NATIONAL INSTRUMENTS

NI myRIO

Design Real Systems, Fast

Official Manual for the *Learn to Design Real Systems Fast with NI myRIO* Hands-On Workshop

Introduction to National Instruments LabVIEW

Students and educators need to have some understanding of the LabVIEW environment before attempting tasks like building a real-time application or customizing an FPGA. To help develop this knowledge, NI designed this hands-on workshop for those who have no LabVIEW experience. However, it is not a substitute for actual training — it does not cover every context menu, button or function in LabVIEW.

LabVIEW is a graphical programming environment that students can use to quickly develop applications that scale across multiple platforms and operating systems. The power of LabVIEW is in its ability to interface with thousands of devices and instruments using hundreds of built-in function libraries to help you accelerate development time and quickly acquire, analyze, and present data.

Applications in LabVIEW mimic the appearance of real instruments (like multimeters, signal generators, or oscilloscopes), so they are called virtual instruments or VIs. Every LabVIEW application has a front panel, an icon/connector pane and a block diagram. The front panel serves as the imitation of the real-world user interface of the device that the VI is defining. Programmers can leverage the flexibility of using multiple forms of representation for the data the instrument is analyzing. The icon/connector pane is analogous to terminals or plugs on a real-world instrument that allow it to be connected to other devices. Therefore, VIs can call other VIs (called subVIs) that are all connected, and in turn, each of those subVIs can contain more VIs similar to function calls in a text-based programming language. Lastly, the block diagram is where the programmer actually creates the code. Unlike text-based programming languages such as C, Java, C++, and Visual Basic, LabVIEW uses icons instead of lines of text to create applications. Due to this key difference, execution control is handled by a set of rules for data flow rather than sequentially. Wires connecting the nodes (coding elements) and VIs on the block diagram determine code execution order.

In summary, LabVIEW VIs are graphical, driven by dataflow and event-based programming, and are multitarget and multiplatform capable. They also have object-oriented flexibility and multithreading and parallelism features. LabVIEW VIs can be deployed to real-time and FPGA targets.

The LabVIEW Getting Started Window

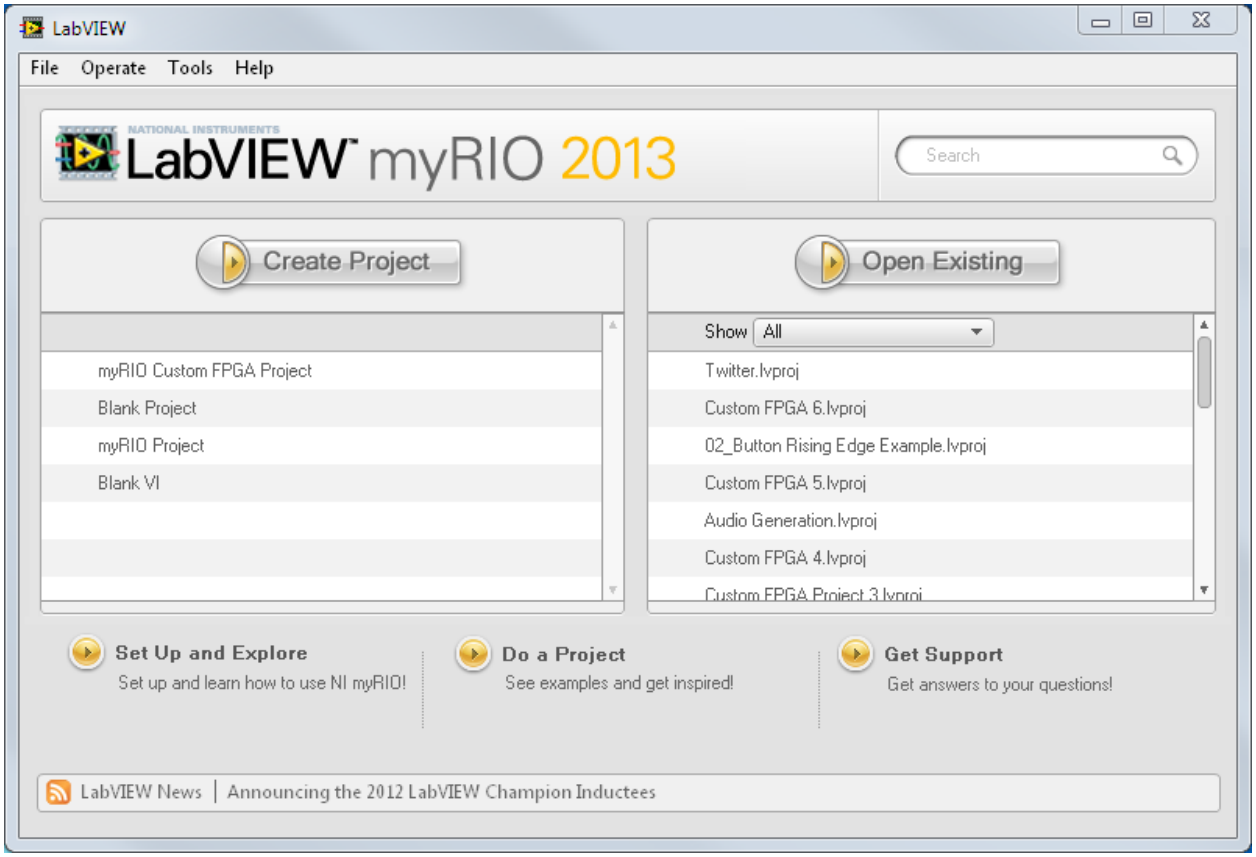
Using the LabVIEW DVD that came with the NI myRIO device, follow the instructions for installing LabVIEW and any add-ons you have purchased. Activate the products using the Product Activation Wizard, and the software should be ready to use.

In Windows, you can launch LabVIEW from the start menu, either by navigating to the LabVIEW shortcut or by searching for "LabVIEW."

When the LabVIEW executable runs and all of the resources are loaded, the Getting Started window appears.

As long as the "show on launch" check box is marked, the Set Up and Explore window appears. For the NI myRIO device, this window has been tailored to the expected needs of a student equipped with an NI myRIO device. With this window, students can launch the Getting Started Wizard for NI myRIO much like the USB monitor does, access getting started tutorials, find help for programming in LabVIEW, and configure the WiFi on the NI myRIO device. For now, close this window.

From the Getting Started window, you can create new projects and VIs and choose from different options to open existing work and the NI Example Finder. You can access this window anytime by selecting **VIEW»Getting Started Window** from any VI. A few additional options are available at the bottom of the window. Each option links you to useful information, tutorials, exercises and support for your endeavors with the NI myRIO device. Use these links often while learning to navigate the LabVIEW environment.



LabVIEW Getting Started Window

Part of the NI myRIO philosophy is to bridge the gap between the real problem and its solution while creating as few intermediate problems as possible. Avoid pitfalls like "blank VI syndrome" (being overwhelmed by the blank block diagram) by using the Getting Started window to find examples, create projects from templates, and take advantage of online forums and support. And when it comes to simple systems and problems, many users and NI engineers have already created material that either does exactly what the system in question requires or something very similar. Reinventing the wheel can be painful but with the power of LabVIEW and the network of engineers and scientists creating code to support the platform, you can avoid that.

What Is NI myRIO?

The NI myRIO embedded student design device was created for students to "do real-world engineering" in one semester. It features a 667 MHz dual-core ARM Cortex-A9 programmable processor and a customizable Xilinx field-programmable gate array (FPGA) that students can use to start developing systems and solve complicated design problems faster—all in a sleek enclosure with a compact form factor. The NI myRIO device features the Zynq-7010 All Programmable system on a chip (SoC) to unleash the power of NI LabVIEW system design software both in a real-time (RT) application and on the FPGA level. Rather than spending copious amounts of time debugging code syntax or developing user interfaces, students can use the LabVIEW graphical programming paradigm to focus on constructing their systems and solving their design problems without the added pressure of a burdensome tool.

NI myRIO is a reconfigurable and reusable teaching tool that helps students learn a wide variety of engineering concepts as well as complete design projects. The RT and FPGA capabilities along with onboard memory and built-in WiFi allow students to deploy applications remotely and run them "headlessly" (without a remote computer connection). Three connectors (two NI myRIO expansion ports [MXP] and one NI miniSystems port [MSP] that is identical to the NI myDAQ connector) send and receive signals from sensors and circuitry that students need in their systems. Forty digital I/O lines overall with support for SPI, PWM out, quadrature encoder input, UART, and I^2C; eight single-ended analog inputs; two differential analog inputs; four single-ended analog outputs; and two ground-referenced analog outputs allow for connectivity to countless sensors and devices and programmatic control of systems. All of this functionality is built-in and preconfigured in the default FPGA functionality, which eliminates the need for expansion boards or "shields" to add utility. Ultimately, these features allow students to do real-world engineering right now—from radio-controlling vehicles to creating stand-alone medical devices.

Student friendly and fully capable out of the box, the NI myRIO device is simple to set up, and students can easily determine its operational status. A fully configured FPGA personality is deployed on the device from the factory, so beginners can start with a functional foundation without having to program an FPGA to get their systems working. However, the power of reconfigurable I/O (RIO) becomes apparent when students start defining the FPGA personality and molding the behavior of the device to the application. With the device's scalability, students can use it from introductory embedded systems classes through final-year design courses.

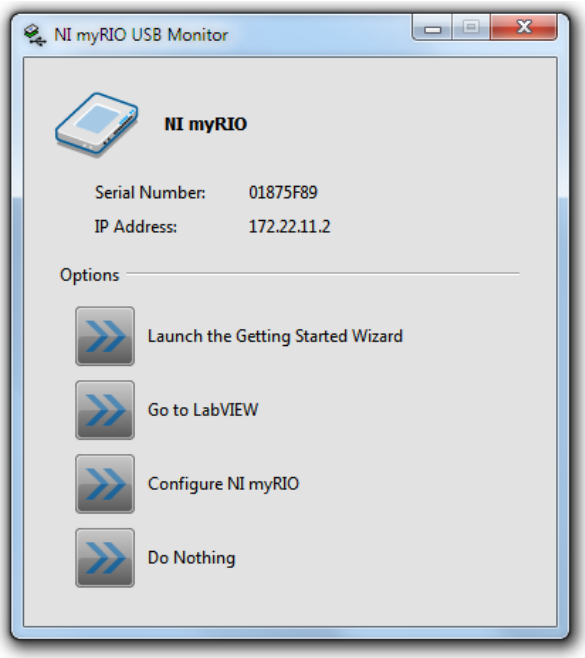
Hardware Setup: Connecting to the NI myRIO Device

One of the goals of the NI myRIO design is to simplify hardware setup. To accomplish this, NI myRIO software provides a custom setup and configuration utility separate from the NI Measurement & Automation Explorer (MAX) configuration utility. You can still use MAX for setup, software installation, and configuration if you are more comfortable with that environment. The NI myRIO device has a USB monitor application that runs when you connect the device to the host computer. Learn how to use the NI myRIO USB monitor and the NI myRIO Getting Started Wizard in the following section.

The NI myRIO USB Monitor

Make sure you power the NI myRIO device using the power adapter that came with it. Plug the USB Type B end of the USB cable into the NI myRIO device. Connect the other end of the cable to your computer's USB port.

Without starting LabVIEW or NI MAX, if the device is powered, the OS should recognise the NI myRIO device and install and set up the drivers for it. Once this is complete, in the Windows OS, Windows should automatically launch the NI myRIO USB Monitor shown below.



Along with the serial number and IP address, you have four options to choose from when an NI myRIO device is detected:

1. Launch the Getting Started Wizard

With the Getting Started Wizard, you can quickly observe the functional status of the NI myRIO unit. This wizard checks for connected NI myRIO devices, connects to the selected device, ensures that the software is up to date, suggests an update if the software is out of date, offers the option of renaming the device, and then shows a screen similar to a front panel that you can use to observe the accelerometer function, turn on and off onboard LEDs, and test the user-defined button.

The final screen of the Getting Started Wizard presents two options:

Start my first project now
Selecting this option launches a web browser-based tutorial similar to the project covered in Exercise 2 of this seminar.

Go straight to LabVIEW
Selecting this option launches the LabVIEW Getting Started window.

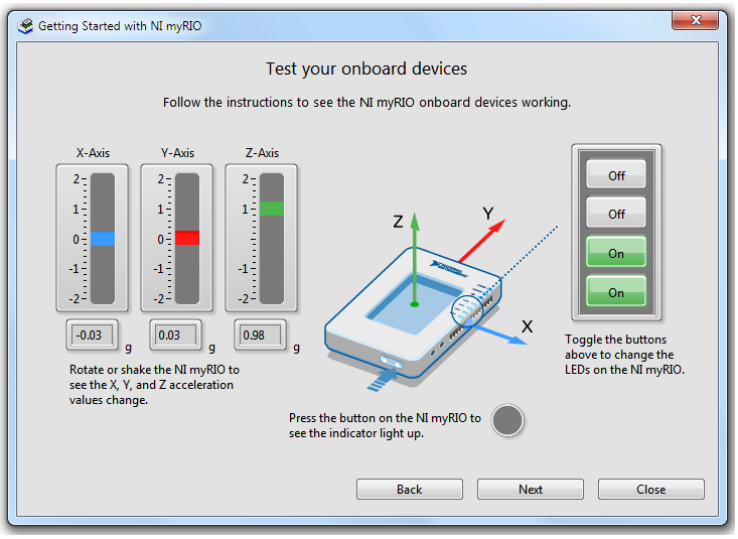
2. Go to LabVIEW
Selecting this option launches the LabVIEW Getting Started window.

3. Configure NI myRIO
Selecting this option opens the web browser-based configuration utility for the NI myRIO device.

4. Do Nothing
If LabVIEW is open already and a project is configured targeting the NI myRIO device, you can use this option to close the NI myRIO USB Monitor when a unit is reconnected to the development computer.

Launching the Getting Started Wizard

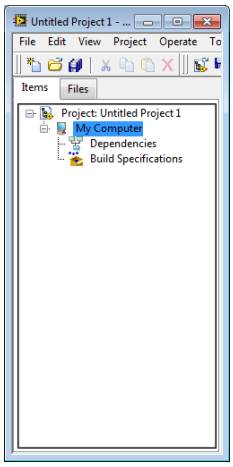
From the NI myRIO USB Monitor, select **Launch the Getting Started Wizard.** Select **Next** and the wizard connects to the NI myRIO unit, checks for software on it, and prompts you to rename the device. If the NI myRIO unit does not have software installed on it, the wizard automatically installs the most up-to-date software. After that, a diagnostic window opens. With it, you can observe the values of the built-in three-axis accelerometer, test the functionality of the user-defined push button, and toggle the four onboard LEDs.



Now that you have installed and configured the NI myRIO device, you can create real-time VIs and run them on the processor (similar to Windows VIs) along with FPGA VIs to harness the power of true parallel processing.

