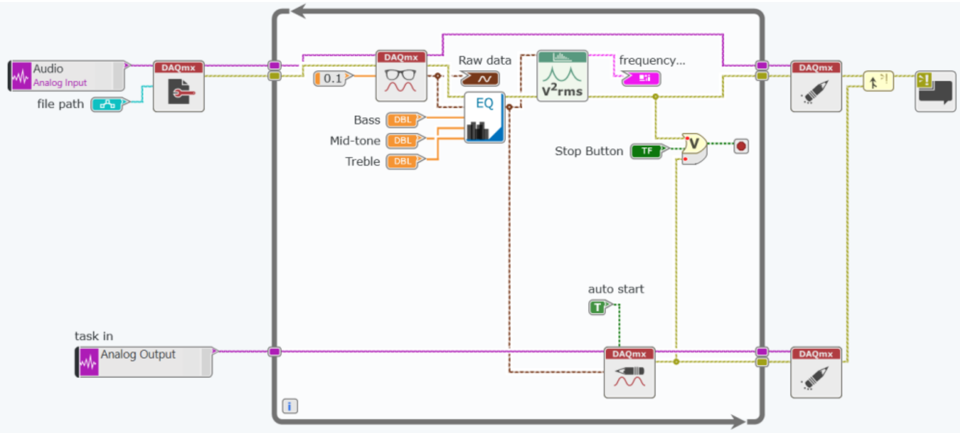


Figure 6-31 Proper task reference and error handling is important in any DAQ application.

6. Save the VI.

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

<End of Exercise>

Chapter 7 Vibration Measurements and Analog Output Control

Goals

- Write a program to control the speed of a fan and acquire data from accelerometers attached to the fan.
- Add code analyze the vibration signal in LabVIEW NXG
- Understand these key concepts:
  - Accelerometer measurements
  - Measurement Panels
  - Creating NI-DAQmx Tasks Interactively
  - Creating NI-DAQmx tasks using the API
  - Combining measurement and control tasks
  - Processing data with LabVIEW NXG

Part A

Measuring Vibration from an Accelerometer

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.

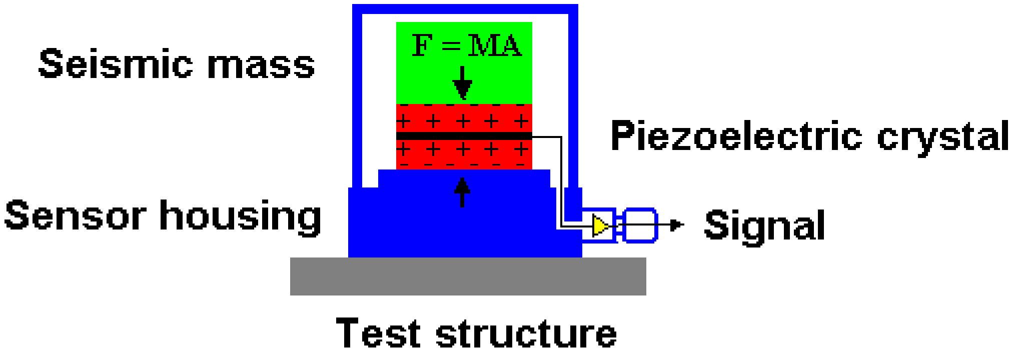


Figure 7-1 An accelerometer uses a seismic mass to create a small electrical signal based on piezoelectric generation.

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 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 C Series Sound and Vibration Input Module 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 can 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. Confirm that the BNC cables are connected to the Sound and Vibration Signal Simulator as follows:

    a) Ch. 0 → X Acceleration

    b) Ch.1 → Y Acceleration

    c) Ch. 2 → Tach Out

3. Launch LabVIEW NXG by pressing the **Windows** key and selecting **LabVIEW NXG**.

4. Select **File » Open Browse…**, and navigate to *C:\Seminars\cDAQ + LabVIEW NXG HO\Exercises\Vibration\* to open **Vibration Exploration.lvproject**.

5. Click on the **home** button to the top left, then select **Use Your Hardware**. This will bring up **SystemDesigner** Live View, where you can see all the hardware currently connected to your system.

6. Click on the **CompactDAQ Chassis** and select **C Series Sound and Vibration Input Module.**

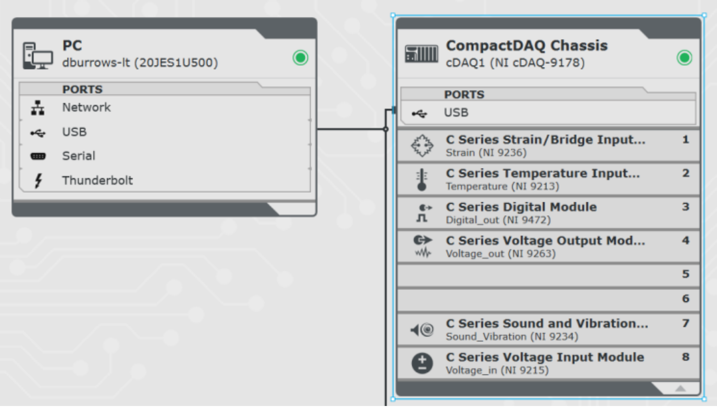
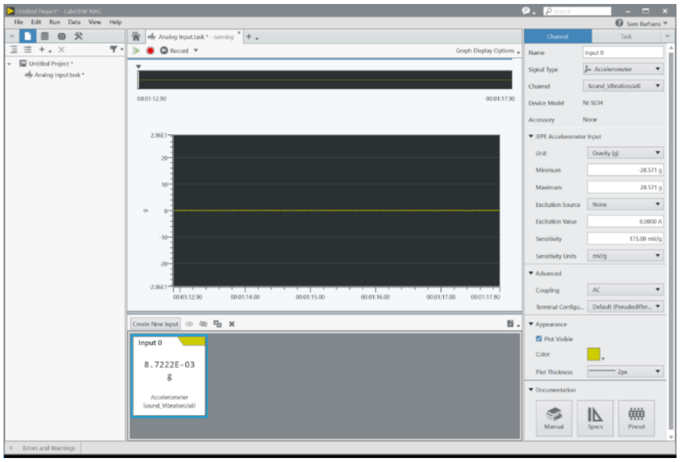


Figure 7-2: Display of all connected hardware with appropriate drivers installed.

7. In the Configuration Pane to the right of your screen, click **Create Analog Input** to begin acquiring a basic voltage measurement in a LabVIEW NXG measurement panel. Data should automatically be acquired and displayed on the graph provided. The graph should look like Figure 7-3.



Input 0
8.7222E-03
g
Accelerometer
Sound_Vibration/ai0

Name: Input 0
Signal Type: Accelerometer
Channel: Sound_Vibration/ai0
Device Model: NI 9234
Accessory: None

▼ IEPE Accelerometer Input
Unit: Gravity (g)
Minimum: -28.571 g
Maximum: 28.571 g
Excitation Source: None
Excitation Value: 0.0000 A
Sensitivity: 175.00 mV/g
Sensitivity Units: mV/g

▼ Advanced
Coupling: AC
Terminal Configu...: Default (Pseudodiffer...

