NATIONAL INSTRUMENTS

NI myRIO

Design Real Systems, Fast

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

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 and simple 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.

Introduction to 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. If you already have some experience with LabVIEW, you can program the NI myRIO device. The following section introduces the graphical nature and capabilities of the LabVIEW programming language. It also examines the Getting Started window and the LabVIEW Project Explorer and concludes with an exercise that results in a simple LabVIEW program.

LabVIEW is a graphical programming environment that students can use to quickly develop applications that scale across multiple platforms and OSs. The power of LabVIEW is in its ability to interface with thousands of devices and instruments using hundreds of built-in libraries and prebuilt VIs 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 contain 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 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.

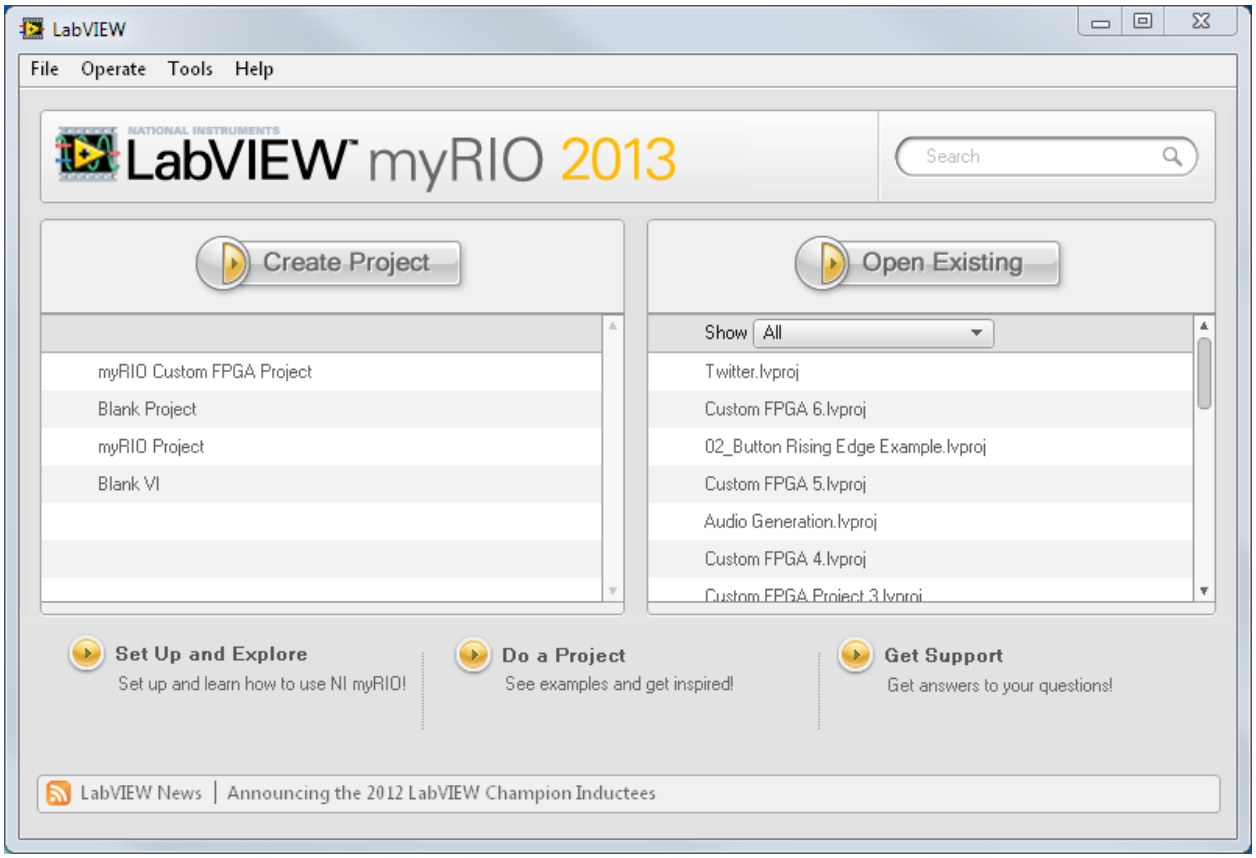
In Windows 7, you can launch LabVIEW from the Start menu, either by navigating to the LabVIEW 2013 shortcut or by searching for "LabVIEW."