Optional Exercise 0a: Creating a LabVIEW Project

This exercise is optional, but recommended for new LabVIEW users



Create a LabVIEW project. A LabVIEW project is required to deploy code to embedded targets, such as a myRIO. Additionally, using LabVIEW projects, even for simple applications, is good programming practice, as it will help you manage the application if it scales and evolves into something more complex in the future

Goal

- Become familiar with the LabVIEW project

The LabVIEW Project Explorer helps you manage all resources related to an application. These resources might include multiple VIs, custom user interface objects, images, text documents, configuration files and deployment information. The project structure allows for quick and easy resource control and you can use the LabVIEW Project Explorer to allocate certain resources to certain devices (typically called targets).

For those of you accustomed to other integrated development environments (IDEs), the LabVIEW Project Explorer will feel familiar and its usage will be mostly the same. LabVIEW is a cross-platform language with comprehensive support for add-ons and libraries. You can write most VIs while targeting them to the development machine and easily retarget them to an NI embedded device, such as NI myRIO, because these targets run an OS that works with LabVIEW applications and add-on libraries.

The LabVIEW Project Explorer hierarchy for an individual project does not reflect the organization of the source code files on the disk. Moving files in the project view does not affect the actual disk copy of it—just the view of that file in the LabVIEW Project Explorer.

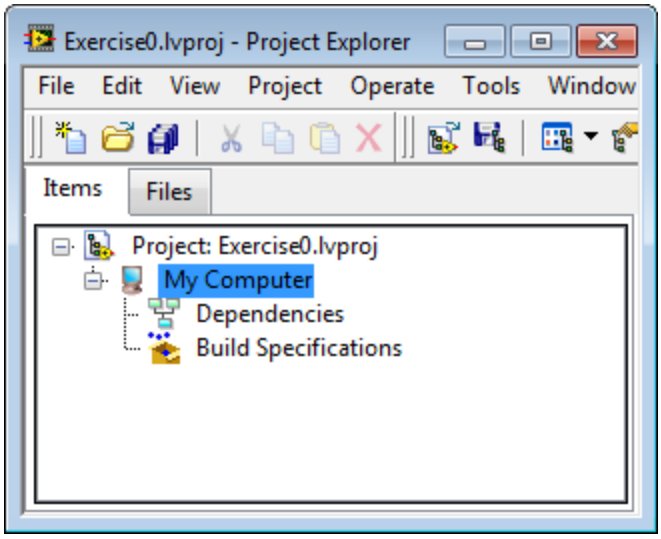
TIP: A *target* is any device that can run a VI. For instance, your developer computer and myRIO device can be regarded as *targets*

Complete the following steps to create a new project:

1. If it is not already open, launch LabVIEW
2. Select the **Create Project** button on the LabVIEW Getting Started window.
3. The Create Project window opens and offers several options. You can explore all of the options using the *More Information* link associated with any project template you are interested in.
4. For now, select the **Blank Project** option as the starting point for the project.
5. Then click the **Finish** button. This creates an empty project and opens the LabVIEW Project Explorer.

6. To save the new project:
   a. From the LabVIEW Project Explorer window, select **File»Save**.
   b. A save dialogue box opens and requests a destination directory and project name. Navigate to **myRIO Workshop\Exercises\Exercise 0** folder which you will find on your desktop. Give the VI a meaningful name, such as **Exercise 0 Project.lvproj**

The LabVIEW Project Explorer window is showing a brand new project that is ready to be populated with resources and source code. The LabVIEW Project Explorer features a standard menu of familiar options (File, Edit, View and so on) along with some options exclusive to LabVIEW. The LabVIEW Project Explorer itself has two panes - items and files. The items page shows the items in a project organised in a hierarchy, while the files page shows where project items are stored on the hard drive.



LabVIEW Project Explorer

In the items tab of the LabVIEW Project Explorer, you can see the organization of the project. The project root, the first item on the list, shows which project you are working in.

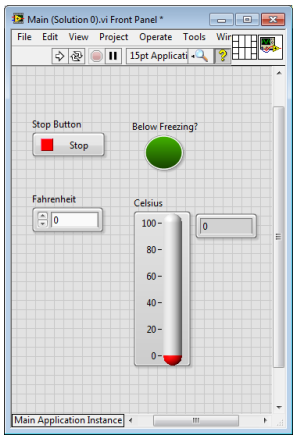
Under the project root, the next level of the tree contains all of the targets that the project is pointing to. A blank project defaults to the local development computer or "My Computer". Under the My Computer target, the build specifications for the target are shown. Build specifications include the build configurations for source distributions and other builds available in LabVIEW toolkits and modules. You use these when distributing applications to colleagues/customers or deploying applications to embedded systems.

The LabVIEW Project Explorer becomes particularly useful when developing complex applications. However, using projects, even for simple applications, is good practice, as it will help you manage the application if it scales and evolves into something more complex in the future.

Keep this project open so you can use it in the next section.

Optional Exercise 0b: Creating a LabVIEW VI in Windows

This exercise is optional, but recommended for new LabVIEW users



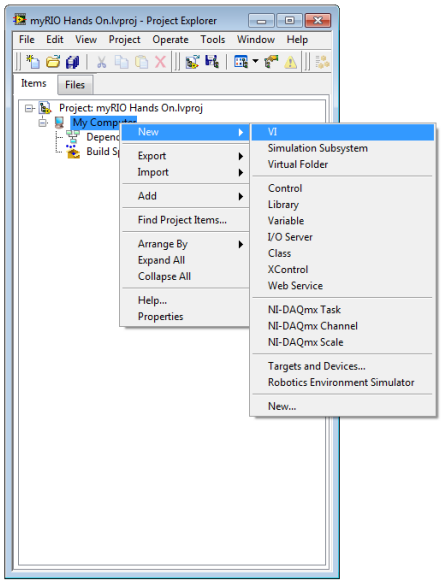
In this exercise, you will create a simple, but functional, application. The application will convert a temperature value from Fahrenheit to Celsius.

Goal

- Experience creating a LabVIEW application from scratch
- Become familiar with the LabVIEW development environment and fundamental graphical programming concepts

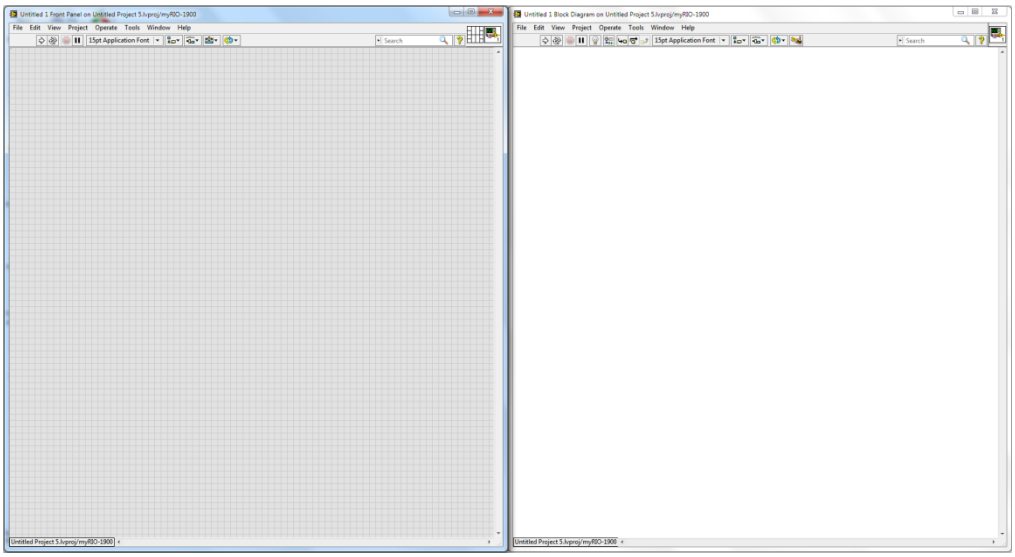
---

To enter the programming portion of LabVIEW, you must create a new VI (virtual instrument). To create a VI that will execute on your local development machine (My Computer – the only target currently in the project), right-click My Computer and select **New » VI** in the LabVIEW Project Explorer.



Creating a Blank VI

This creates a blank VI under the **My Computer** target in the project. This new blank VI consists of two windows: the front panel and the block diagram.



Front Panel and Block Diagram

Save the new VI by selecting **File»Save** from either the front panel or the block diagram. When the save dialog appears, give the VI a meaningful name (for example, **Exercise 0**)

Now that you have saved the VI, you can create the temperature conversion code. Complete the following steps to write the VI:

1. We want our code to run continuously and, just like text-based languages, LabVIEW uses loops to continuously execute tasks. You can terminate loops with a button on the front panel or by using logic to determine when a task has been completed.

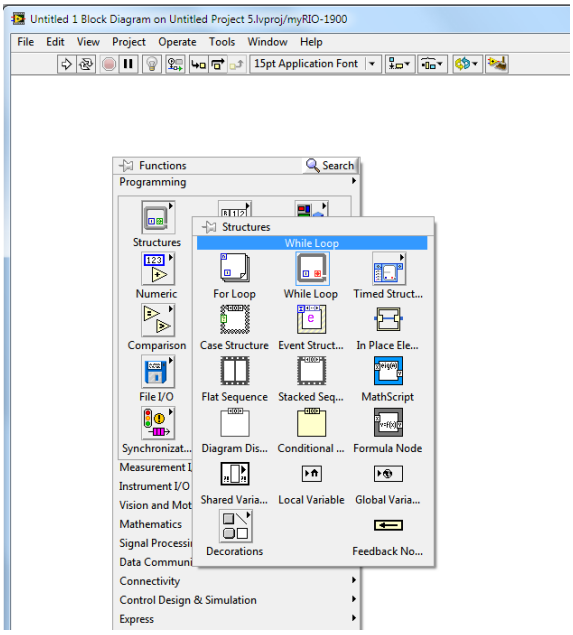
   For this exercise, use a While Loop to continuously monitor the Fahrenheit value and convert it to Celsius. Use a button on the front panel to control termination.

   a. Switch to the block diagram by either selecting the block diagram window (white background) or pressing **<Ctrl+E>** from the front panel (this shortcut can be used to quickly switch back and forth between these two windows).

TIP: Learning keyboard shortcuts greatly accelerates navigation through the LabVIEW environment. Also, they make you look like a LabVIEW wizard!

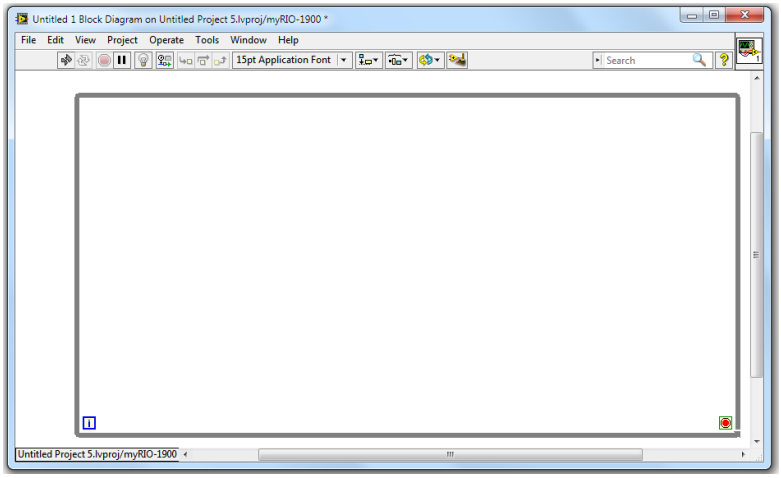
   b. **Right-click** on the block diagram to open the Functions palette. This palette contains all of the nodes and VIs you use to program in LabVIEW. You can customize the palette display, but by default all add-ons and VIs are available. Navigate to **Programming»Structures»While Loop** and select the While Loop.

      Notice that the cursor changes to the *draw-loop* icon



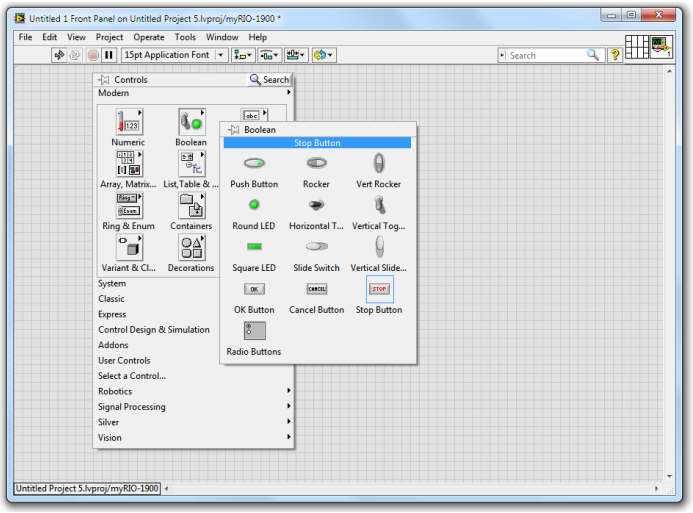
Selecting a While Loop

c. Draw a **while loop** on the block diagram. **Left-click** and drag the cursor to create the loop. You can resize loops by hovering the cursor over the placed while loop, then dragging one of the eight blue resizing squares.



Block Diagram with While Loop

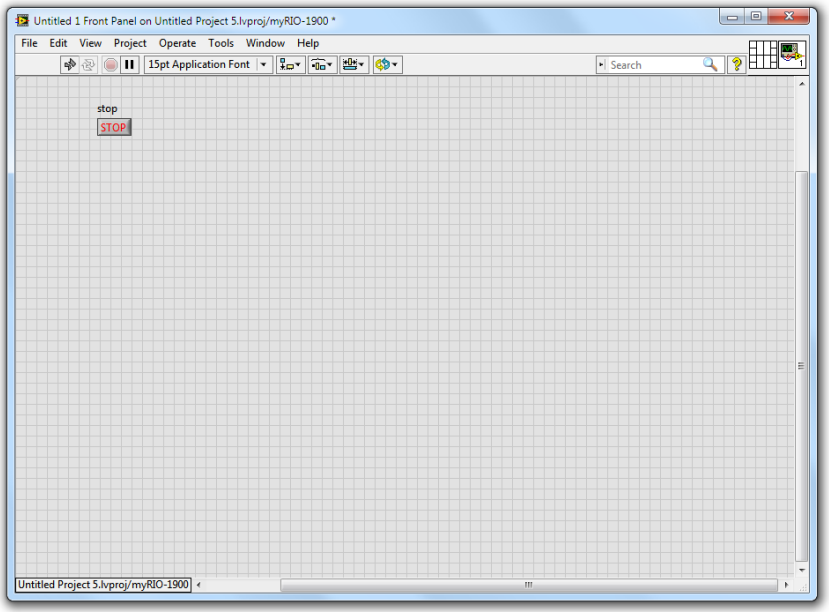
d. Switch back to the front panel (grey background) using the **<Ctrl+E>** shortcut. Now **right-click** to open the Controls palette. This palette contains all of the elements used to create the VI's user interface. Navigate to **Modern»Boolean»Stop Button**.



Selecting a Stop Button

e. **Left-click** the stop button in the palette and drag it onto the front panel, then **left-clicking** on an appropriate location.