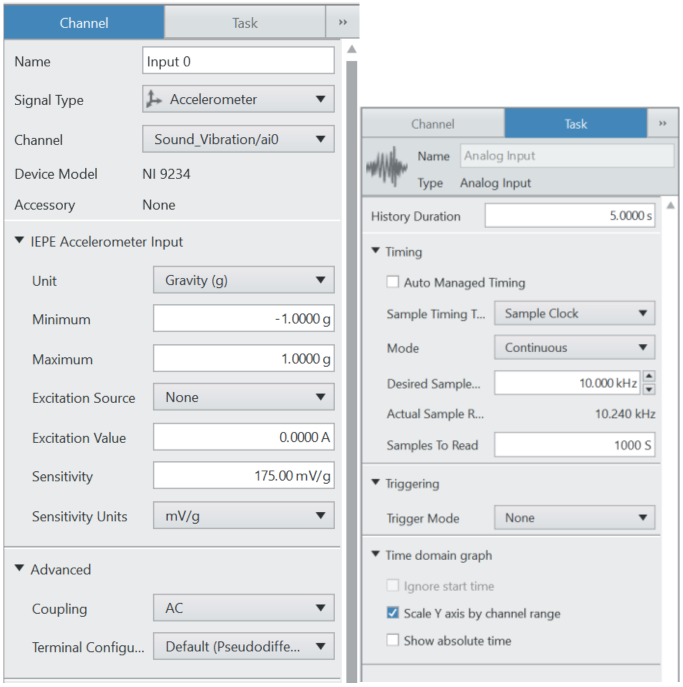
▼ Appearance
☑ Plot Visible
Color: 🟨
Plot Thickness: 2px

▼ Documentation

Figure 7-3: The configured measurement is used to confirm electrical connections and quickly acquire data.

In addition to verifying hardware functionality, measurement panels allow you to interactively create acquisition and generation configurations for DAQ hardware. These configurations are saved in files called tasks.

8. To set up the measurement task, use the configuration pane on the right to set the Channel and Task settings as shown in Figure 7-4.



Name: Input 0
Signal Type: Accelerometer
Channel: Sound_Vibration/ai0
Device Model: NI 9234
Accessory: None

▼ IEPE Accelerometer Input
Unit: Gravity (g)
Minimum: -1.0000 g
Maximum: 1.0000 g
Excitation Source: None
Excitation Value: 0.0000 A
Sensitivity: 175.00 mV/g
Sensitivity Units: mV/g

▼ Advanced
Coupling: AC
Terminal Configu...: Default (Pseudodiffe...

Name: Analog Input
Type: Analog Input
History Duration: 5.0000 s

▼ Timing
☐ Auto Managed Timing
Sample Timing T...: Sample Clock
Mode: Continuous
Desired Sample...: 10.000 kHz
Actual Sample R...: 10.240 kHz
Samples To Read: 1000 S

▼ Triggering
Trigger Mode: None

▼ Time domain graph
☐ Ignore start time
☑ Scale Y axis by channel range
☐ Show absolute time

Figure 7-4 The configuration pane contains measurement parameters for the measurement panel.

9. After entering the settings, ensure the panel is running, then bump the demo box or the table and verify that the graph reacts to your input. If you detect no noticeable change, please alert your instructor at this time.

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. LabVIEW NXG's interactive measurement panels provide the insight into your system and setup to help you eliminate mistakes early in your development process. Similar functionality is available through the Test Panels of NI MAX, a hardware configuration and management tool.

10. Once you have confirmed that the vibration hardware is working, click the stop icon ● on the top left to stop acquisition.

11. Save the task by clicking **File » Save Analog Input.task** or pressing CTRL + S.

12. Create a new folder to store your tasks by clicking the plus sign above the project files and selecting **folder**. Name the folder **Measurement Tasks**, and place the analog input task in the folder by dragging and dropping it into place. Your project files should look like Figure 7-5 .

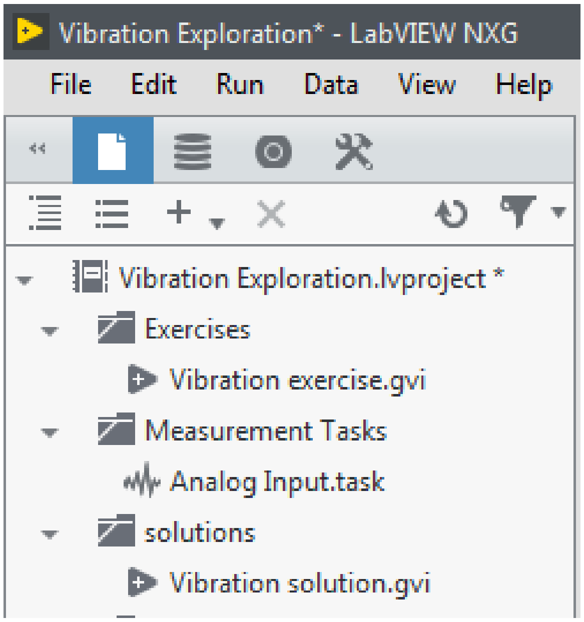


Figure 7-5 The VIs used in your project can be found in the project files view of the navigation pane.

13. Double click **Vibration solution_Task Method.gvi**.

14. Through the rest of this section you will build the VI shown in Figure 7-6. Ensure that the switch on the Sound and Vibration Signal Simulator in your hardware demo box is flipped to BNC. Press the run button in the solution program.

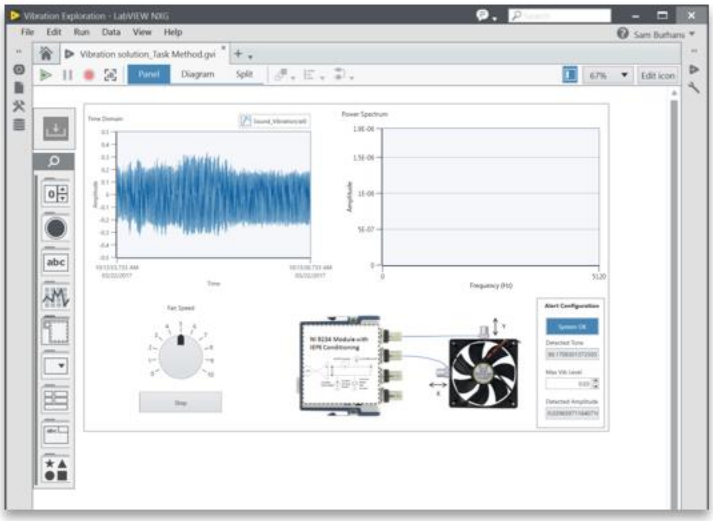


Figure 7-6 This VI will acquire a time-domain vibration signal and calculate the frequency content using LabVIEW VIs

15. Using the knob in the center of the front panel, 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. When finished, close the solution without saving.

17. To start the exercise, double-click **Vibration exercise.gvi**. The Front Panel has already been partially 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 LabVIEW NXG to calculate the frequency components of the acquired signal.