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 **LabVIEW»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 employees 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.

The LabVIEW Project Explorer

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

For those of you accustomed to other integrated development environments (IDEs), the LabVIEW Project Explorer will seem familiar and its usage will be mostly the same. LabVIEW is a cross-platform language with comprehensive support for its add-ons and libraries. You can write most VIs while targeting them to the development machine and easily retarget them to an NI device such as NI myRIO because these targets also 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. Typically, a project is created in a directory and all of its source code is placed in the same directory with no folder system, even if the project view is highly organized. 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.

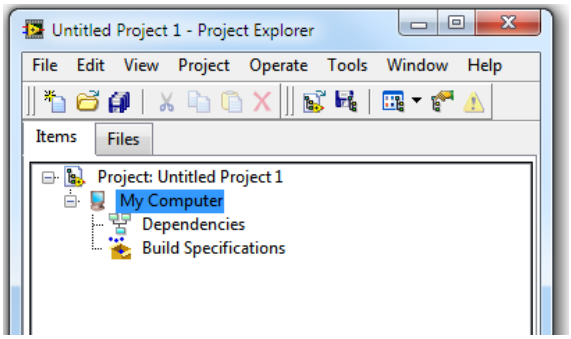
Complete the following steps to create a new project:

1. Select the **Create Project** button on the LabVIEW Getting Started window.
2. The Create Project window opens and offers several options. Explore all of the options by either creating a project or using the *More Information* link associated with any project template you are interested in.
3. For now, select the **Blank Project** option as the starting point for the project.
4. Then select the **Finish** button. This creates an empty project and opens the LabVIEW Project Explorer.

To save a new project:

1. From the LabVIEW Project Explorer window, select **File»Save**.
2. A save dialogue box opens and requests a destination directory and project name. Choose an appropriate location to save the project and give it a meaningful name.

Now the LabVIEW Project Explorer window is open and a brand new project 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 organized in a hierarchy, while the files page shows all of the items in the project with associated files on the disk.



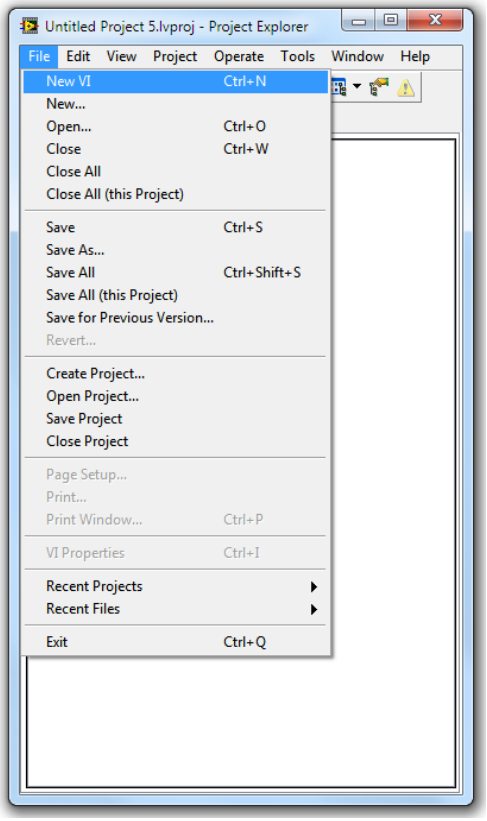
LabVIEW Project Explorer

In the items page 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 target 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 deploying applications to embedded systems. While this workshop focuses on simple VIs, you need to understand the LabVIEW Project Explorer once you are developing more involved code. For now, however, the project is pretty void and doesn't accomplish much. Build a simple VI in this project to dive into programming in LabVIEW. **Keep this project open so you can use it in the next section.**