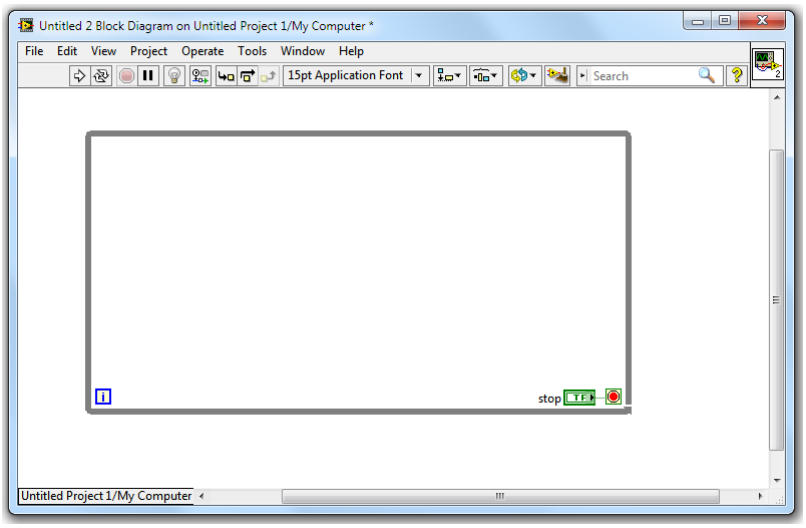
TIP: Aesthetically different versions of the stop button can be found in the system, classic and silver sub-palettes. You can select the button that best fits your desired style of UI. LabVIEW also allows you to create custom controls if required.



Front Panel with Stop Button

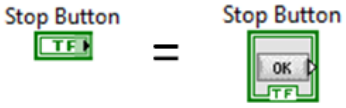
f. Switch to the block diagram and locate the new stop button terminal. **Left-click** and drag the stop button terminal – relocate it next to the *conditional terminal* (located in the lower right corner of the While Loop)

g. Wire the stop button to the *conditional terminal* by **left-clicking** on the wiring terminal on the **stop button** and then left clicking the conditional terminal. A green wire should appear between the two icons. The colour green indicates that Boolean values will pass through the wire (other data types have differently coloured wires).



While Loop with Stop Condition Wired

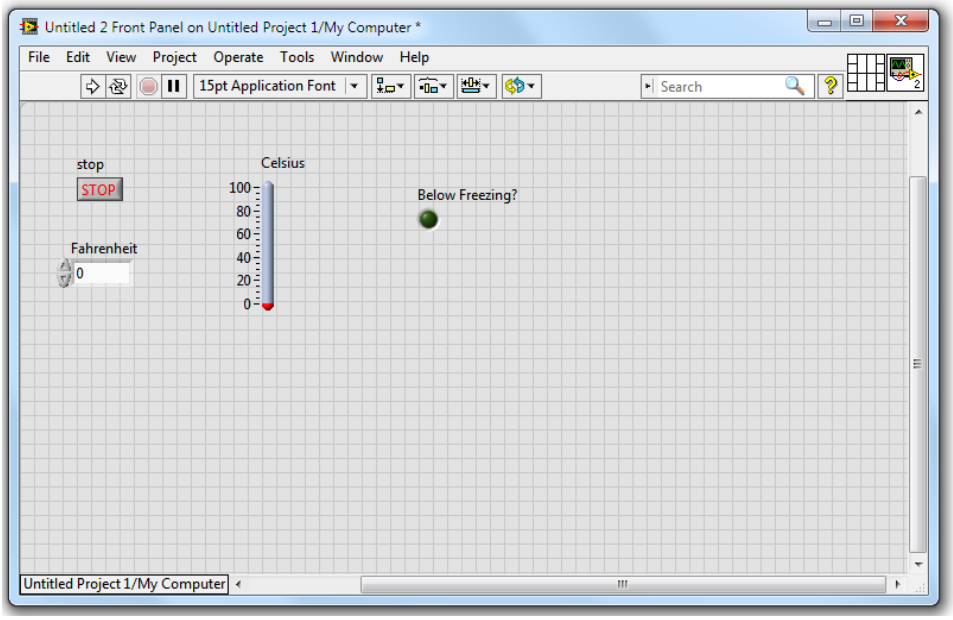
TIP: Terminals are the block diagram representation of front panel objects. They can be viewed in two different styles. One is smaller and more compact. The other is larger, but more descriptive.



You can switch between these two styles by right clicking on the terminal and selecting **View as Icon**

2. Now create the Fahrenheit control and Celsius indicator on the front panel.
   a. Switch to the front panel and **right-click** to open the Controls palette. Navigate to **Modern»Numeric»Numeric Control**.
   b. Place the control on the front panel by **left-clicking** it in the palette, moving the cursor over the front panel and **left-clicking** again to place it.
   c. Notice that the control is labeled "Numeric Control." This is not a very descriptive name. Rename the control by **double-clicking** on the text label, then typing "Fahrenheit."
   d. Open the Controls palette (**right-click**) and navigate to **Modern»Numeric»Thermometer.** Place the thermometer on the front panel.
   e. In a similar fashion to the Fahrenheit control, rename the thermometer indicator by **double-clicking** the label. Name this indicator "Celsius."
   f. Now add a virtual LED to the front panel, which will illuminate when the temperature falls below freezing. Place an LED on the front panel by navigating to **Modern»Boolean»Round LED**.

g. Place the LED on the front panel.
h. Rename the LED "Below Freezing?"



Front Panel with Controls and Indicators

3. With the proper controls and indicators on the front panel, you must implement the mathematics and logic to convert Fahrenheit to Celsius on the block diagram.
   a. Switch to the block diagram and observe the new icons for the Fahrenheit, Celsius and Below Freezing controls and indicators. Move the Fahrenheit icon inside the While Loop on the left and both the Celsius and Below Freezing icons inside the While Loop on the right. Give yourself plenty of space between the icons for the code.

The Block Diagram with unwired Control Terminals

   b. Place a subtract node on the block diagram by **right-clicking** to open the Functions palette and navigating to **Programming»Numeric»Subtract**. Place the node just to the right of the Fahrenheit icon.
   c. **Wire** the output terminal of the Fahrenheit icon to the top input terminal of the subtract node.
   d. **Right-click** on the bottom input terminal of the subtract node and select **Create»Constant.**
   e. Enter a value of "32" into the constant.

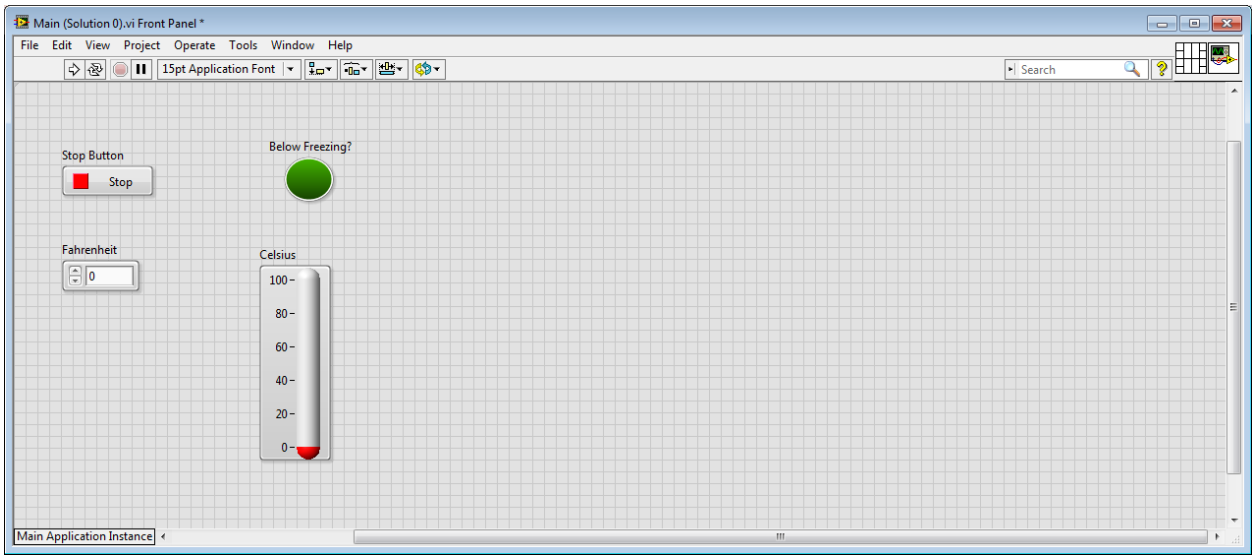
f. Place a multiply node on the block diagram by **right-clicking** to open the Functions palette and navigating to **Programming»Numeric»Multiply**. Place the node to the right of the subtract node.
g. **Wire** the output terminal of the **subtract** node to the top input terminal of the **multiply** node.
h. Place a **divide** node on the block diagram by **right-clicking** and navigating to **Programming»Numeric»Divide**. Place the **divide** node somewhere below the **subtract** node.
i. **Right-click** on the top input terminal of the **Divide** function and select **Create»Constant**. Assign a value of "5" to this constant.
j. **Right-click** on the bottom input terminal of the **Divide** function and select **Create»Constant**. Assign a value of "9" to this constant.
k. **Wire** the output terminal of the **divide** node to the bottom input terminal of the **multiply** node.
l. **Wire** the output terminal of the **multiply** node to the input terminal of the Celsius indicator icon.
m. Place a "less than 0" comparison node on the block diagram by **right-clicking** and navigating to **Programming»Comparison»Less Than Zero?** Place the node somewhere to the right of the **multiply** node.
n. **Wire** the output of the **multiply** node to the **less-than-zero?** node.
o. **Wire** the output of the **less-than-zero?** node to the input of the Below Freezing indicator icon.

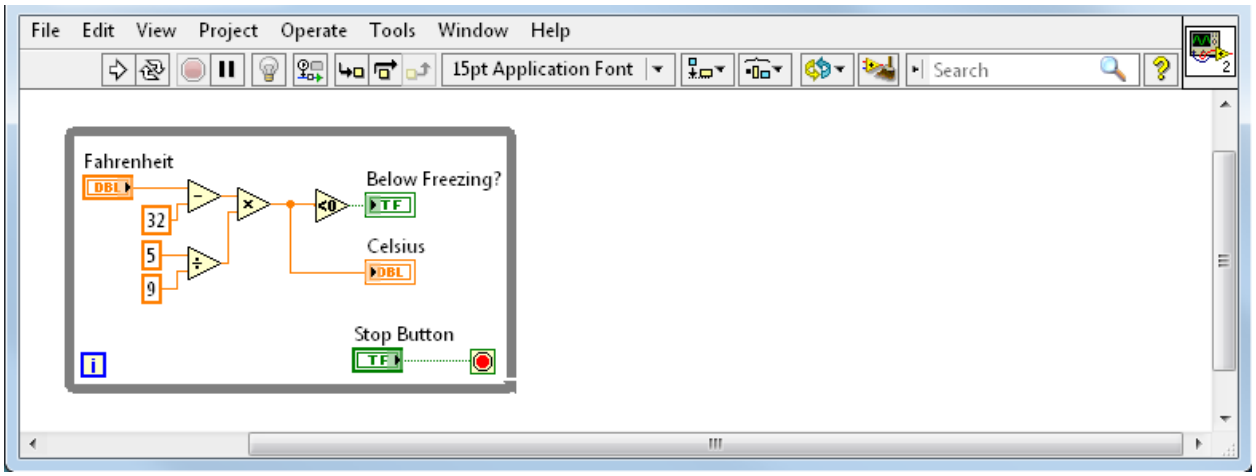
4. Now that you have written all of the code, execute the VI. Verify that the VI has been correctly wired by comparing your VI with the completed application on the next page.
   a. Switch to the front panel using **<Ctrl+E>**. Make sure you can see all the controls and indicators on the screen.
   b. Locate the run arrow on the toolbar and **left-click** it to begin program execution. 
   c. By default, the **thermometer** shows only values between 0 and 100 (you can change this by double-clicking on the scale and entering different values). For this exercise, enter Fahrenheit values to see the Celsius indicator update with the correct temperature value (between 0 and 100).
   d. While the code is running, try clicking the execution highlighting button on your block diagram toolbar.  Notice how the execution of your code slows down, and LabVIEW animates the flow of data around your block diagram. It is a great debugging tool, which allows you to visualize the movement of data and the order of function execution.
   e. When you are finished using the VI, you can terminate execution with the stop button.

5. Save and close the temperature conversion VI and project.

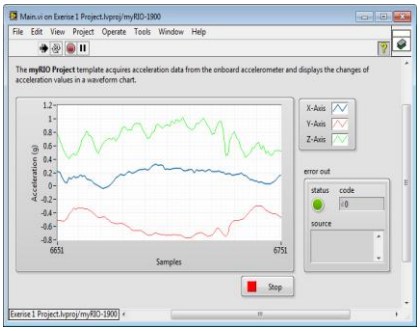
At the end of this exercise, you should feel fairly comfortable with the graphical programming approach. The LabVIEW cross-platform model makes it easy to create applications for many supported devices. In the next workshop exercises, you will use the same environment to create programs for the NI myRIO device. You will use the same structures, controls, indicators and functionality as you did here.

Completed Exercise 0 VI





Exercise 1: Create Your First myRIO Project



Create a new myRIO project using the project templates included in LabVIEW 2013 for NI myRIO. We will then explore the auto-generated myRIO code in detail.

Goal

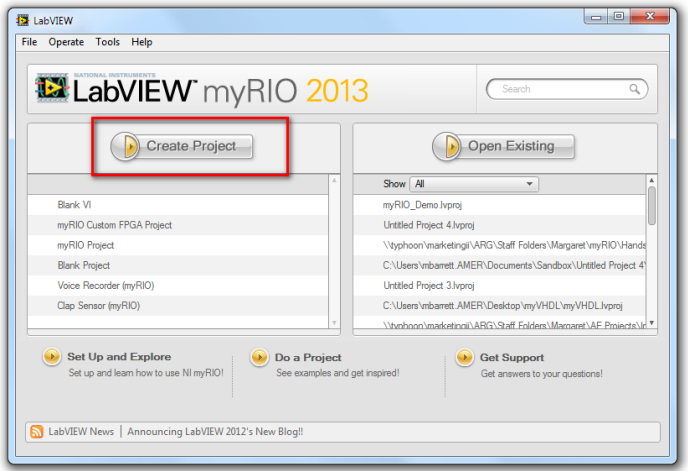
- Becoming familiar with Project Templates and myRIO development options
- Gain further experience of fundamental LabVIEW programming concepts
- Take your first sensor measurement with myRIO

Introduction: Running Real-Time Code on the NI myRIO Device

Real-time code runs on the processor built into the NI myRIO device. This code can receive data from and send data to the FPGA using FPGA I/O nodes, DMA FIFOs, and Express VIs that use the default FPGA "personality." The inputs and outputs of the NI myRIO expansion connectors and the NI miniSystems port communicate with the processor through the FPGA. The FPGA is covered in more detail later; for now, remember that the NI myRIO unit is shipped with an FPGA personality that is configured to pass all of the input data and output data to and from the connector pins and onboard devices (buttons, LEDs, accelerometer) to the processor that is running the real-time code.