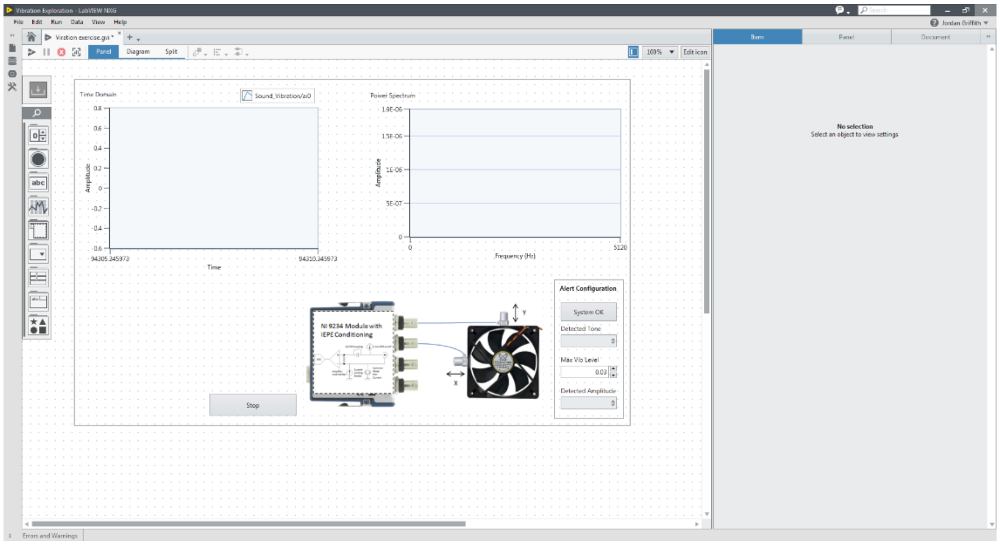
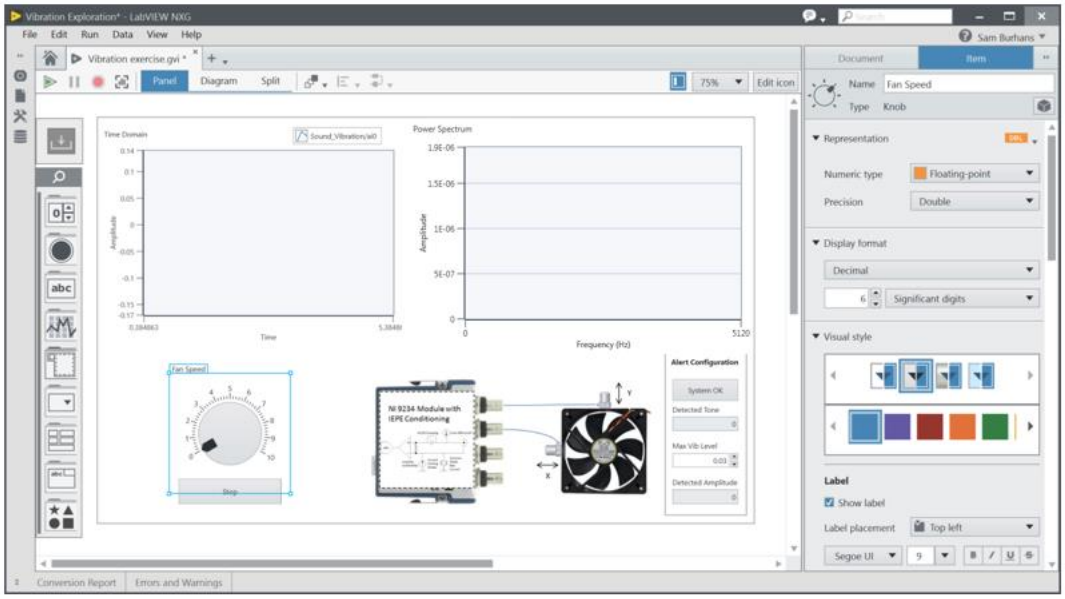


Figure 7-7 Beginning with the Front Panel design is a good way to organize your thoughts around the inputs and outputs that you expect.

18. Note that one of the controls from the solution, the knob control for adjusting fan speed, has not been provided for you. To add one, press to **Ctrl + Space** or click on the magnifying glass icon at the top of the palette and type “knob” to quickly locate the knob control.

19. Once selected, click in the front panel to place the control in the empty space above the stop button. You may want to expand the size of the control and give it a descriptive label like “Fan Speed” to make it more user friendly. Your front panel should now look like Figure 7-8.



Name Fan Speed
Type Knob
Representation
Numeric type Floating-point
Precision Double
Display format
Decimal
6 Significant digits
Visual style
Label
☑ Show label
Label placement Top left
Segoe UI 9

Figure 7-8 You can add controls to the front panel by selecting them from the palette. Controls can be arranged and sized in any way you'd like.

20. Notice that the run arrow in the top left 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 can compile and 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 .

21. Close the Errors and Warning pane and select **Diagram** Panel Diagram Icon at the top of your front panel. The keyboard shortcut to switch between the front panel and block diagram is Ctrl + E. When the block diagram opens, you will see the following code:

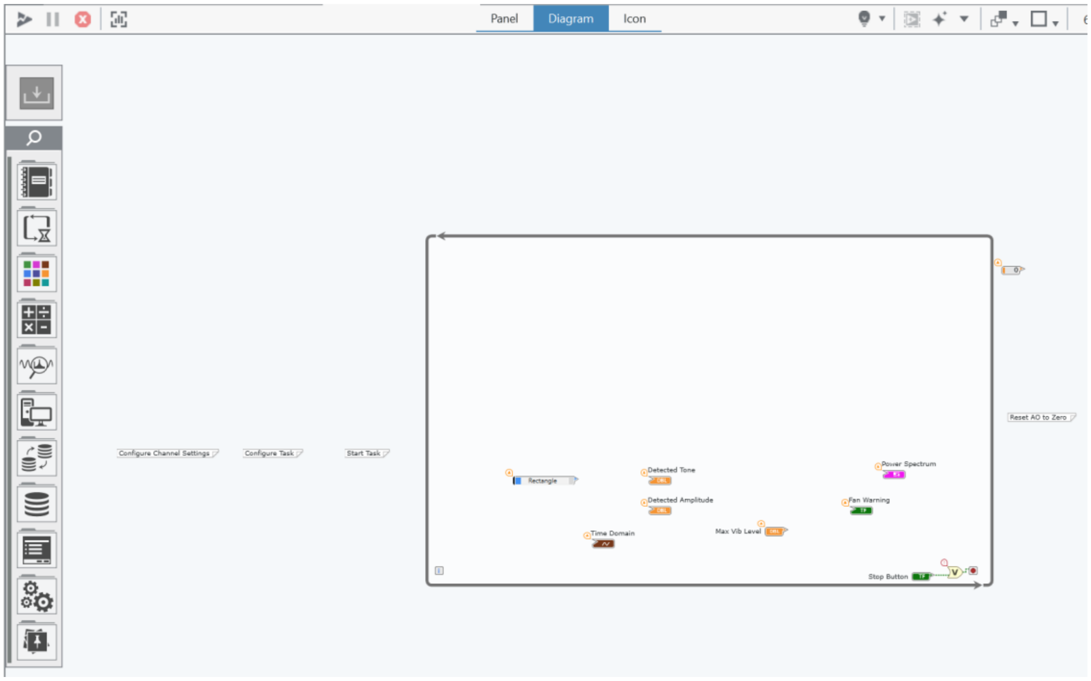


Figure 7-9 The Block Diagram does the work of your program. This is where you will write the code to control the Front Panel.

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 **Time Domain** icon. LabVIEW will highlight the location of the corresponding **Time Domain** indicator on your panel. 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 white text 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.

22. Take note of the **unplaced items** menu of the block diagram palette. This is where the elements that correspond to newly placed front panel items, like the knob added earlier, will be available.

This ensures you can easily locate new elements and add them to your block diagram code efficiently. When new elements are available in the unplaced items menu, there will be a blue dot over the icon as in Figure 7-10.

23. Place the Fan Speed control in the top left corner of the while loop for later use.

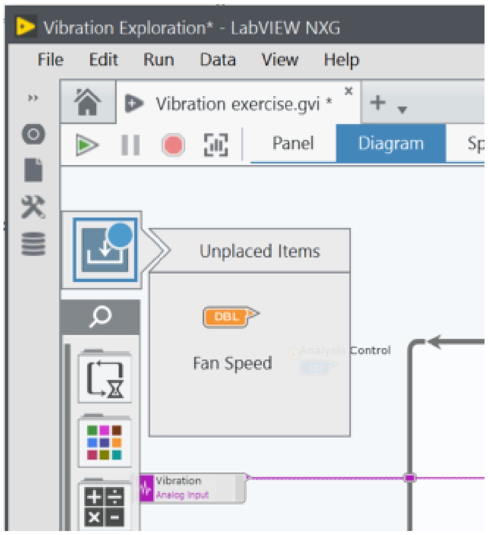


Figure 7-10 New block diagram elements are stored in the Unplaced Items menu for easy access.