Exercise 1: Building a LabVIEW VI in Windows

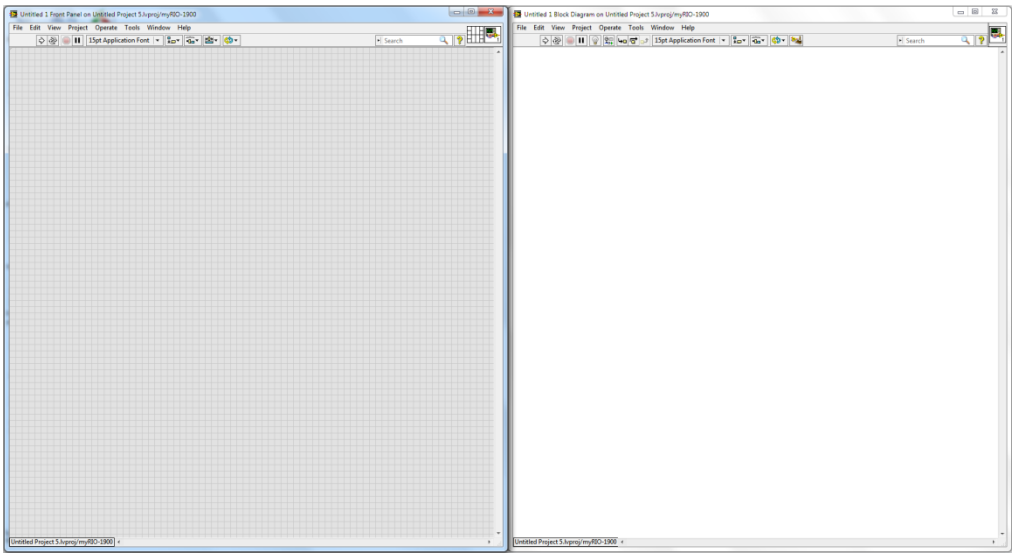
To learn about the LabVIEW graphical programming environment, create a basic piece of code that illustrates some of the properties of dataflow programming. In this exercise, program a simple temperature conversion VI to change Fahrenheit units to Celsius units.

To enter the programming portion of LabVIEW, you must create a new VI by selecting **File»New VI** in the LabVIEW Project Explorer.



Creating a Blank VI

This operation creates a blank VI under the My Computer target in the project you created a few moments ago. This new blank VI consists of two windows: the front panel and the block diagram.



Front Panel and Block Diagram

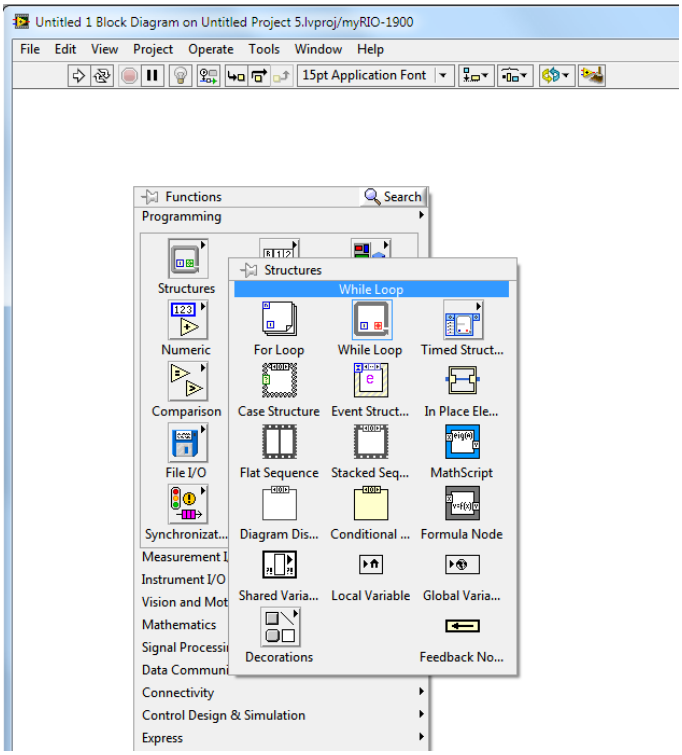
Save the new VI by selecting **File»Save** from either the front panel or the block diagram. When the save dialog appears, give the VI a meaningful name. You should save VIs in the same folder as the project they are going to be used in. You can save VIs that you want to share between multiple projects in one of the project's locations or in a completely different directory. The LabVIEW Project Explorer keeps track of them and asks you to locate missing VIs if a file is misplaced or deleted.

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

1. This code runs continuously, and, just like text-based languages, LabVIEW uses loops to run tasks continuously. You can terminate loops with a button on the front panel or in the traditional manner of 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, and use a button on the front panel to control termination.
    a. Switch to the block diagram by either selecting the block diagram window or pressing <Ctrl-E> from the front panel (<Ctrl-E> switches back to the front panel; use this shortcut to quickly swap between these two windows).