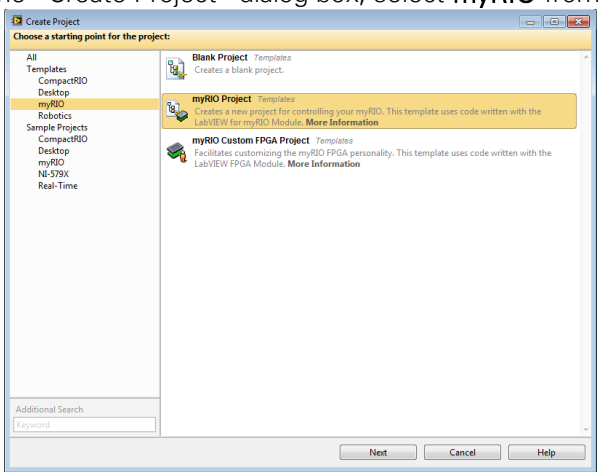
Using the default FPGA personality is the fastest way of creating simple stand-alone applications or prototyping code for more complicated projects.

1. From the LabVIEW Getting Started Window, select the **Create Project** button.



Creating a Project

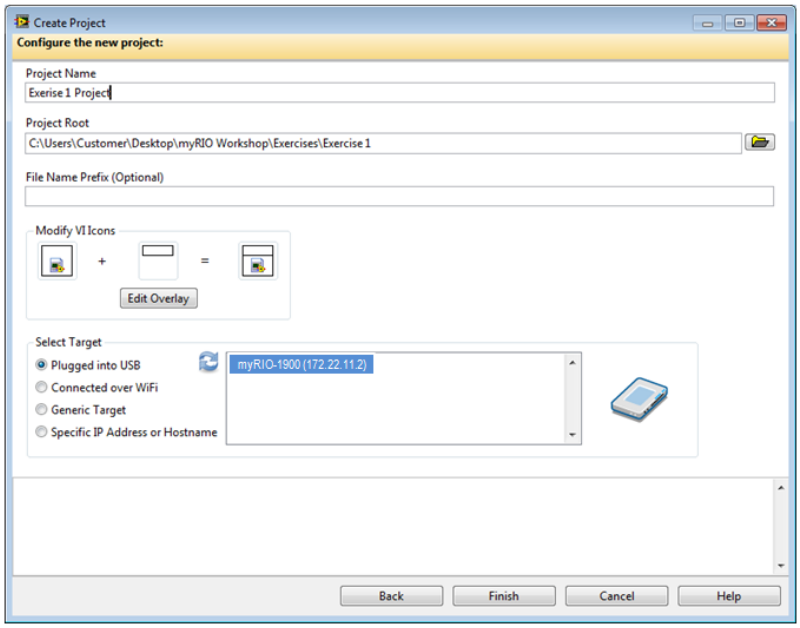
2. In the left pane of the "Create Project" dialog box, select **myRIO** from the **Templates** tree.



NI myRIO Project Templates

3. The right pane now shows three options: a standard Blank Project, a myRIO Project, and a myRIO Custom FPGA Project. The blank project is the same as the project created in Exercise 0.
    a. Use the myRIO Project template to create a project with the default FPGA personality. This template is useful for projects not requiring the extended functionality and configuration of the FPGA.
    b. Use the myRIO Custom FPGA Project template to set up a personalized FPGA definition on the NI myRIO device. For instance, all three connectors combined offer a total of eight PWM digital I/O lines. But if you want to connect more PWM-controlled devices or PWM inputs, you can reconfigure the FPGA to support PWM on more of the digital I/O lines. This applies to other communications protocols such as I^2C, SPI and so on.

4. Select the **myRIO Project** from the list and press the **Next** button.
5. Give the project a meaningful name and select an appropriate directory in the Project Root, such as the **myRIO Workshop\Exercises\Exercise 1** directory found on your desktop. Verify that the **Plugged into USB** radio button is selected and that the correctly named NI myRIO unit appears in the list to the right.



Configuring a LabVIEW Project

6. Select **Finish** when everything appears to be configured correctly.

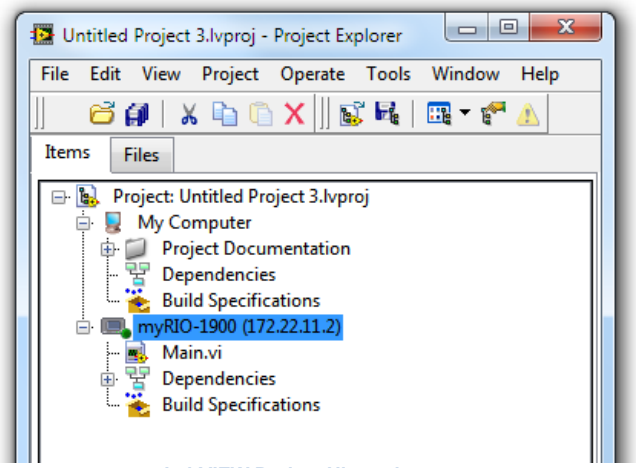
This NI myRIO template sets up the NI myRIO device as a target in the LabVIEW project. One critical concept to remember is that when a VI is targeted to the NI myRIO unit, the code in the VI is actually running on the device even if the front panel is being viewed on the development computer. This is called **interactive front panel mode**, which is intended for debugging and development only.

Usually, in the final form of the application, code is fully *deployed* to an untethered NI myRIO unit so front panel controls and indicators are inaccessible. However, you can use network-published shared variables or some form of wireless network streaming to send/receive data to/from a host machine for additional processing, display and storage.

The possibilities become really exciting at this point. Even without reconfiguring the FPGA, you can acquire data and make control decisions quickly on the RT processor of the NI myRIO unit. In the meantime, network communications allow the development computer to store and/or analyze data and even send higher level control decisions back to the NI myRIO device.

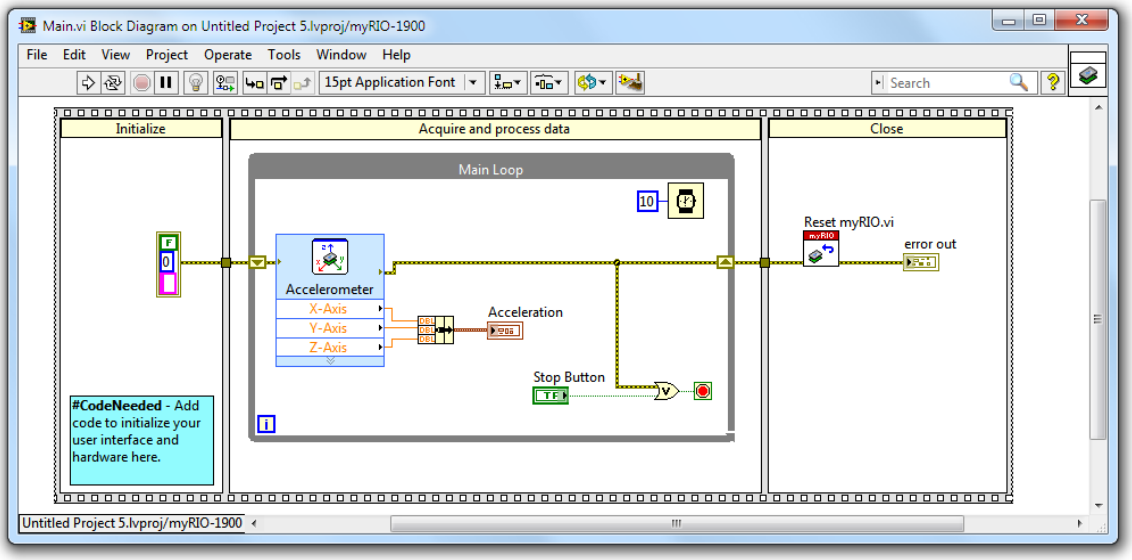
For this exercise, the NI myRIO unit is tethered to the development computer, but you still want to run the code on its processor. This means that you can interact with the front panel of the VI running on the RT processor of the NI myRIO device, and the NI shared variable engine takes care of the necessary network communications to handle the data transfer. For some testing and experimental setups, the interactive front panel mode is adequate and eliminates the need for more complicated data communication between the host and RT target.

The LabVIEW Project Explorer now shows two target devices: "My Computer" and "myRIO-1900 (xxx.xx.xx.x)." The NI myRIO target already contains a VI in the project called "Main.vi." This VI features some code to help you get started.



LabVIEW Project Hierarchy

1. Expand the myRIO-1900 (xxx.xx.xx.x) target within the LabVIEW Project Explorer, open Main.vi by **double-clicking**.
2. The front panel of the VI opens and should contain a waveform chart and a stop button.
3. Before running the VI, examine the block diagram. Switch to the block diagram by pressing **<Ctrl-E>**.
4. The structure surrounding the While Loop that looks like three classic film reel frames is called a **Sequence Structure**.



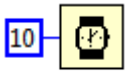
Sequence structures represent one way of forcing the execution of your code. The code in each frame of the structure (from left to right) must execute before the next frame can start. You can use **tunnels** to pass data across the frames.

Note: you could also achieve this behavior using appropriate dataflow programming techniques without the overhead of a sequence structure.

TIP: *Tunnels* are the little square dots that appear on the edges of loops, frames and case structures. You can create them automatically with the wiring tool by connecting two terminals separated by one of the aforementioned structures, or you can connect them manually by clicking on the edge of one of the structures. Tunnels pass data from a structure or loop only when all of the code in that structure has finished executing.

a. The "initialize" frame of the sequence structure is the first to execute. The only action occurring in this frame is an error constant cluster being created and passed to the next frame via a tunnel. But this would be an appropriate place to initialize any variables, user interface objects and hardware devices.
b. The "acquire and process data" frame receives the error cluster, and a While Loop (called "Main Loop") begins executing. Inside the main loop, an Express VI (the blue VI that says "Accelerometer") is used to acquire data from the accelerometer built into the myRIO device. Then the data read from the x-axis, y-axis, and z-axis are bundled into a cluster and passed into the acceleration waveform chart indicator (the chart on the front panel).

Note the "wait (ms)" VI in the upper-right corner.



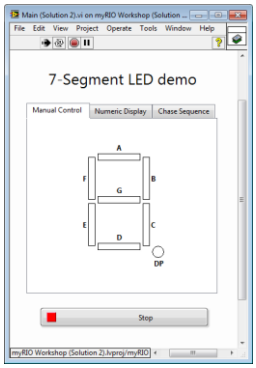
This VI forces the loop to execute every 10 ms, giving the loop a frequency of 100 Hz (assuming that the code in the loop can execute that fast).

Notice that the while loop will stop executing if the user clicks the stop button on the front panel OR an error occurs during executing.

c. When the "Main Loop" While Loop finishes executing, the error is passed out and through the frame of the sequence structure into the final frame. Use this frame to close any references on the NI myRIO device before the program exits. When adding your own custom functionality, you can use this frame to close I/O lines and save or delete data.

5. Now that you have analyzed the code structure, observe how it behaves at run time. Switch back to the front panel by pressing <Ctrl-E>. Click the run arrow to start this VI executing on the NI myRIO device
6. Once the VI has been successfully deployed to the NI myRIO unit, the waveform chart starts displaying the samples from the accelerometer. Move the NI myRIO device to see the accelerometer readings to change in real time
7. Press the stop button to exit the While Loop and finish executing the sequence structure
8. Save and Close this application to return to the LabVIEW splash screen

Exercise 2a: Create Real-Time Code to Run on the NI myRIO Device



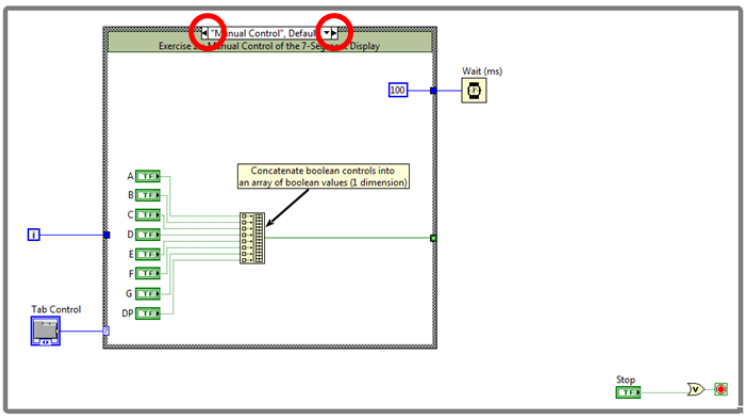
This exercise will guide you through completing a LabVIEW VI that will enable manual control of a 7-segment display

Goal

- Gain more experience in writing and deploying code to the NI myRIO device
- Understand how to develop simple Digital I/O tasks

---

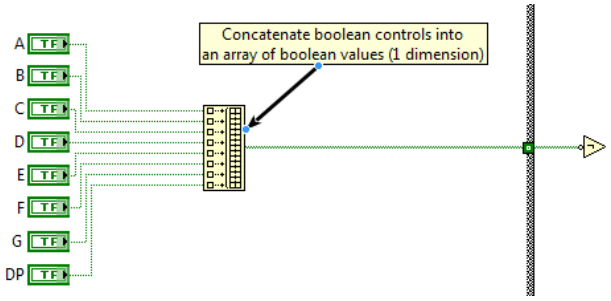
1. Return to the LabVIEW splash screen by closing any LabVIEW applications that you still have open. Now select **File»Open Project**, then navigate to *myRIO Workshop\Exercises\Exercise 2* on your desktop and open the **myRIO Workshop (Exercise 2).lvproj**
2. Open **Main.vi** under the myRIO-1900 (xxx.xx.xx.x) target device in the LabVIEW Project Explorer. Notice that, in the interest of time, the **front panel** has already been created for you. This exercise will lead you through completing the block diagram.
3. Open the **block diagram** (ctrl + E)
4. Ensure that the "Manual Control" sub diagram of the case structure is showing – use the left and right arrows at the top of the case structure to scroll through the different cases.



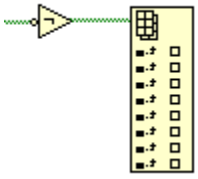
TIP: The front panel buttons, that represent the various display segments, are built into an array of Boolean values. This simplifies wiring (as you are now working with a single wire, rather eight individual wires) and reduces repetitive computations (LabVIEW allows you to perform processing on an entire array, rather than having to process each element individually).