24. 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.

25. Find the task files saved at the beginning of this exercise in the Measurement Tasks folder in the Navigation Pane. Right click on **Analog Input.task** to rename the task to **Vibration AI.task**. Drag **Vibration AI.task** on to the block diagram as shown in Figure 7-11.

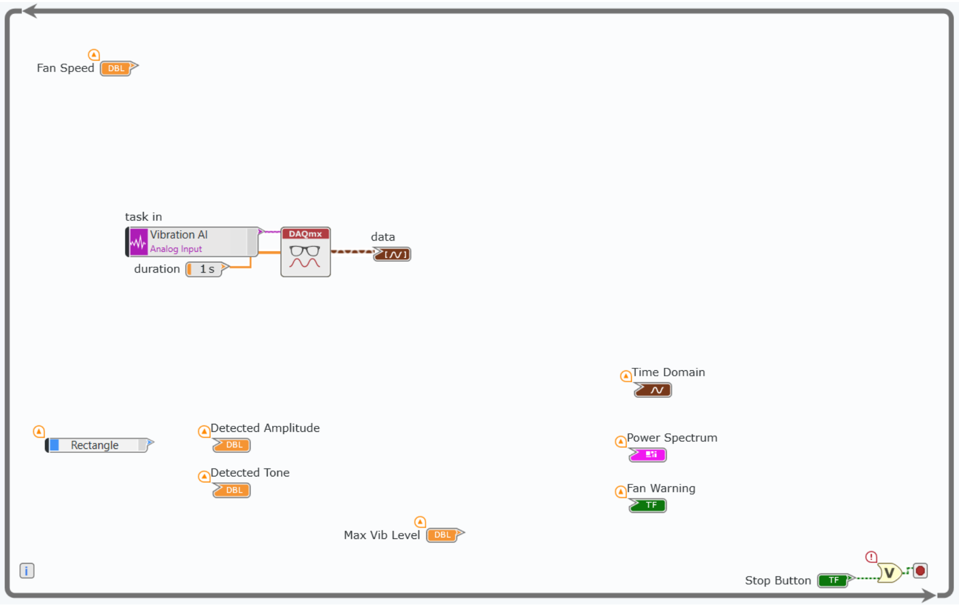


Figure 7-11 Dragging a task on to the block diagram places a task constant, DAQmx Read or Write function, and any required inputs or relevant outputs.

26. Delete the *data* indicator that was placed along with the task, it is not required for this program. Clear any broken wires with Ctrl + B.

27. Wire the **data** output of DAQmx Read to the provided **Time Domain** indicator; this will plot our acquired time domain data in the graph on the front panel.

28. Move the **task in** constant outside the left side of the while loop; it should remain wired to the DAQmx Read function.

29. For this exercise, a large part of the coding will come from the DAQmx palette. To access this palette, click **Hardware Interfaces**, then select **NI-DAQmx**.

30. Click on the **double arrow** (<<) to keep this palette visible (arrow in Figure 7-12).

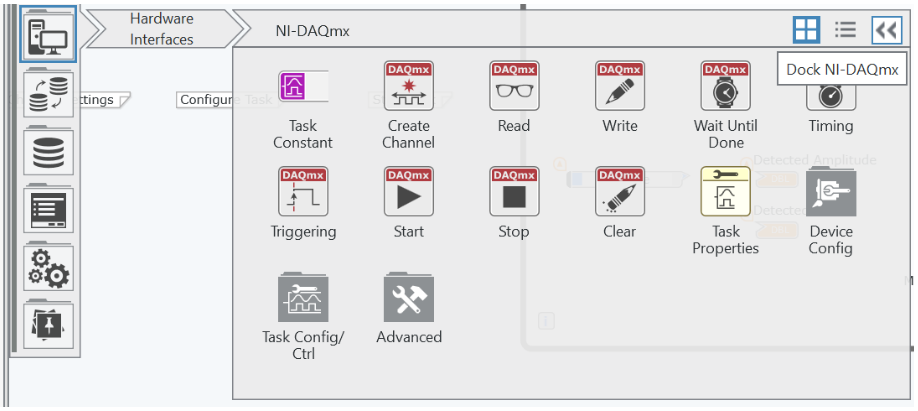


Figure 7-12 The DAQmx palette includes all the required VIs for communicating with your NI CompactDAQ chassis.

31. Drag the following VIs from the DAQmx palette and place them as shown in Figure 7-18.

    a) DAQmx Stop Task

    b) DAQmx Clear Task

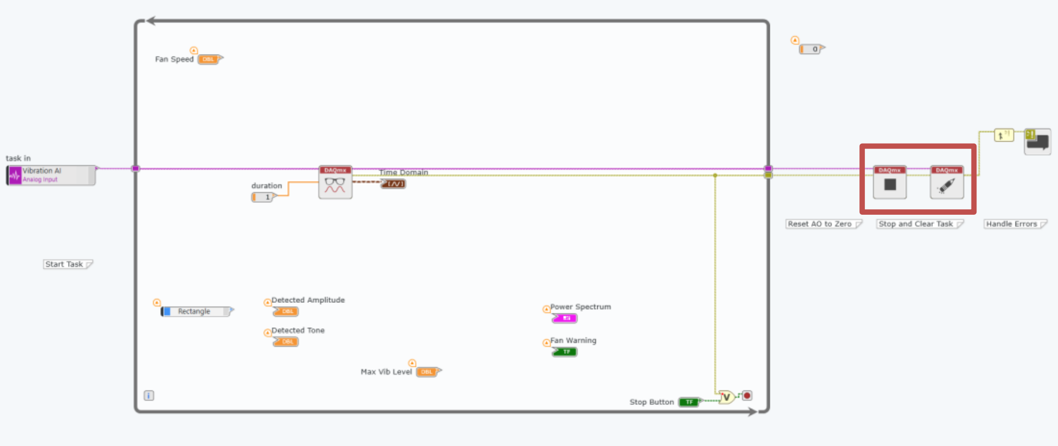


Figure 7-13 DAQmx functions will always follow the same pattern, regardless of the hardware operation.

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 or written to, then the task is stopped and the channel is cleared. When using DAQmx tasks as in this case, channel creation and configuration are embedded in the task you created using measurement panels. Doing so programmatically will be covered in Part D of this exercise.

32. Wire task and error wires through the code as in Figure 7-14. This will pass the task values and any errors that may arise through the code. The Retain First Error and Display Error functions at the end of the program will then display any errors that may have occurred during operation to the user.

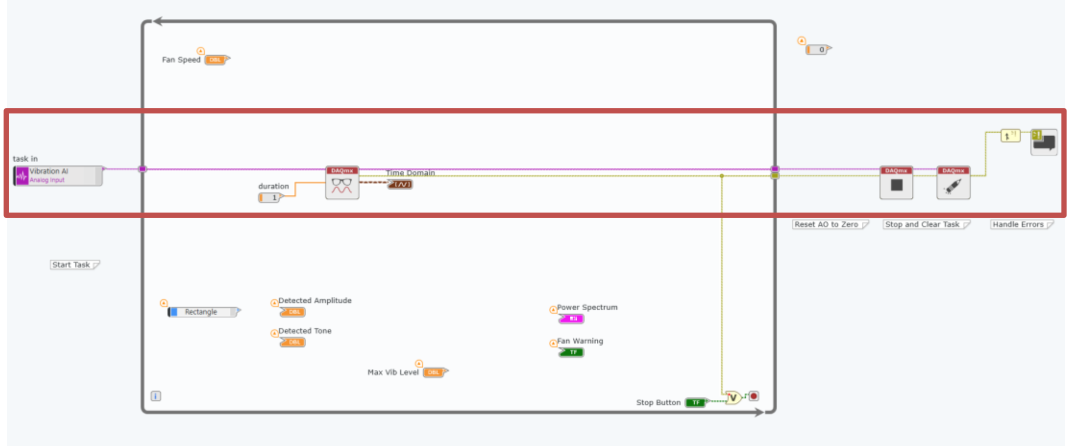


Figure 7-14 Wiring the task and error wires through dictates the flow of your code.