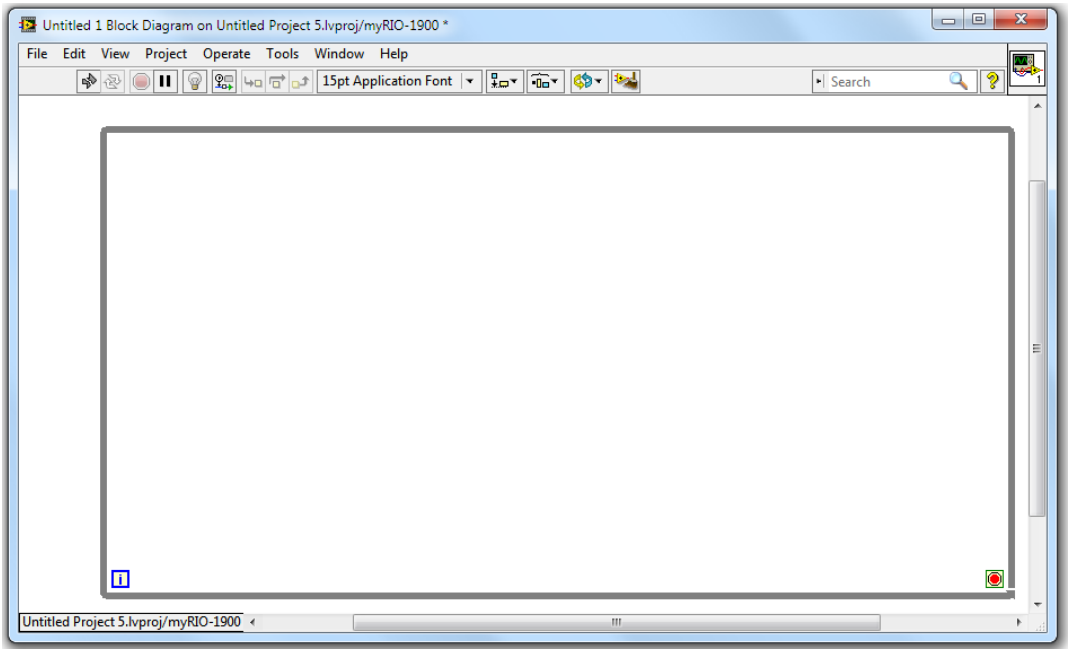
TIP: Learning keyboard shortcuts greatly accelerates navigation through the LabVIEW environment. Also, they make you look like a LabVIEW wizard!

    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. Locate the labels for each of the subpalettes. Navigate to **Programming»Structures»While Loop** and select the While Loop.



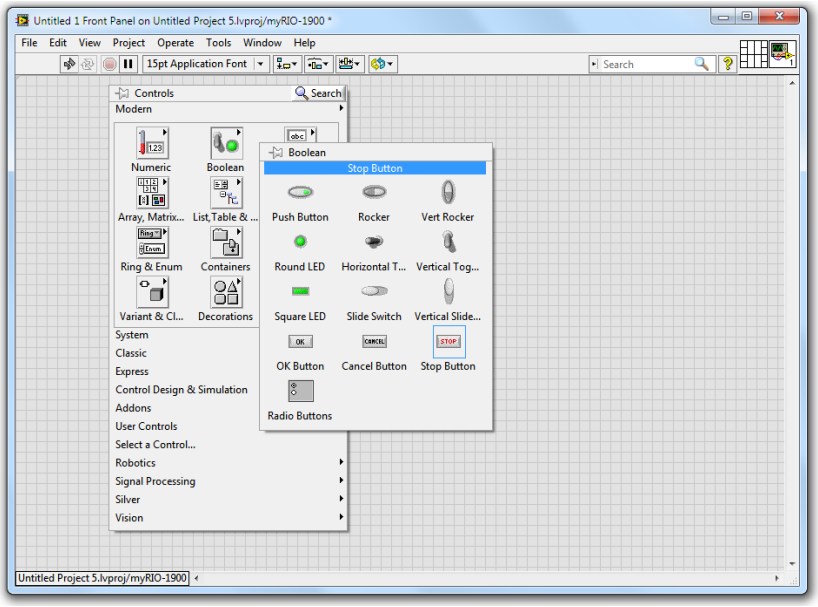
Selecting a While Loop

c. Draw a loop on the block diagram. Left-click and drag the cursor to create the loop and left-click again to finish drawing the loop. You can resize loops by holding the cursor over the border, left-clicking the loop to select it, hovering over one of the eight blue resizing squares, and left-clicking and dragging to resize.