Notice that the wire that transfers the array is thicker than the wires that transfer the individual, scalar data items

5. Because the 7-Segment display is active low (segments illuminate when a digital line is low), we need to invert the Boolean array to ensure that the user interface is intuitive.
    a. **Right-click** the block diagram and navigate to **Programming»Boolean»Not**
    b. Place the **Not** gate to the right of the Boolean (green) tunnel on the case structure border
    c. Wire the Boolean tunnel to the input of the **Not** gate
    d. This **Not** gate will invert every Boolean value within the array

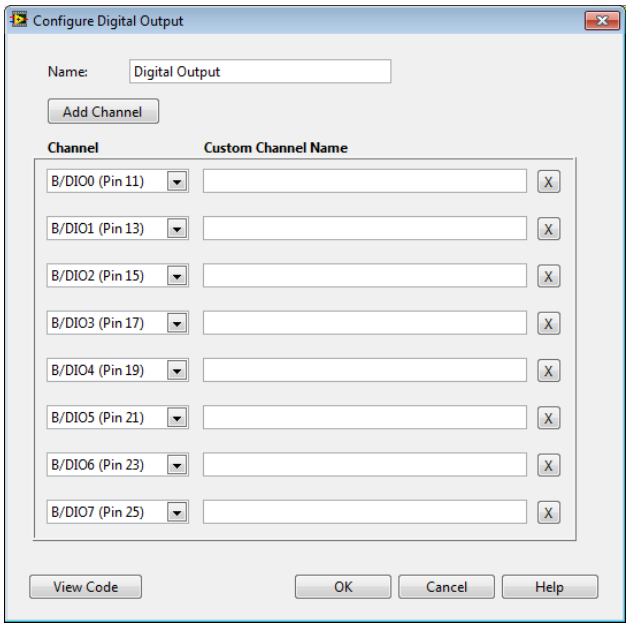


6. Now that we have inverted the Boolean array, let's disassemble the array back into its constituent elements.
    a. Right-Click and navigate to **Programming»Array»Index Array**
    b. Place the **Index Array** function to the right of the **Not** gate
    c. Wire the output of the **Not** function into the **Array** input of the **Index Array** function
    d. Hovering the mouse cursor over the Index Array function causes a small blue box to appear on the lower edge of the function. Drag the blue box down to expand the Index Array function to show 8 element outputs

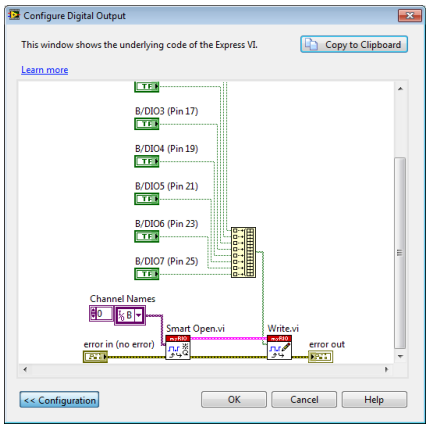


7. Now that we have access to the individual elements in the inverted Boolean array, we need to send the values to the myRIO digital lines. These digital lines are physically connected to the 7-segment display.
    a. **Right-click** and navigate to **myRIO»Digital Out**. Notice that the function has a pale blue boarder. This means that it is an Express VI, and can be configured by a simple menu-driven pop-up window
    b. Place the **Digital Output** function on the right of the **Index Array**. Once placed you will be greeted by the config window
    c. By default only one digital channel is shown: A/DIO0 (pin 11) – this is *Digital IO Line 0* on MXP connect A.
    d. Using the drop down Channel menu, change the channel to **B/DIO0 (Pin 11)**

e. Now that you have correctly selected the first digital line, add the additional seven channels. Simply click the **Add Channel** button seven more times. Each click will add a new channel. LabVIEW will automatically increment the number of the digital line. The config window should now look like the following image.

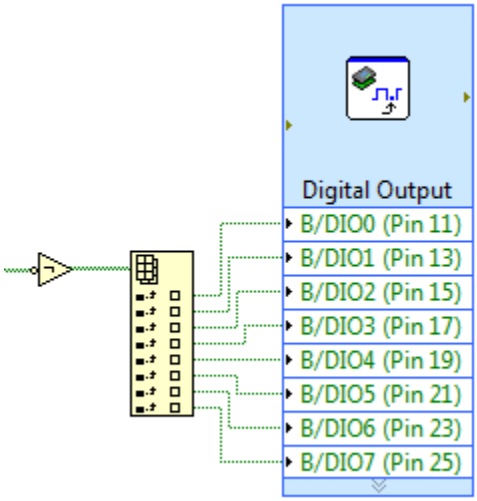


f. The ExpressVI automatically generate the low-level code required to handle this Digital task. If you would like to see what the underlying code looks like, click the view code button

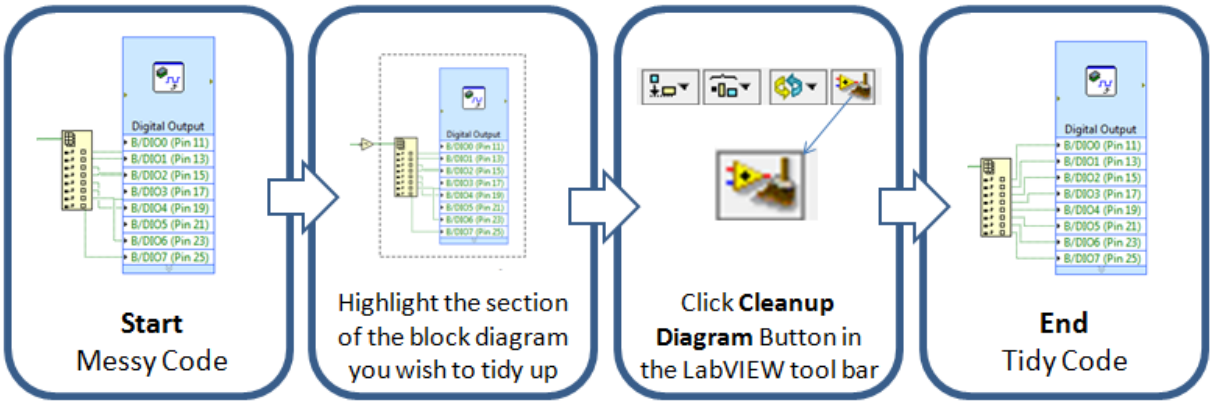


g. Once you are happy with the configuration, click **OK**.

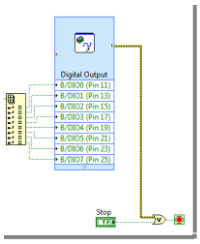
8. Back at the block diagram, we need to now connect the Boolean values from the Index Array to the Boolean input of the Digital Output task.
    a. The first element out from the **Index Array** function should be wired to **B/DIO0 (Pin 11)**
    b. The second element out from the **Index Array** function should be wired to **B/DIO1 (Pin 13)**
    c. Continue in this fashion, until all array elements are wired to their Digital Output lines



TIP: Don't worry if your wiring looks messier than the above image, LabVIEW includes an automatic cleanup feature that can help. Simply highlight the **Index Array** function and the **Digital Output ExpressVI** by drawing a box around them with your mouse. Now click the **Clean Up Diagram** button in the LabVIEW toolbar.



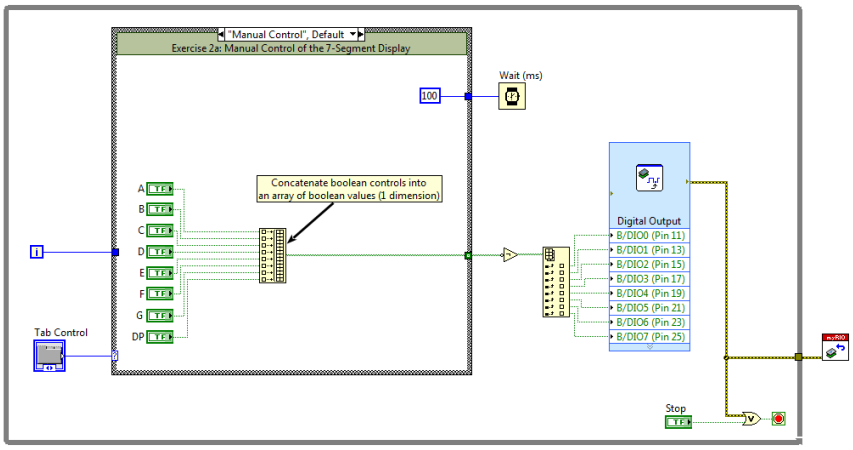
9. Adhering to good development practices, you should program the VI to gracefully shut down if an error occurs.
    a. To achieve this, wire the **error out** terminal of the **Digital Output ExpressVI** to the **Or** function connected to the **stop terminal** of the **while loop**. Now the **while loop** will stop executing the code if the user clicks the **Stop button** OR an error occurs.



10. Once the while loop has stopped executing, it is a good idea to reset the myRIO. This ensures that all I/O channels are reset back to known, default values.
    a. Right-click and navigate to **myRIO»Utilities»Reset myRIO.vi**
    b. Place the **Reset myRIO.vi** on the right of the **while loop**
    c. Wire the **error out** terminal to the edge of the **while loop**. Notice that LabVIEW automatically creates a tunnel for passing data out of the loop
    d. Wire the output of the error tunnel to the Error In input of the **Reset myRIO.vi**

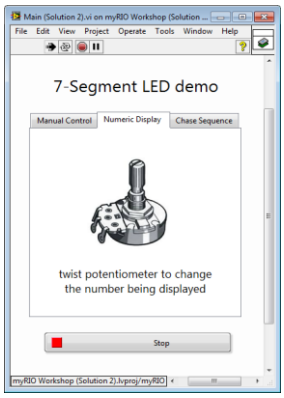
TIP: Using wires to connect block diagram elements enforces dataflow and ensures that functions will execute in a defined order. Here, the **Reset myRIO.vi** cannot execute until the **while loop** has finished and passed data (in this case the error cluster) to it. In this application, the error wire ensures that the **Reset myRIO.vi** will be the last function to execute.

11. Your block diagram is now complete. Please ensure your code looks similar to the block diagram below. Save your code.



12. Click the LabVIEW run button to deploy the code to the myRIO. Switch to the front panel and interact with the buttons on the **Manual Control** tab. The 7-segment display should illuminate accordingly
13. Stop the VI, but leave it open for the next section of the exercise

Exercise 2b: Create Real-Time Code to Run on the NI myRIO Device



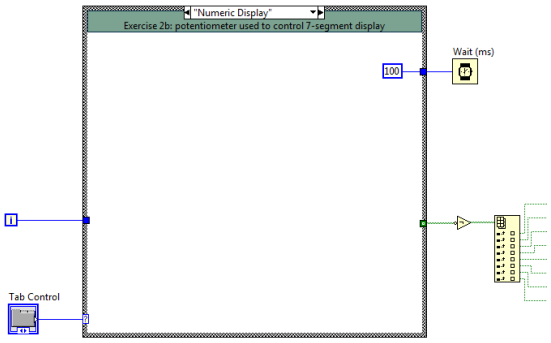
You will now extend the application functionality to allow a user to twist a potentiometer to display different numeric values on the 7-Segment display

Goal

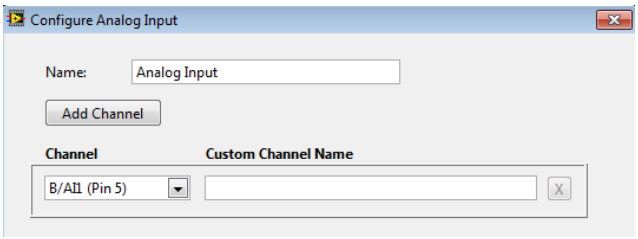
- Understand how to develop simple Analogue Input tasks
- Understand how to utilize custom code modules using SubVIs

---

1. Switch back to the block diagram, and select the **Numeric Display** case within the **case structure**. Notice that the case is currently empty.



2. Our first job is to acquire voltage readings from the potentiometer on the circuit board attached to MXP connector B. To do this we need to create an analogue input task.
   a. Right-click and navigate to **myRIO»Analog In**
   b. Place this function inside the **Numeric Display** case. A configuration window should now appear
   c. The myRIO features 12 different Analogue Input (AI) channels. Just like the Digital Out task we previously defined, we must first select which AI channel we want to acquire data from. Click the drop down **Channel** menu and select **B/AI1 (Pin 5)**

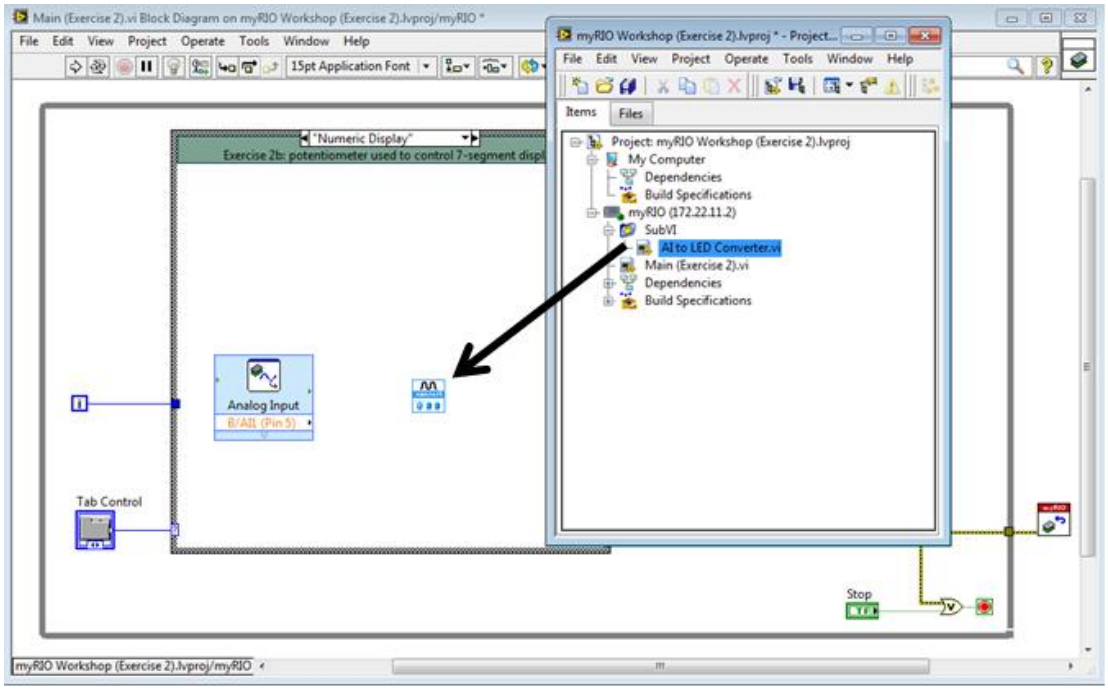


d. Feel free to click the **View Code** button to review the underlying, low-level code generated by the ExpressVI
e. Click **OK** to return to the block diagram

3. The **Analog Input ExpressVI** will acquire voltages from the potentiometer on the plug-in circuit board via the myRIO AI1 channel on MXP connector B. The potentiometer has been wired to produce a voltage between 0 and 3.3V. We must now convert that analogue reading into a digital pattern that will represent a numeric digit (between 0 and 9) on the 7-segment display.