33. Wire the error wire from the DAQmx Read VI to the input of the Or function, as shown in Figure 7-15. This VI will OR the inputs to stop the while loop from executing. The code you just wrote will stop the loop if an error occurs or a user stops presses the stop button.

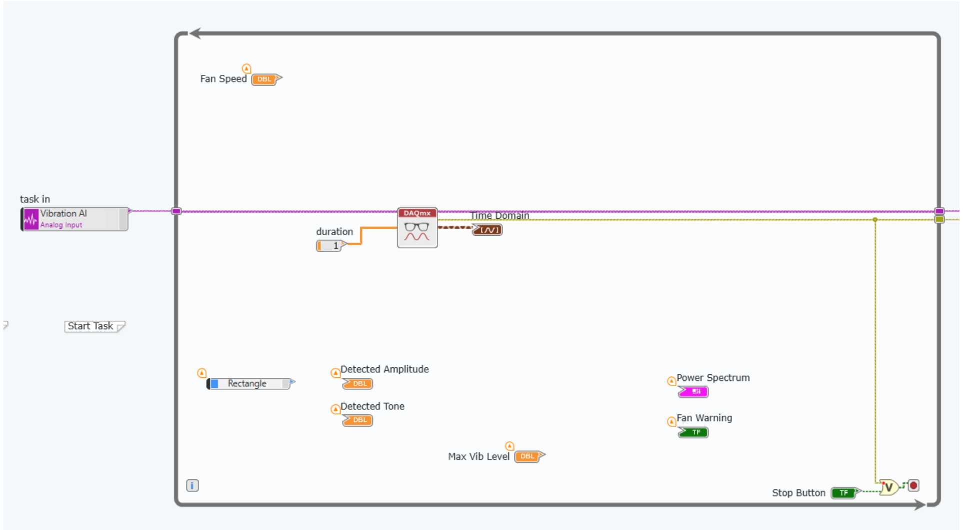


Figure 7-15 The OR function concatenates the inputs, stopping the While Loop if the code has an error or the user presses stop.

34. Once you wire the error wire to the OR function, your code should be ready to run. Check to see that the Run Arrow is intact (▶).

35. Press **Ctrl+E** to toggle to the Front Panel.

36. 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 with the dial. Notice the changing signal on the screen.

37. Stop the program by clicking the **stop** button.

Part B Controlling the Fan Speed with Analog Output

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, we must first create an analog output task.

a) Create an Analog Output using the **New** button: + ▾

b) By default, Voltage_out/ao0 is added to the task. Change this to Voltage_out/ao1.

c) In the **Test Data** section of the configuration pane, change the **Voltage** to 5V.

d) Change the physical fan switch from Dial to BNC, then press the **Run** button. You should hear the fan turn on.

e) Set the **Voltage** to 0V. You should hear the fan turn off.

f) Stop the analog output with ✖. Save the task as Fan Control AO.task.

2. Open the Block Diagram of Vibration exercise.gvi

3. Repeat the drag-and-drop process from Part A for **Fan Control AO.task**. Place is as shown in Figure 7-16.

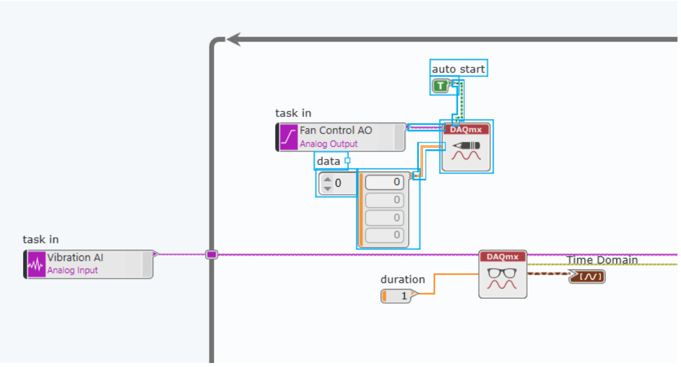
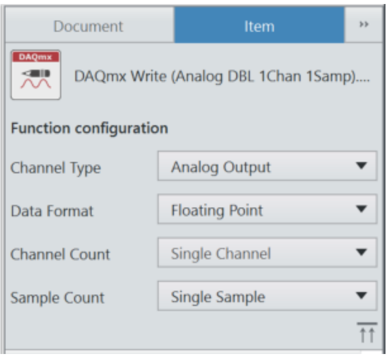


Figure 7-16 Dragging a pre-configured task onto the block diagram provides all the code necessary to perform that task's operation.

4. Delete the numeric array constant and move the task constant outside of the while loop.

5. Connect the pre-existing **Fan Speed** control to the **data** terminal of the DAQmx Write VI.

6. Notice that the wire you just created is broken. Select the wire and hover your cursor over it, a tooltip should appear telling you why the wire is broken. In this case, the data type of the control (double) does not match the input terminal data type (1D Array of doubles). To fix this, reconfigure the DAQmx Write function as **Single Channel, Single Sample** using the configuration pane as in Figure 7-17.



DAQmx Write (Analog DBL 1Chan 1Samp)....

Function configuration

Channel Type: Analog Output

Data Format: Floating Point

Channel Count: Single Channel

Sample Count: Single Sample

Figure 7-17 You can reconfigure most DAQmx functions by selecting the function on the block diagram. The configuration pane will populate with properties that can be adjusted.

7. Like in the previous section of this exercise, drag the following VIs onto the block diagram and place them like you see in Figure 7-18.

   a) DAQmx Write

   b) DAQmx Stop

   c) DAQmx Clear

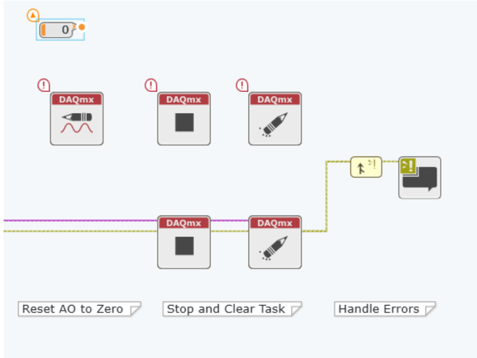


Figure 7-18 The flow of DAQmx code almost always follows the same pattern

You may have noticed that you now have 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.

8. Wire the Fan Speed control into the data input of the first DAQmx Write VI.

9. Wire the numeric constant next to the second DAQmx Write VI as seen in Figure 28. Select this DAQmx write function and configure as shown in Figure 7-17.

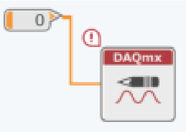


Figure 7-19 The second DAQmx Write writes a zero to the buffer to make sure the fan turns off when the VI is stopped.

10. Place a **Retain First Error** VI between the Clear Task and Error Handler. This VI takes errors from multiple sources to simplify error handling. To better understand how this function works, use context help (Ctrl + H).