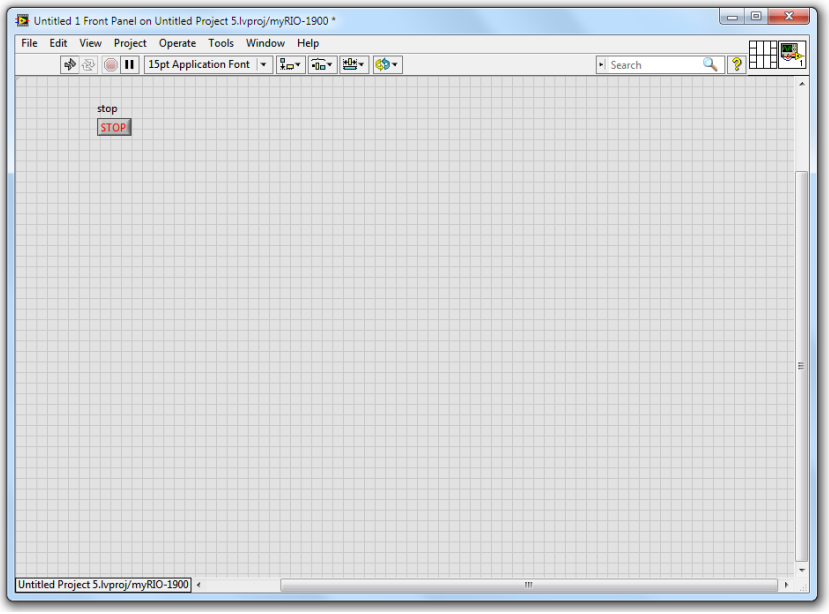
Block Diagram with While Loop

d. Switch to the front panel and right-click to open the Controls palette. This palette contains all of the elements to create the VI's display and controls analogous to a real instrument. Navigate to **Modern»Boolean»Stop Button**.



Selecting a Stop Button

e. Left-click the stop button icon and drag the item on the front panel. Notice the outline of the control you are about to place floating over the front panel. Use this visual cue to place the button anywhere on the front panel by left-clicking.

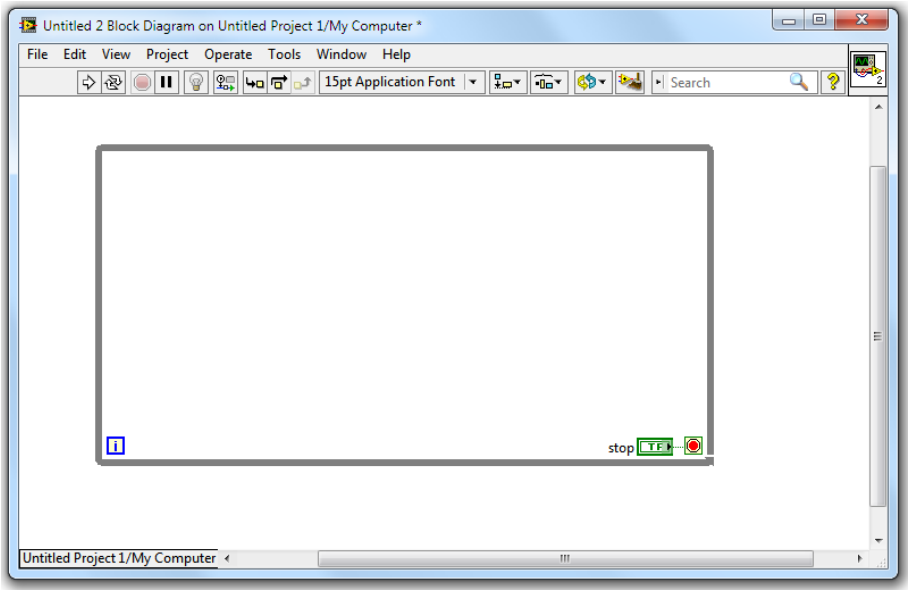


Front Panel with Stop Button

f. Now switch back to the block diagram and locate the new Stop button icon. If the Stop button icon appears outside the While Loop, left-click and drag it inside. Holding the mouse