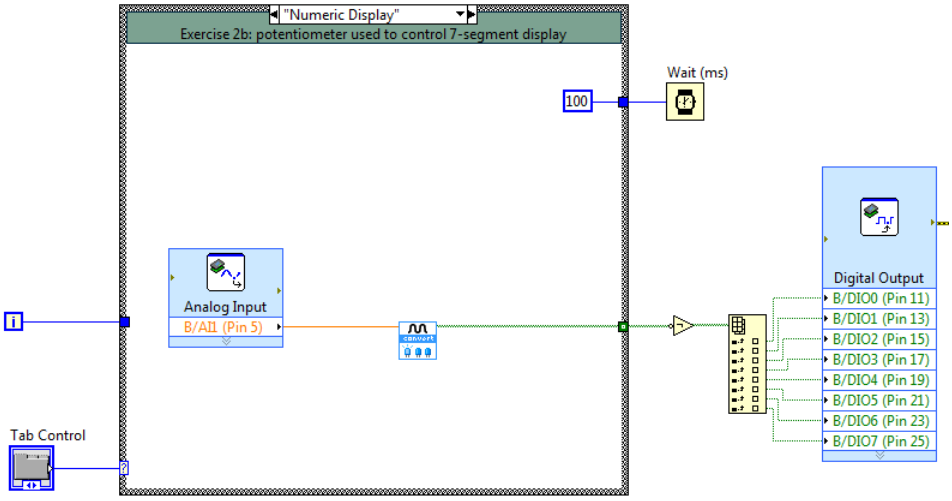
   To save time, we have pre-created a custom conversion function for you. This custom function has been developed as a **SubVI** (a VI that can be called by another VI)

   a. Leaving the **Main (Exercise 2).vi** open, bring up the **myRIO Workshop (Exercise 2).lvproj**
   b. Notice that, under the myRIO target, there is a folder called SubVI. Inside this folder is a VI called **AI to LED Converter.vi**. This is the function that has been pre-created for this exercise. Drag it from the project onto the block diagram of the **Main (Exercise 2).vi**



TIP: SubVIs are a great way of creating custom functionality in LabVIEW. Not only do they create neat, modularized code, but SubVIs are a great way of sharing custom functions between applications and even developers. For instance, you can head to the www.ni.com/community/ to access thousands of example applications and custom SubVIs that the vibrant LabVIEW community generates.

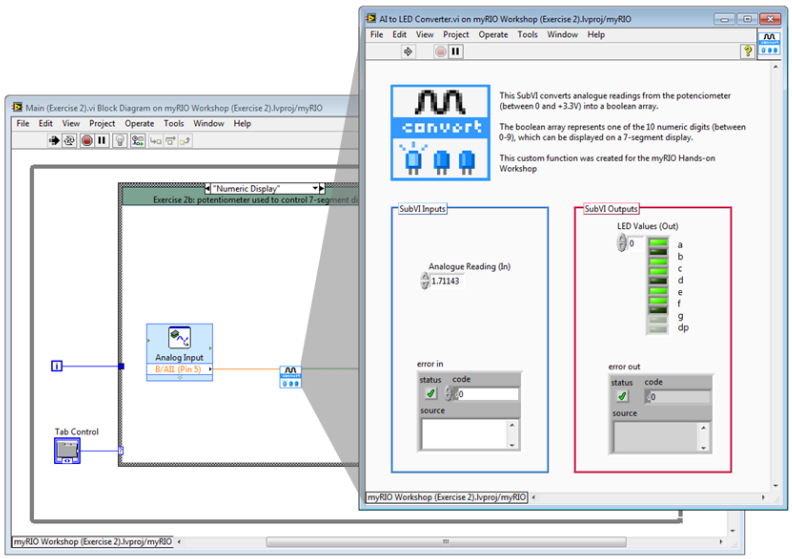
4. Wire the **B/AI1 (Pin 5)** terminal of the **Analog Input ExpressVI** to the **Analogue reading (In)** terminal of the **AI to LED Converter.vi**.
5. Wire the **LED Values (Out)** terminal of the **AI to LED Converter.vi** to the Boolean tunnel on the right-hand boarder of the **case structure**.



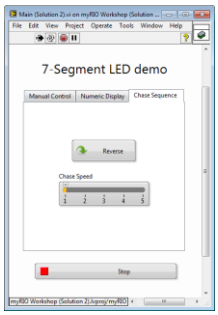
6. The new functionality is ready to deploy to the myRIO. Save your code and run the VI.

7. On the front panel, notice that the **Manual Control** functional still works as before. However, now when you select the **Numeric Display** tab, you should be able to interact with the circuit board potentiometer to control the numeric value shown on the 7-segment display.

TIP: While the code is running, switch to the block diagram and double click on the **AI to LED Converter** SubVI. This opens the front panel of the SubVI. Interact with the potentiometer to watch the digital pattern change accordingly. To learn how the conversion works, switch to the SubVI's block diagram (Ctrl + E) to review the code behind the SubVI.



Exercise 2c: Optional Challenges



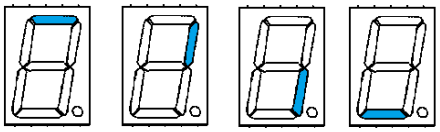
If you have finished the previous exercises before the rest of the group, use this time to further extend the application functionality by solving the following challenge exercises

Goal

- No instructions… time to think for yourself
- Gain experience in array manipulation and software timing in LabVIEW

Challenge 1

Create a rotating *chase sequence*, in which a single illuminated segment appears to move around the periphery of the 7-segment display.



Enable this mode when the user selects the **Chase Sequence** tab on the front panel

Challenge 2

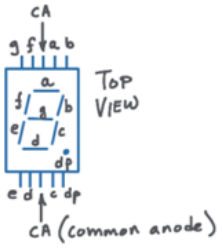
Make the speed of the *chase sequence* adjustable by the user

Challenge 3

Allow the user to reverse the direction of the *chase sequence* using a button on the front panel

---

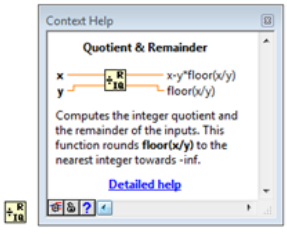
The following table recaps on how the individual segments are connected to the myRIO digital IO lines



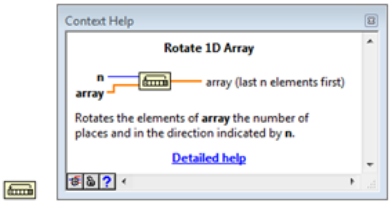
Pin	B/DIO	Terminal
a	0	11
b	1	13
c	2	15
d	3	17
e	4	19
f	5	21
g	6	23
dp	7	25

There are many ways of solving these challenges. However, you may wish to consider some of the following useful block diagram elements

The **iteration terminal** of the while loop, which increments on each iteration of the loop

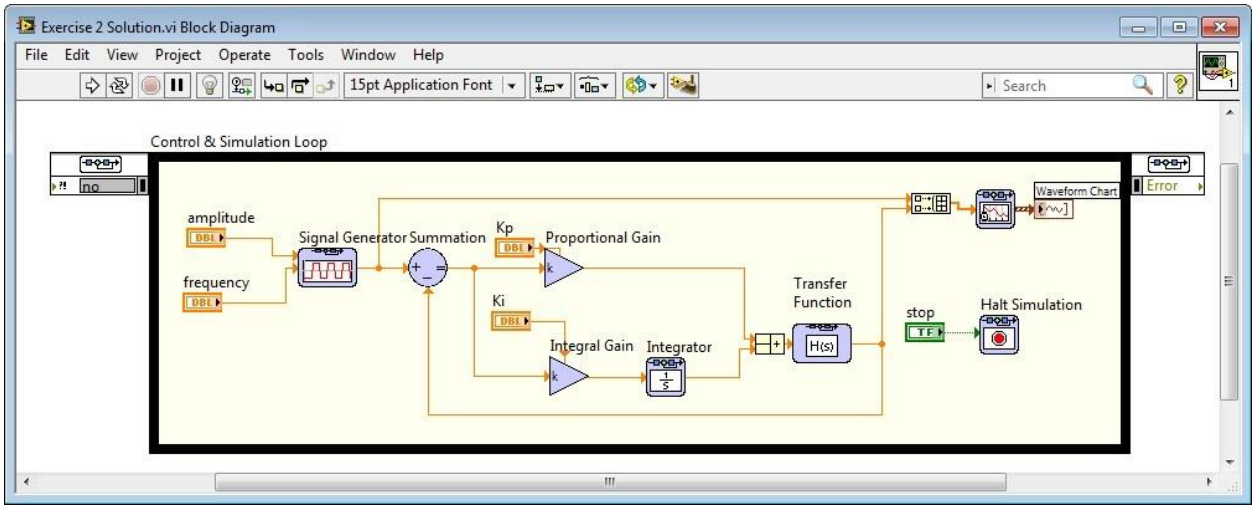


A 1D Boolean array with a single true value



LabVIEW Control Design and Simulation

Control design is a process that involves developing mathematical models that describe a physical system, analysing the models to learn about their dynamic characteristics, and creating a controller to achieve certain dynamic characteristics. Control systems contain components that direct, command and regulate the physical system, also known as the plant. Controls engineers, as well as students, concern themselves with both the modeling and implementation of these control systems, as a means of achieving a specified response from the plant. Plants can vary in complexity from a simple mass-spring damper system or RC circuit, up to the complex systems used in satellites. Control systems are traditionally modeled as block diagrams.

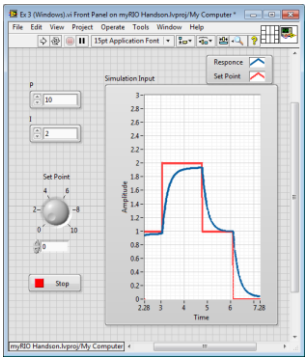


To create controls block diagrams like the one shown above, LabVIEW incorporates the Control Design and Simulation Toolkit. This toolkit, also called CD&SIM, adds several capabilities to LabVIEW. Specifically, CD&SIM enables developing models of many different plants directly in LabVIEW, including Transfer Function and State-Space models.

LabVIEW CD&SIM also facilitates simulation and control of these control plants. CD&SIM contains a Control and Simulation Loop, which is essentially a while loop with additional functionality for controls running in the background, including various ODE solvers. The simulation loop allows for building controllers and importing plant models optimised with the Control Design toolkit. Once a controller has been sufficiently analysed in simulation, the simulated plant can be easily removed and replaced with actual I/O points. This allows control systems to be analysed, simulated and deployed all from the LabVIEW environment.

LabVIEW Real-Time and NI myRIO are appropriate for to simulation and implementation of actual control algorithms. Loop rate, or how fast a control loop executes, is crucial for controls. Simple controllers may only require a loop rate in the tens of Hertz to meet their specification, but more complex controllers require code to execute at kilohertz speeds and above. Code running in LabVIEW Real-Time on a myRIO is able to execute quickly, and with much greater determinism than code which might be running on a Windows machine.

Exercise 3a: Develop a simulated control system that executes on the development machine



Create a simple PI controller to control the voltage across the capacitor in a simulated RC circuit. We will model the RC circuit using a simple 1st order transfer function

Goal

- Become familiar with the Control Design & Simulation Module for LabVIEW, which simplifies analysis and construction of plant and control models using transfer functions, state-space or zero-pole-gain

---

1. Save and close any currently open LabVIEW Vis and projects.

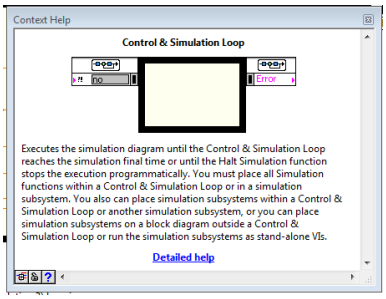
2. Once again, to save time we have provided a starting point for the application. Select **File»Open Project** and navigate to *myRIO Workshop\Exercises\Exercise 3* on your desktop, then open the **myRIO Workshop (Exercise 3).lvproj**

3. Open **Main (Exercise 3).vi** under the **My Computer** target in the LabVIEW Project Explorer. Because this VI is currently under the My Computer, it will execute on the development machine rather than the myRIO.

4. The front panel is already complete, but the block diagram has some missing elements. Switch to the block diagram

5. To save time, we have already constructed a simple PI (Proportional Integral) control algorithm. Notice that the code is surrounded by a thick black border – this is a Control & Simulation Loop. Think of it as a special while loop, that allows you to use the Control Design and Simulation programming interface.

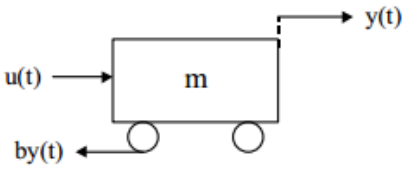
   If you would like learn more about any of these functions click **<Ctrl+H>** to turn on context sensitive help, then hover the mouse over the function of interest. Click on the **Detailed Help** hyperlink to be taken to the full LabVIEW help file.



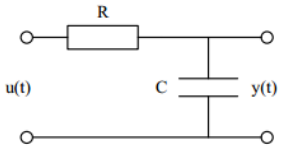
Because the control algorithm has already been implemented, we will now create a simple simulated plant using a 1st order transfer function.

TIP: A transfer function is a mathematical representation of the relationship between the input and output of a linear time-invariant system. Examples of 1st order systems include cruise control systems and resistive-capacitive (RC) circuits.

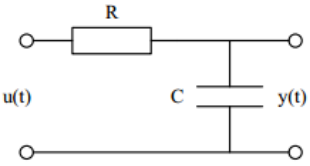
Example: 1st Order Mechanical system



Example: 1st Order Electrical system



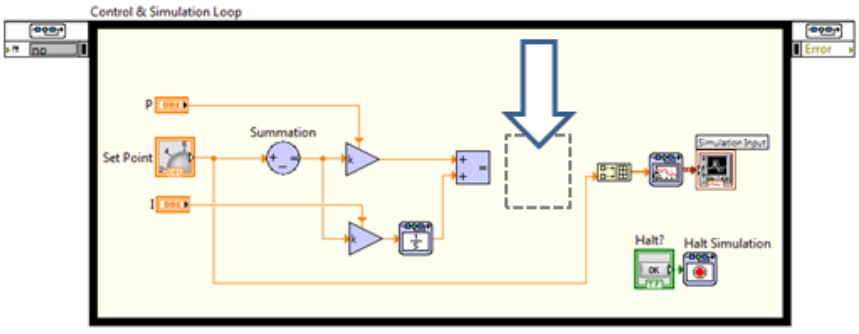
Our goal for this exercise is to model a simple RC circuit as a transfer function. The input to our circuit will be the voltage drop across the entire circuit - u(t) in the figure below. The output from our circuit will be the voltage across the capacitor - y(t) in the figure below. The following transfer function model of the circuit results:



$$H_C(s) = \frac{V_C(s)}{V_{in}(s)} = \frac{1}{1 + RCs}$$

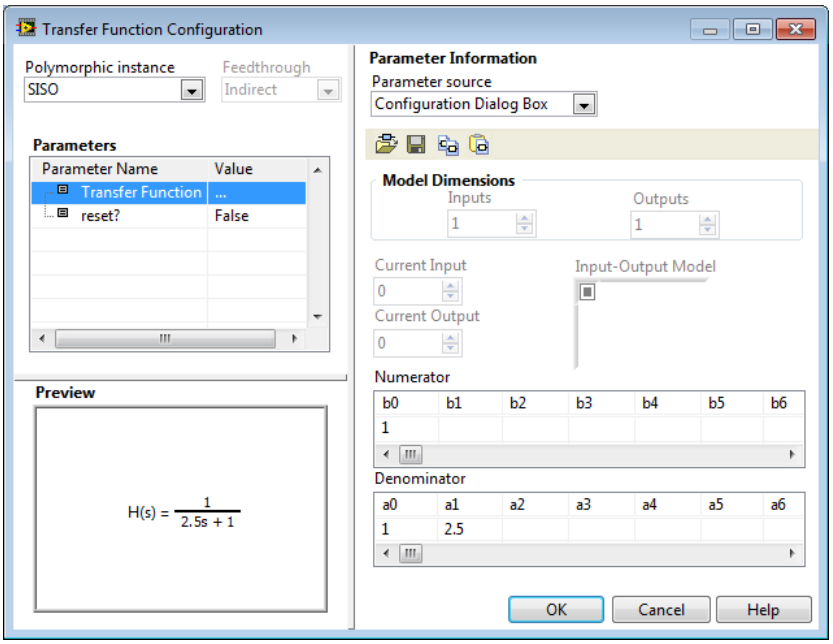
For our circuit, RC = time constant = 2.5. Let's implement this transfer function in LabVIEW.

a. Right-click and navigate to Control Design and Simulation»Simulation»Continious Linear Systems»Transfer Function.
b. Place the Transfer Function to the right of the square summation block



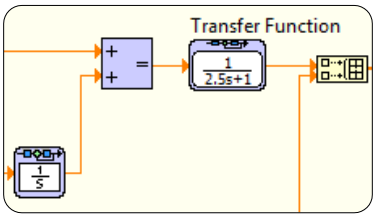
c. Double-click on the transfer function to bring up the configuration window

d. Referring back to the RC circuit transfer function we previously described, enter a value of 2.5 (the RC circuit time constant) as the value of the **denominator a1**. Notice that the transfer function preview updates automatically.
e. Ensure that the Transfer **Function Configuration Window** looks similar to the image below.