11. Wire the task and error wire through the code, as seen in Figure 7-20.

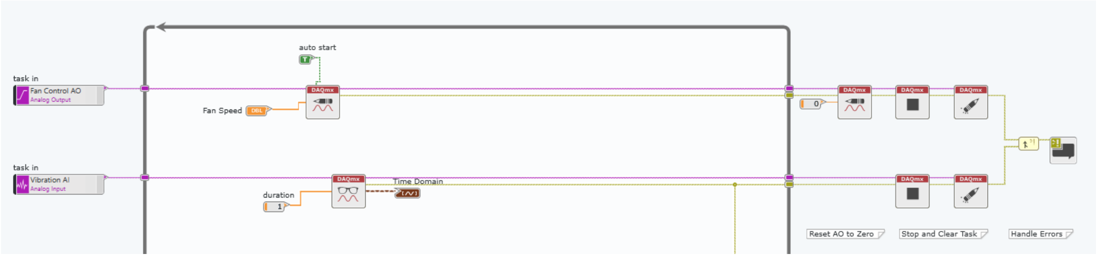


Figure 7-20 The task and error wire communicate information between each of the VIs in the code.

12. Expand the OR function inside the while loop to accept an additional input as in Figure 7-21.

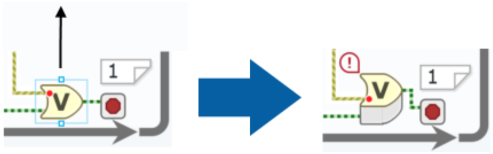


Figure 7-21 Some functionals can be expanded to accept additional inputs.

13. Create a branch of the analog output task error wire (CTRL + Click where you would like to branch the wire) and wire it to the newly created terminal of the OR function.

14. Switch to the Front Panel. Your run arrow should not be broken at this time.

15. Press the Run button.

16. 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.

17. Stop the VI and save your work. You will use it in the next part of this exercise.

Part C Moving from Tasks to the DAQmx API

As you'll recall from earlier parts of this exercise, Tasks are created by interactively configuring their settings through a LabVIEW NXG measurement panel. This is very convenient for getting quick measurements and testing hardware configurations as you create them. As applications expand to include multiple tasks of different types, however, making changes to settings configured within a task can become *inconvenient. Imagine a project with 20, 30, or 100 different tasks and the time it would take to open a measurement panel to adjust a configuration setting on each one.*

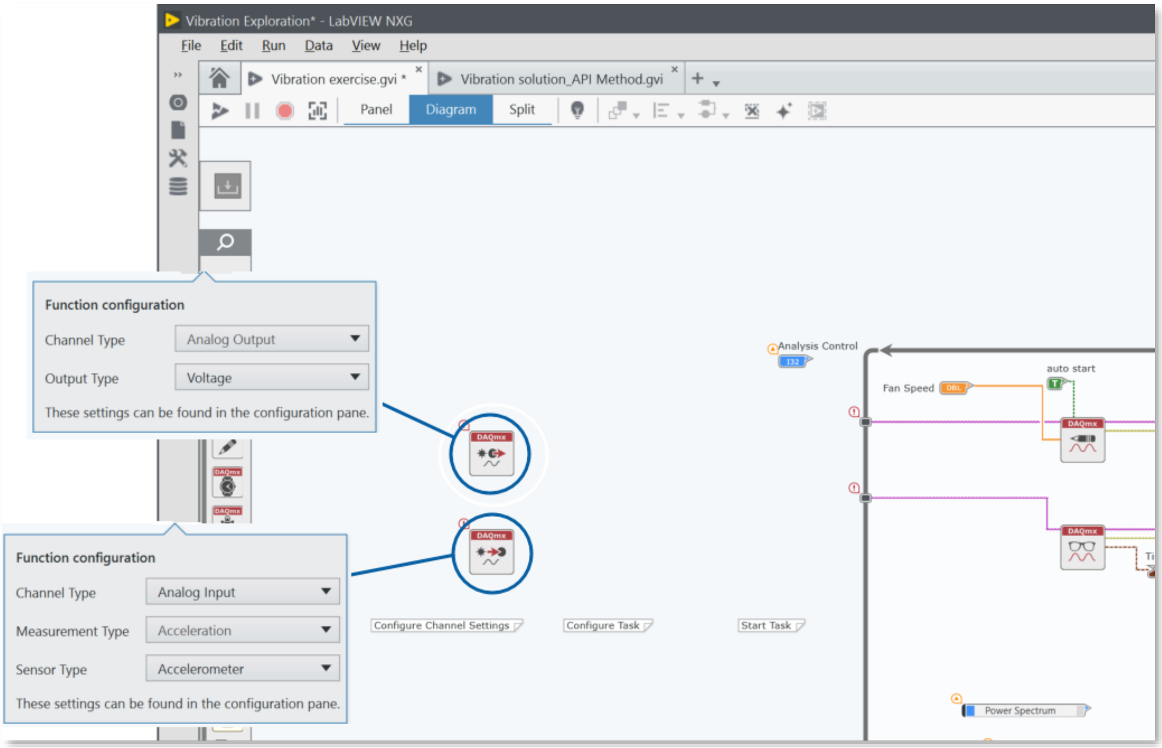
The task model of DAQmx, while convenient and quick, does not necessarily scale for all applications. Luckily, tasks can also be created and configured programmatically using the same DAQmx API functions. Adopting this method means creating more customizable programs that may be better suited for building streamlined projects. In this part of the exercise, we'll take the existing vibration exercise VI and adapt it to use the API instead of tasks.

1. To begin, open the block diagram of Vibration exercise.gvi.

2. Delete Fan Control AO and Vibration AI task input controls. Clear any broken wires with Ctrl + B.

3. In the palette, navigate to **hardware interfaces » NI-DAQmx** (you may want to pin this the palette for easy access).

4. In the palette, find **DAQmx Create Virtual Channel** and place it twice. Configure each as specified in Figure 7-22, below.



Function configuration

Channel Type: Analog Output

Output Type: Voltage

These settings can be found in the configuration pane.

Function configuration

Channel Type: Analog Input

Measurement Type: Acceleration

Sensor Type: Accelerometer

These settings can be found in the configuration pane.

Figure 7-22 Create Virtual Channels.gvi creates a virtual channel to reference physical channels of a DAQ device.

5. To allow the user to select which physical channels will be used for acquisition and generation, we must create front panel **controls**. To do so, right-click on the Physical Channels terminal on the left side of each VI and select **Create control,** as in Figure 7-23. Give the controls meaningful labels, like *Fan Control Channels* and *Vibration Input Channels*.

A control, as the name implies, is an input with a visible element on the front panel that can be controlled by the user. Conversely, a constant cannot be changed by the user at run time and is only visible on the block diagram.

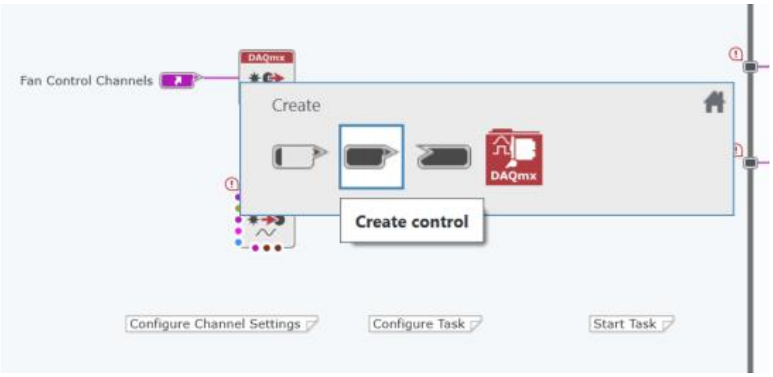


Figure 7-23 Controls allow the user to adjust inputs as run time. Every control will have a front panel element that corresponds to a block diagram element.