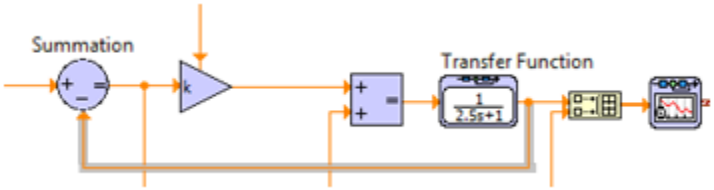


f. Note that we could use this config window to define a much more complex transfer function. Feel free to click the **Help** button to learn more about other configuration options.
g. Click **OK**

6. Wire the **result** terminal of the **summation block** to the **input u(t) terminal** of the **transfer function.** This simulates the input voltage to our RC circuit.
7. Wire the **output y(t)** of the **transfer function** to the top input of the **Built Array** function. This simulates the voltage across the capacitor.



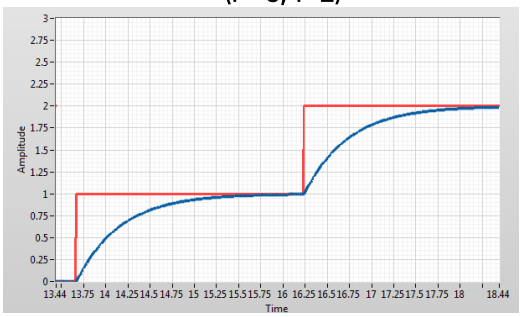
8. Now that we have implemented our simulated plant (transfer function), we just need to close the loop on the control system. Wire the **output y(t)** of the **transfer function** to the **subtract operand** of the circular **summation function**.



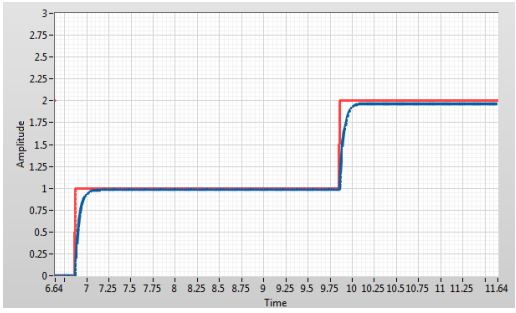
9. The application is now complete. **Save** the application, switch to the front panel, and click **Run.** Remember that this application is now running on the host processor.

   While the application is executing, try interacting with the **set point** control and experimenting with the **control gain** values.

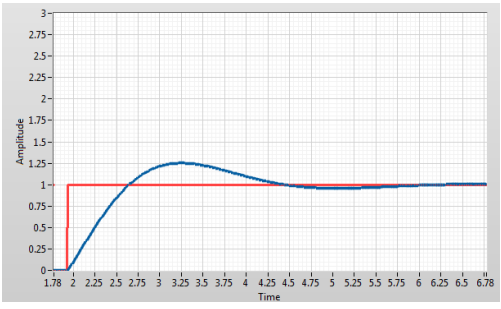
Using the modest default gains
(P=5, I=2)



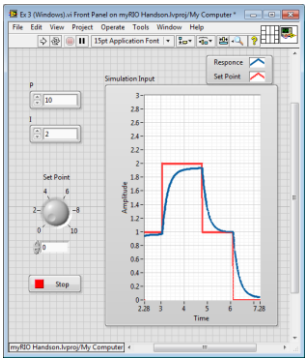
Increasing **P** causes a faster response, but can cause instabilities and oscillations
(P=50, I=2)



Increasing **I** decreases steady state error, but can increase overshoot and settling time
(P=4, I=10)



Exercise 3b: Develop a simulated control system on that executes on the myRIO processor



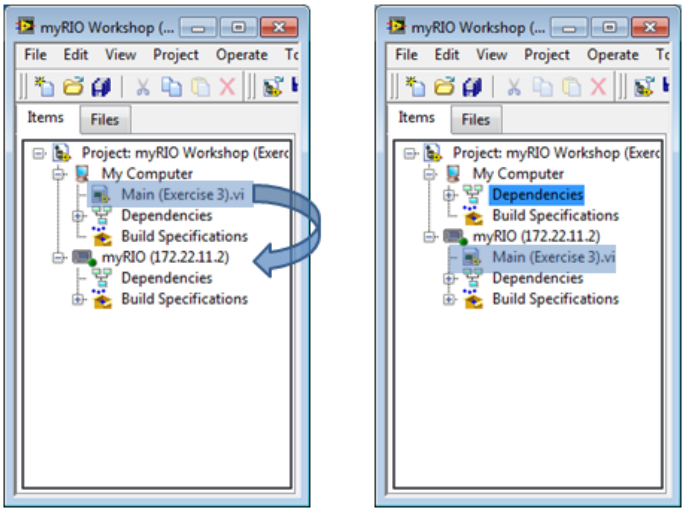
Deploy the control design and simulation code to myRIO, such that it runs with greater determinism.

Goal

- Experience the simplicity of moving LabVIEW code from a development machine to an embedded target

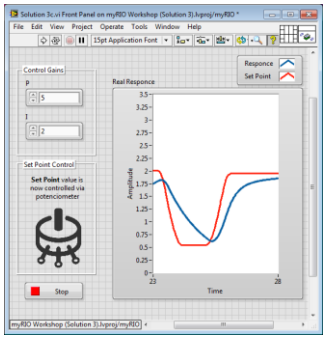
---

1. Save and close the **Main (Exercise3).vi**, and return to the **myRIO Workshop (Exercise3)** project.
2. As previously mentioned, the **Main (Exercise3).vi** is currently under the My Computer target. To deploy the code to the myRIO RT processor, you simply have to drag the **Main (Exercise3).vi** under the myRIO (xxx.xx.xx.x) target.



3. Double-click the **Main (Exercise3).vi** to open it up. Without any modifications to the code, you can now click the run button to deploy it to the myRIO.
4. You will see a pop-up window that displays the deployment progress. Once this has finished, the code will be running on the myRIO processor.
5. Stop the VI, but leave it open for the final part of the exercise

Exercise 3c: Move from Simulation to the Real World



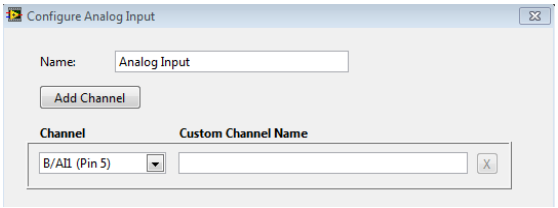
Now that the control system is running on the myRIO, we have easy access to real world I/O. In this exercise we will remove the simulated plant and replace it with a *real* RC circuit. We will also move set-point control from software to hardware.

Goal

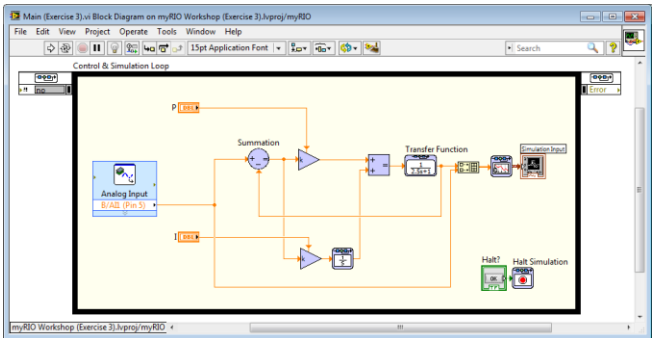
- Gain further experience with Analogue Input and Output, to see how simple it is to move from *Simulation* to *Real World experimentation* using LabVIEW

---

1. We will now remove the **set-point** dial control from the front panel, and replace it with an Analogue Input function, to allow us to control the voltage set-point of our circuit using the potentiometer.
    a. Switch to the block diagram of **Main (Exercise3).vi**
    b. Delete **Set Point** control terminal. Notice that this leaves a *broken wire*. A broken wire identifies that there is a coding error – in this case, because we are no-long sending any data into the wire.
    c. Right-click and navigate to **myRIO»Analog In**
    d. Place this function inside the **Control & Simulation Loop** where the **Set Point** terminal used to be. A configuration window should now appear.
    e. Click the drop down **Channel** menu and select **B/AI1 (Pin 5)**



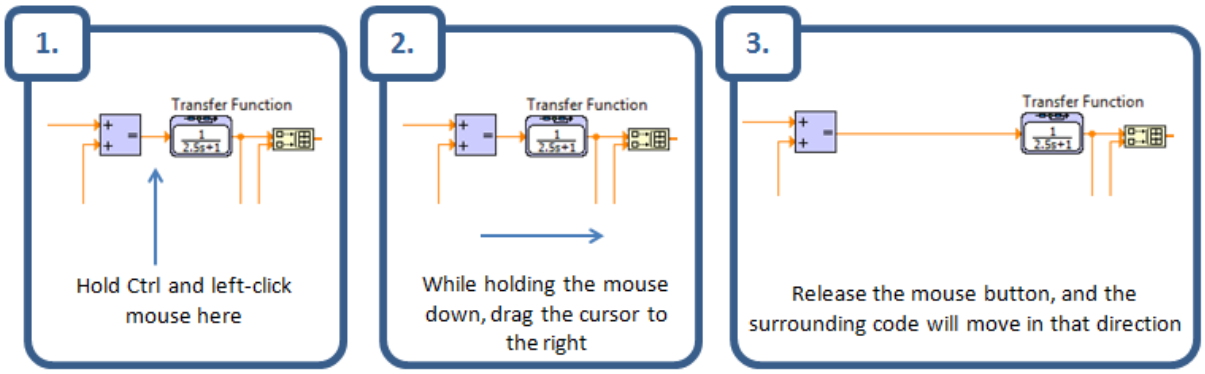
    f. Click **OK**
2. Back on the block diagram, wire the **B/AI1 (Pin 5)** output of the **Analog Input ExpressVI** to the broken wire. Notice that the wire now becomes solid again, illustrating that we have resolved the previous error by attaching a data source to the wire.



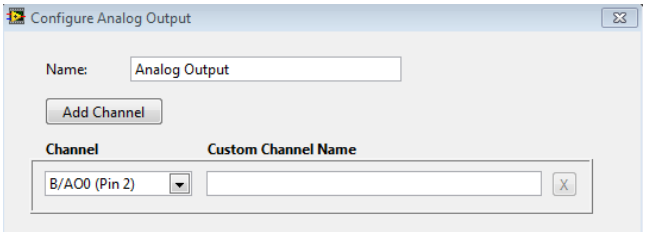
3. Switch back to the front panel and **click** the **run** button. You should now be able to control the set-point of your simulated RC circuit using the potentiometer on the plug-in circuit board.

4. The final stage of the exercise is to replace the simulated plant (transfer function model of our RC circuit) with Analogue I/O, which will enable us to interact with a real plant (physical RC circuit)
    a. Switch to the block diagram
    b. In order to replace the **transfer function** with **Analog In** and **Analog Out** functions, we need to make some space in the block diagram. Rearrange the existing code to increase the distance between the square **summation** block and the **transfer function**

TIP: LabVIEW includes a short-cut that simplifies making space on a block diagram. Simply hold Ctrl then click & hold the mouse in the area of the block diagram you wish to make space. Then drag the mouse in the direction you want to move the existing code. Finally, release the mouse button.

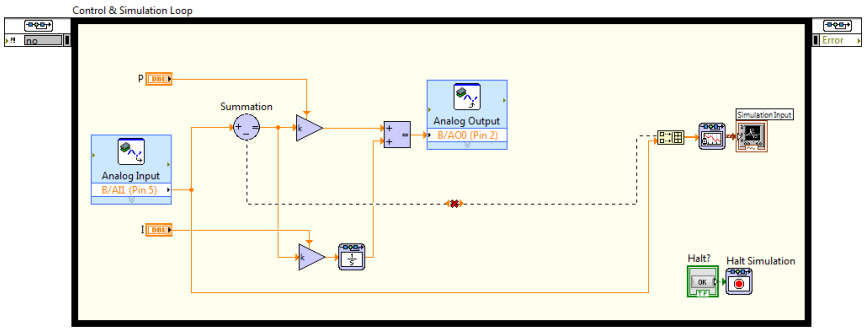


    c. Delete the transfer function
    d. **Right-click** and navigate to the **myRIO»Analog Out**
    e. Place this function to the right of the square **summation** block
    f. In the resulting configuration window, click the **Channel** drop-down menu and select **B/AO0 (Pin 2)**

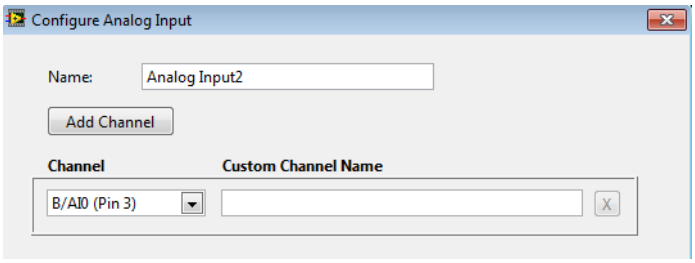


    g. **Click OK**

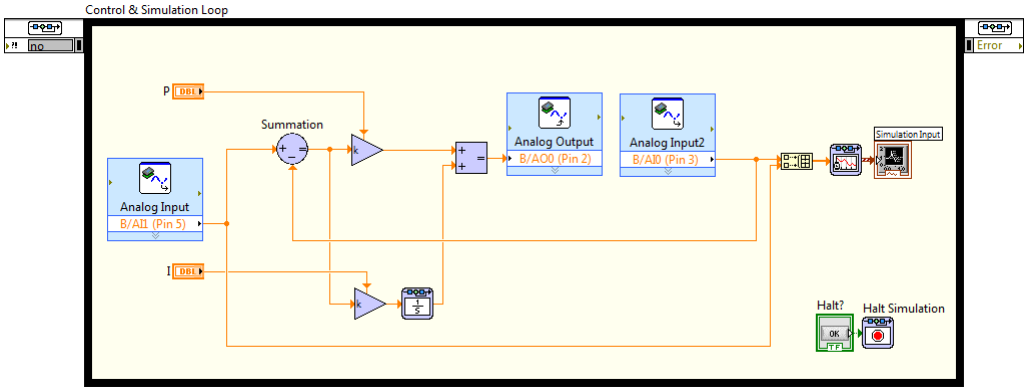
h. Wire the **Result** terminal of the square **summation** block to the **B/AO0 (Pin 2)** of the **Analog Output ExpressVI**



5. We are now sending the control value to the RC circuit using Analogue Output. Finally, we need to close the loop on the control system by measuring the voltage dropped across the capacitor.
   a. **Right-click** and navigate to the **myRIO»Analog In**
   b. Place this function to the right of the **Analog Output** ExpressVI
   c. In the resulting configuration window, click the **Channel** drop-down menu and select **B/AI0 (Pin 3)**



   d. Click **OK**
   e. Back on the block diagram, wire the **B/AI0 (Pin 3)** terminal of the **Analog Input ExpressVI** to the currently broken wire that connects the top terminal of the **Index Array** function to the **subtract operand** of the circular **summation** block.



6. The code is now complete. Save the VI.
7. Switch to the front panel and enter the following values into the Control Gains controls:

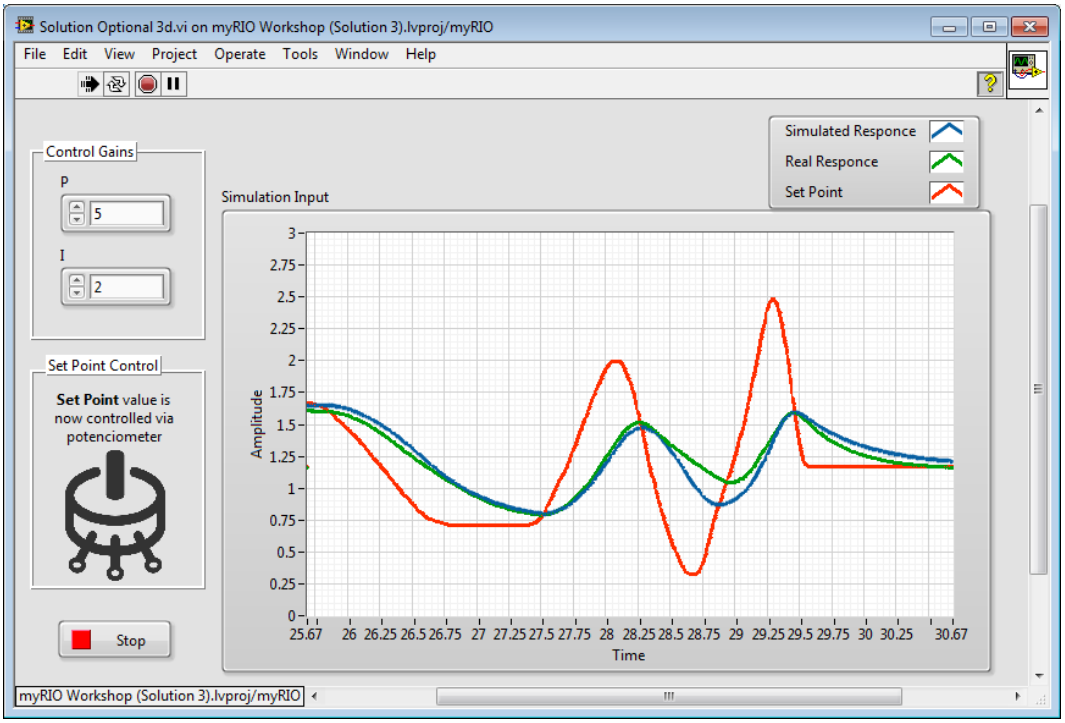
P=5 I=2

8. Click the **run** button.

We are now controlling a real plant; the simple RC circuit on the plug-in circuit board. Try interacting with the potentiometer to adjust the set-point, and then watch the voltage dropped across the capacitor change accordingly. Try experimenting with the control gains and review the changes in control response

Exercise 3d: Optional exercise

If you have successfully completed Exercise 3c before the rest of the attendees, try modifying the code to simultaneously display the responses of the simulated plant and the real plant on the same graph.



TIP: A simple way of achieving this would be to include two separate control algorithms running within the same **Control and Simulation loop**

Field-Programmable Gate Arrays

FPGAs are silicon chips that operate around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. You can configure these logic blocks to process all the basic logic gates that come in standard ICs and, in many cases, more complex logic. The first commercially viable FPGAs were invented by Xilinx cofounders Ross Freeman and Bernard Vonderschmitt in 1985. The clear advantage of the FPGA is that you can modify the hardware-level logic without acquiring or modifying physical hardware and ICs. This means you can design custom logic for your system and reconfigure the FPGA to execute the logic at run time. You develop the FPGA "personality" in a software environment and then implement the personality on the silicon level. Due to the nature of FPGAs, individual sections of the chip that are independent of one another can execute in true parallel.

True parallelism means that tasks running on the FPGA are truly independent and highly deterministic. Determinism is critical in controls, robotics, and other mechatronics applications (a typical FPGA system can be designed to react to digital inputs in as little as 25 ns [40 MHz], and sometimes faster). Some examples of LabVIEW FPGA applications include intelligent DAQ, ultrahigh-speed control, specialised communication protocols, CPU task offloading to save the processor for more complicated analysis, complex timing and synchronization, and hardware-in-the-loop testing. With the ability to react to changes in data so quickly, the FPGA help you design industrial-quality systems and experiments without huge investments in actual industrial equipment.

With the LabVIEW FPGA Module, the process of programming FPGAs is completely graphical and features fully automated compilation. You design the FPGA personality using VIs from the LabVIEW FPGA Module palette in LabVIEW. Then when you are ready to compile, LabVIEW generates the intermediate VHDL files required by the Xilinx compiler, starts the compiler, and passes the files to it. This produces a bitfile that is then downloaded onto the FPGA chip's flash memory to be read at run time. When the FPGA runs, it reads this bitfile and then reconfigures itself per the file's instructions.

Exercise 4: Exploring the NI myRIO FPGA Shipping Personality

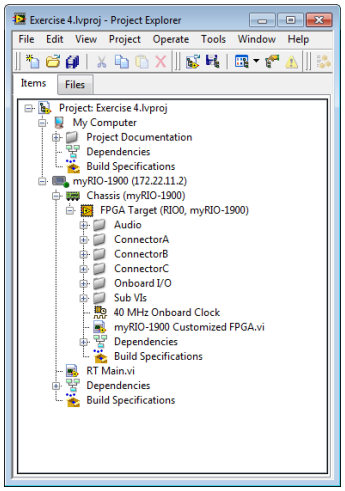
Create a new myRIO project using the Custom FPGA project template. Then explore the default FPGA personality and consider how you would modify it for you application

Goal

- Become familiar with the look and feel of FPGA programming in LabVIEW
- Consider how your application could benefit from the power of FPGAs

---

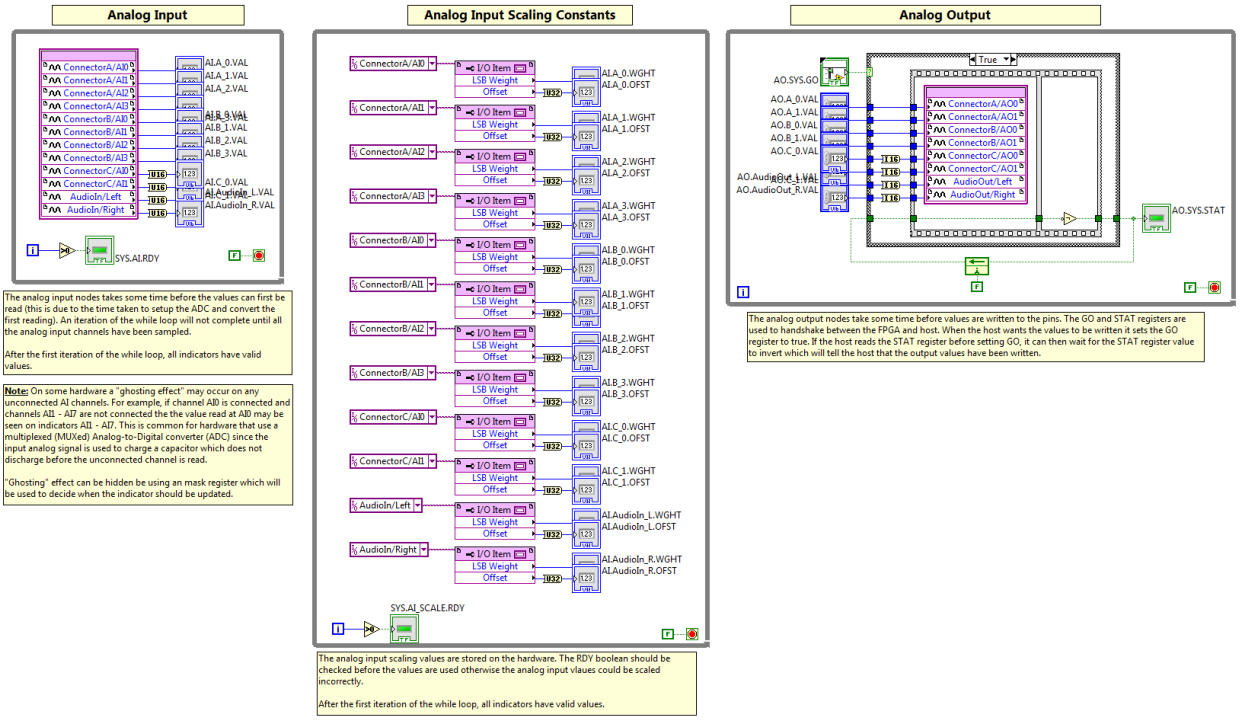
1. Save and close any open applications to return to the LabVIEW Getting Started window.
2. Select **Create Project**.
3. In the Create Project window, choose the starting point for the project from the tree in the left pane. Under the templates category, select **myRIO**.
4. In the right frame, select **myRIO Custom FPGA Project**. The *more information* link in the description of the Custom FPGA Project contains a description of how to set up the project.
5. Select **Next** to configure the project.
    a. Give the project an appropriate name (eg. Exercise 4 Project), select a location to save the project (eg. myRIO Workshop\Exercises\Exercise 4) and select the NI myRIO device plugged into USB.
6. Select **Finish** to create the project.
7. When the LabVIEW Project Explorer loads the new project, my computer and myRIO – 1900 (xxx.xx.xx.x) appear as targets.
8. Expand the NI myRIO target and notice the new "chassis" tree. This tree appears on devices that contain a targetable FPGA. Logically, the FPGA target falls under the chassis tree, as the FPGA is integrated within the myRIO chassis.



The FPGA's primary role is handling all of the I/O on the NI myRIO device, so you can find the I/O under this FPGA target as project-unique items. The hierarchy of the FPGA target is organised into folders so the user can tell where each I/O node is located on the physical device. The two MXP connectors, MSP connector, and the onboard I/O all have unique folders. The folders are further subdivided into the I/O type (digital/analog) and the physical banks of I/O. You can drop these unique I/O items into an FPGA VI to read or write to that location. Any control or indicator on the front panel of the FPGA VI can be written to or read from, respectively, in the RT VI.

9. Open "**myRIO-1900 Customized FPGA.vi**" to view the shipping code placed on the FPGA.
10. Explore this code.

The FPGA target has a 40 MHz clock configured in the LabVIEW project. Any VI created in this portion of the tree is assumed to be an FPGA VI, and LabVIEW automatically restricts the functions and data types allowed in the VI. You can either create a brand new FPGA VI or modify the default FPGA VI.



The default FPGA VI, that ships with NI myRIO, exemplifies safely handling I/O data from the FPGA and preparing it to be passed up to the Real-Time VI executing on the ARM processor. The default FPGA personality handles all of the inputs and outputs on both the MXP connectors and the MSP connector, including PWM, UART, I²C, SPI, and quadrature encoder I/O. This default FPGA VI is more than sufficient for basic applications that students work with (consult the NI myRIO user manual for the default pinouts of the connectors).

To vastly simplify the architecture of projects, use the default FPGA personality and only program Real-Time Host VIs (to be executed on the ARM processor on the NI myRIO device) and Windows VIs (to be executed on the development machine). However, advanced projects can benefit from the power, determinism and co-processing enabled by a customized FPGA VI.

Resources and Next Steps

Accessory Kits

You can purchase accessory kits to facilitate the rapid realization of projects with the NI myRIO device. View kit pricing and availability at ni.com/myrio/accessories.

The Starter Kit includes a battery holder, a protoboard (which plugs into the MXP ports), a wire kit, LEDs, switches, a speaker, a microphone, a DC motor, encoders, a Hall effect sensor, and a piezo element. This kit is intended to get a student started quickly building simple circuits and expanding their understanding of digital and analog inputs and outputs.

The Mechatronics Kit includes all of the Starter Kit contents plus DC motors with encoders, an ambient light sensor, an ultrasonic range finder, servo motors, a compass, a DC motor driver (H bridge), an infrared proximity sensor, a triple axis digital output gyro, and a triple axis accelerometer. This kit is aimed at students who are building basic robots or mechatronics systems for their applications.

The Embedded Systems Kit includes a UART LCD screen, a digital temperature sensor, digital potentiometers, a barometric pressure sensor, a keypad, an LED matrix, an RFID kit, and an SPI EEPROM. This kit is intended for students building systems to run stand alone from the host computer.

NI myRIO Project Essentials Guide

The NI myRIO Project Essentials Guide provides a multimedia learning resource for students who are completing projects at all levels. This resource was designed to help students get started interfacing NI myRIO to a broad variety of sensors, actuators, and other components using LabVIEW software. The guide breaks down wiring, I/O requirements, device theory, and programming for over 20 different devices common to NI myRIO projects.

ni.com/white-paper/14621/en

C Support for NI myRIO

NI myRIO is based on NI reconfigurable I/O (RIO) technology, which gives you the ability to program both a processor running a real-time OS and a customizable FPGA. In addition to LabVIEW software, the NI myRIO processor is fully programmable in C or C++ using the default shipping personality placed on the FPGA.

ni.com/myrio/c-support

NI myRIO Community

Connect with other NI myRIO users, view projects and download example code.

ni.com/community/myrio

Next Steps

National Instruments is invested in the success of the professors and students using our products. Now that NI myRIO has been introduced and an idea of the potential of this device is understood, the next step is to learn more about LabVIEW and RT systems. National Instruments offers many avenues for learning how to successfully configure hardware, write code, and deploy it. The two most accessible are online resources intended to teach the user the basics of LabVIEW called "Learn LabVIEW" and "Learn RIO." You can access these resources from ni.com/students/learn free of charge. Both "Learn LabVIEW" and "Learn RIO" contain video tutorials and simple exercise sets that can rapidly accelerate a user's learning curve. These two modules are strongly recommended for all NI myRIO users.