6. Continue to place the following VIs and wire their task and error wires together as shown in Figure 7-24:
   a. DAQmx Timing
   b. DAQmx Start

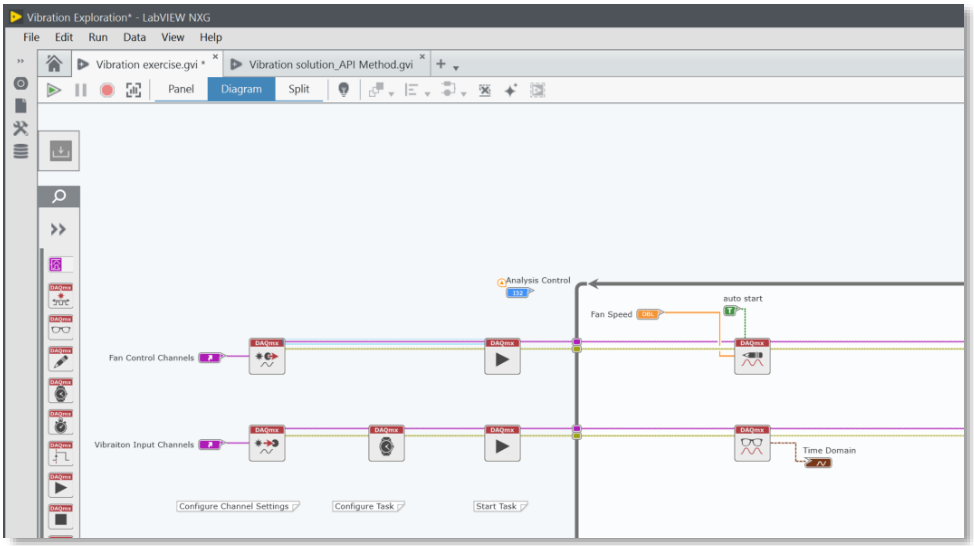
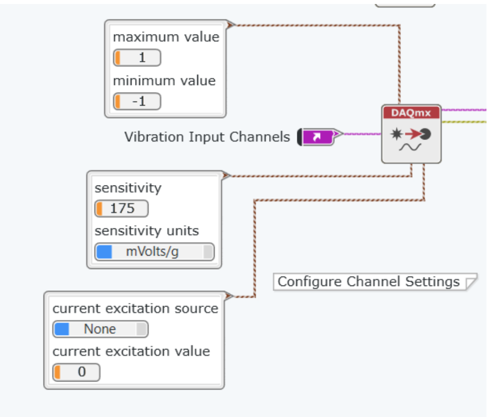


Figure 7-24 The DAQmx API will always follow the same basic order of operations, regardless of hardware used or measurement type.

7. Take a closer look at the DAQmx Create Virtual VIs – note how they have different inputs because they are creating virtual channels for different measurement types. The analog input vibration instance of this VI has configuration inputs specific to accelerometer measurements. Right-click

and select **Create constant** for each of the following inputs. Modify the values of the constants to match Figure 7-25:

a. Measurement range
b. Sensitivity
c. Current excitation



maximum value 1

minimum value -1

Vibration Input Channels

sensitivity 175

sensitivity units mVolts/g

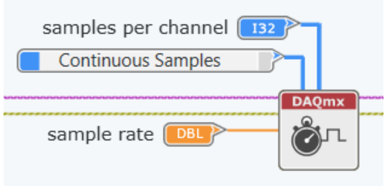
Configure Channel Settings

current excitation source None

current excitation value 0

Figure 7-25 Inputs to the Create Virtual Channels VI vary depending on how the VI is configured. For acceleration, as in this case, measurement-specific inputs are available like accelerometer sensitivity.

8. The DAQmx timing VI is used to configure a task's timing mode, sample rate, and samples per channel. Create controls and constants, as in Figure 7-26.



samples per channel I32

Continuous Samples

sample rate DBL

Figure 7-26 DAQmx Timing is used to configure timing mode, sample rate, and samples per channel for a task.

9. DAQmx start.gvi commits the settings configured previously and allows the task to start. If no triggering is configured, data acquisition will begin when this function is called. DAQmx start is a very important function when task synchronization becomes important, as it gives you the ability to control the order in which various tasks start.

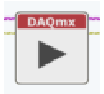
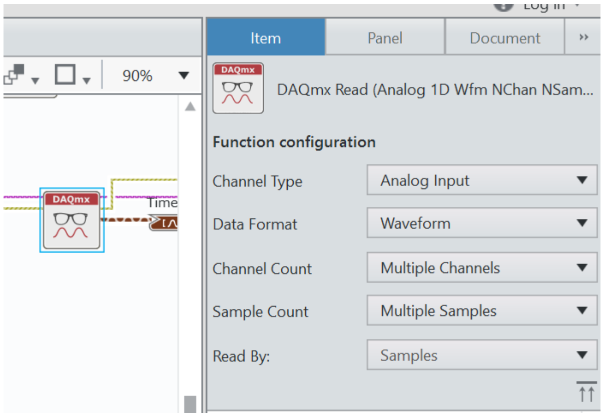


Figure 7-27 DAQmx start commits the task configuration and transitions it into the running state to begin acquisition or generation.

10. Click on the DAQmx Read Function. In the Configuration Pane, change **Read By:** to **Samples**. This will configure our task to read sets of samples at a time, instead of acquiring for 1s at a time. Delete the Duration constant from this function and press CTRL+B to clean up wires.



DAQmx Read (Analog 1D Wfm NChan NSam...

Function configuration

Channel Type: Analog Input

Data Format: Waveform

Channel Count: Multiple Channels

Sample Count: Multiple Samples

Read By: Samples

Figure 7-28 Changing the DAQmx Read VI to "Read By: Samples" means that when this function is called, it will read a set of X samples at the sample rate, rather than 1s of samples.

11. By default, the DAQmx Read function will acquire any and all available samples per channel. A good practice is to set DAQmx Read to read the same number of channels that are being written to the hardware buffer, as specified at the DAQmx Configure Timing VI. The easiest way to achieve this is to branch a ware from the **Samples Per Channel** control and connect it to the **number of samples per channel** input of DAQmx Read, as shown in Figure 7-29.

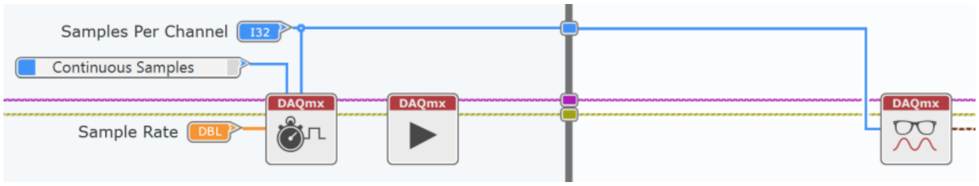


Figure 7-29 You should specify the samples per channel to both the DAQmx Configure Timing VI and the DAQmx Read VI.

12. At this point, the run arrow should be solid. Switch to the front panel.

13. The controls we created on the block diagram are not currently visible on the front panel. New controls and indicators are stored in the **unplaced items** (arrow in Figure 7-30) bin or the palette for easy access. Place the controls from the unplaced items on the front panel as shown in Figure 7-30, you may also need to re-arrange existing elements.

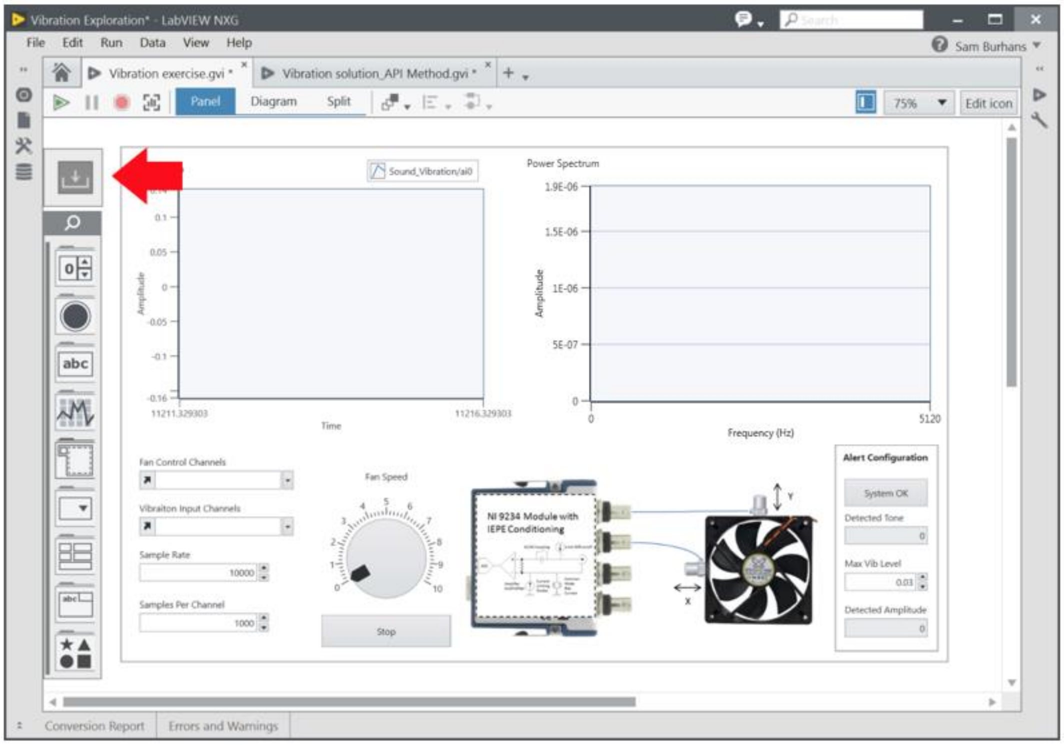
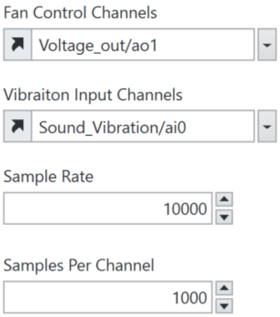


Figure 7-30 New front panel elements are stored in the unplaced items menu of the palette.

14. Set the Fan control channels, vibration channels, sample rate, and samples per channel controls appropriately, as in Figure 7-31.



Fan Control Channels: Voltage_out/ao1

Vibraiton Input Channels: Sound_Vibration/ai0

Sample Rate: 10000

Samples Per Channel: 1000

Figure 7-31 Using the DAQmx API means creating front panel controls that allow the user of your program to adjust configuration settings easily at run-time.

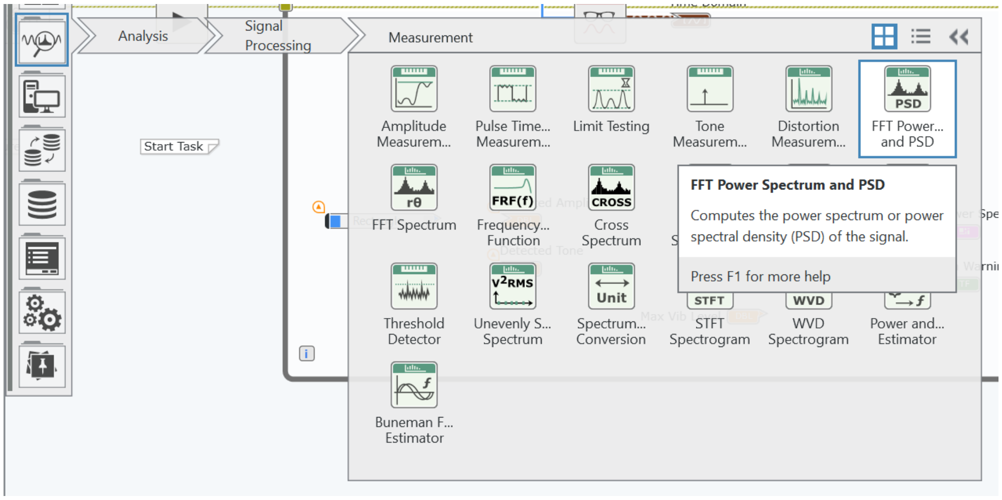
15. Run the VI. It should perform the same as it did previously when using tasks created in measurement panels. Save the VI and leave it open for the final part of this exercise.

Part D Adding Analysis to Your Acquisition Code

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 exercise.gvi.
2. From the palette, select **Analysis » Signal Processing » FFT Power Spectrum and PSD** as seen in Figure 7-32.



FFT Power Spectrum and PSD

Computes the power spectrum or power spectral density (PSD) of the signal.

Press F1 for more help

Figure 7-32 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 7-33, leave the default configuration as-is (Power Spectrum; Continuous).

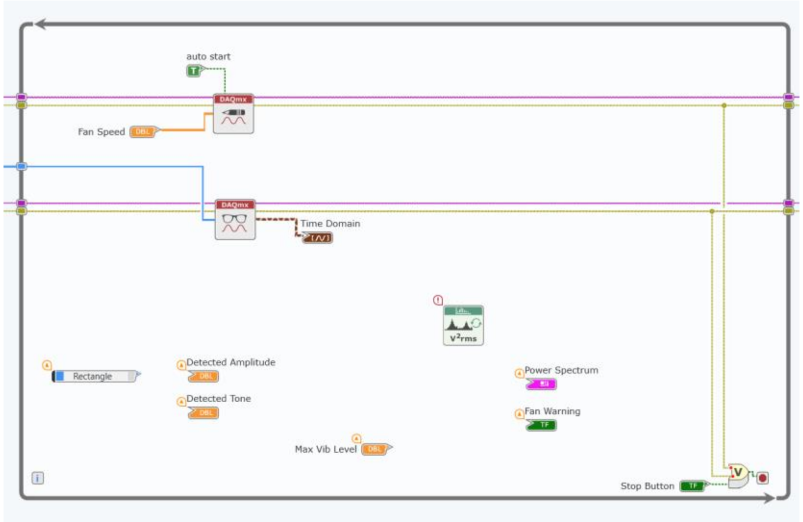


Figure 7-33 The FFT Power spectrum will calculate the frequency spectrum during the acquisition.

4. Wire the inputs to the VI as seen in Figure 7-34.

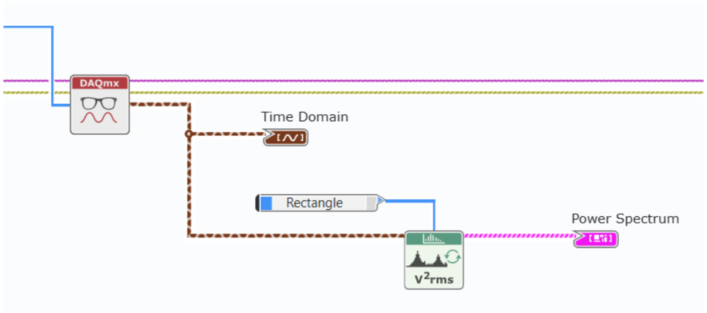


Figure 7-34 The power spectrum will be calculated each time the loop in the code runs.

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 in the graph on the LabVIEW PS tab.

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.

8. Stop the VI.

Bonus Optional Challenge Exercise

In the VI, there are four controls and indicators we did not use: **Detected Amplitude, Detected Tone, Max Vib Level,** and **Fan Warning**.

For an additional challenge, use the analysis functions available in LabVIEW to make use of these controls and indicators.

- Represent the amplitude of the vibration signal acquired with the **Tone Measurement** function.
- Represent the tone of the vibration signal acquired with the **Tone Measurement** function.
- Allow the user to select a maximum vibration level – when the vibration exceeds this point, turn on the **Fan Warning** LED**.**

<End of Exercise>