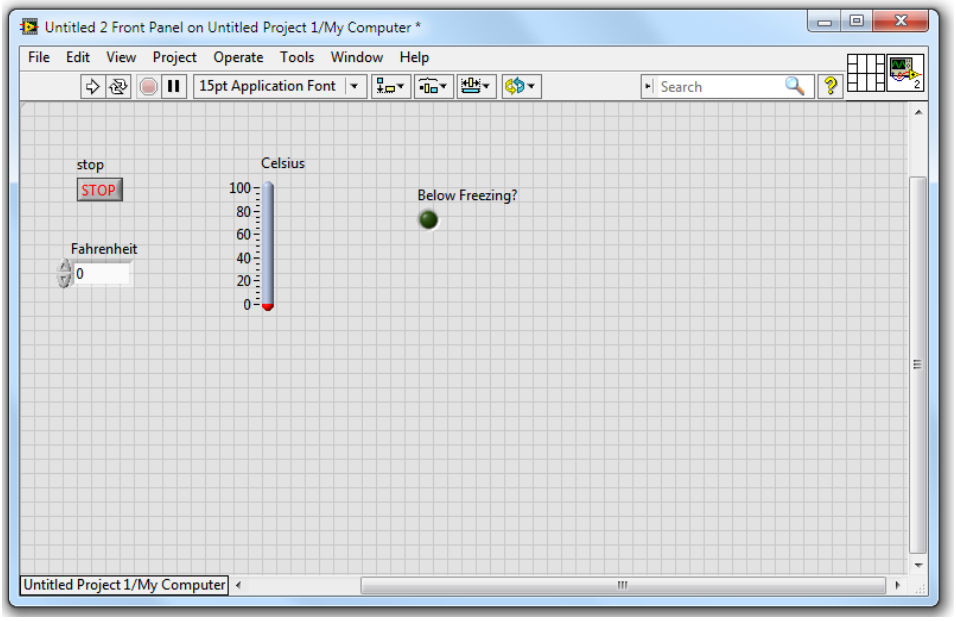
over the stop button icon makes the output terminal appear (a little green circle on the right side of the icon). Move the cursor over this circle, and the cursor changes to look like a spool of wire.

g. Wire the Stop button to the green loop stop condition icon (located in the lower right corner of the While Loop) by left-clicking on the wiring terminal on the stop button and moving the cursor over to the stop condition terminal (a shadow of the wire you are placing appears behind the cursor as you move it). Then left-click on the input terminal of the stop condition icon to connect the stop button control to it. A green wire should appear between the two icons. This indicates that the components are correctly wired Boolean values (other data types have differently colored wires).



While Loop with Stop Condition Wired

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 anywhere on the panel.
   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, selecting and deleting the text, and then typing the new name. Call this control "Fahrenheit."
   d. Open the Controls palette (right-click) and navigate to **Modern»Numeric»Thermometer.** Place the thermometer anywhere on the front panel (left-click, move cursor, left-click).
   e. In a similar fashion to the Fahrenheit control, rename the thermometer indicator by double-clicking the label. Name this indicator "Celsius."
   f. Now create additional functionality for detecting with a virtual LED when a temperature falls below freezing. Place an LED on the front panel by navigating to **Modern»Boolean»Round LED**.
   g. Place the LED anywhere on the front panel (left-click, move cursor, left-click).
   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.
    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 and select **Create»Constant**. Assign a value of "5" to this constant.
    j. Right-click on the bottom input terminal 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 Figure 1 below, and check that the run arrow is unbroken.
   a. Switch to the front panel (<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. 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. For the next exercises in this manual, you will use the same environment to create programs for the NI myRIO device with the same structure, controls, indicators, and functionality as the example in this exercise. However, you must first install, set up, and configure the device to be used with the development computer. The tools included with the NI myRIO device help make setup and configuration straightforward and simple. The next section covers hardware setup and walks through the Getting Started Wizard for the NI myRIO device.

Exercise 1 VI:

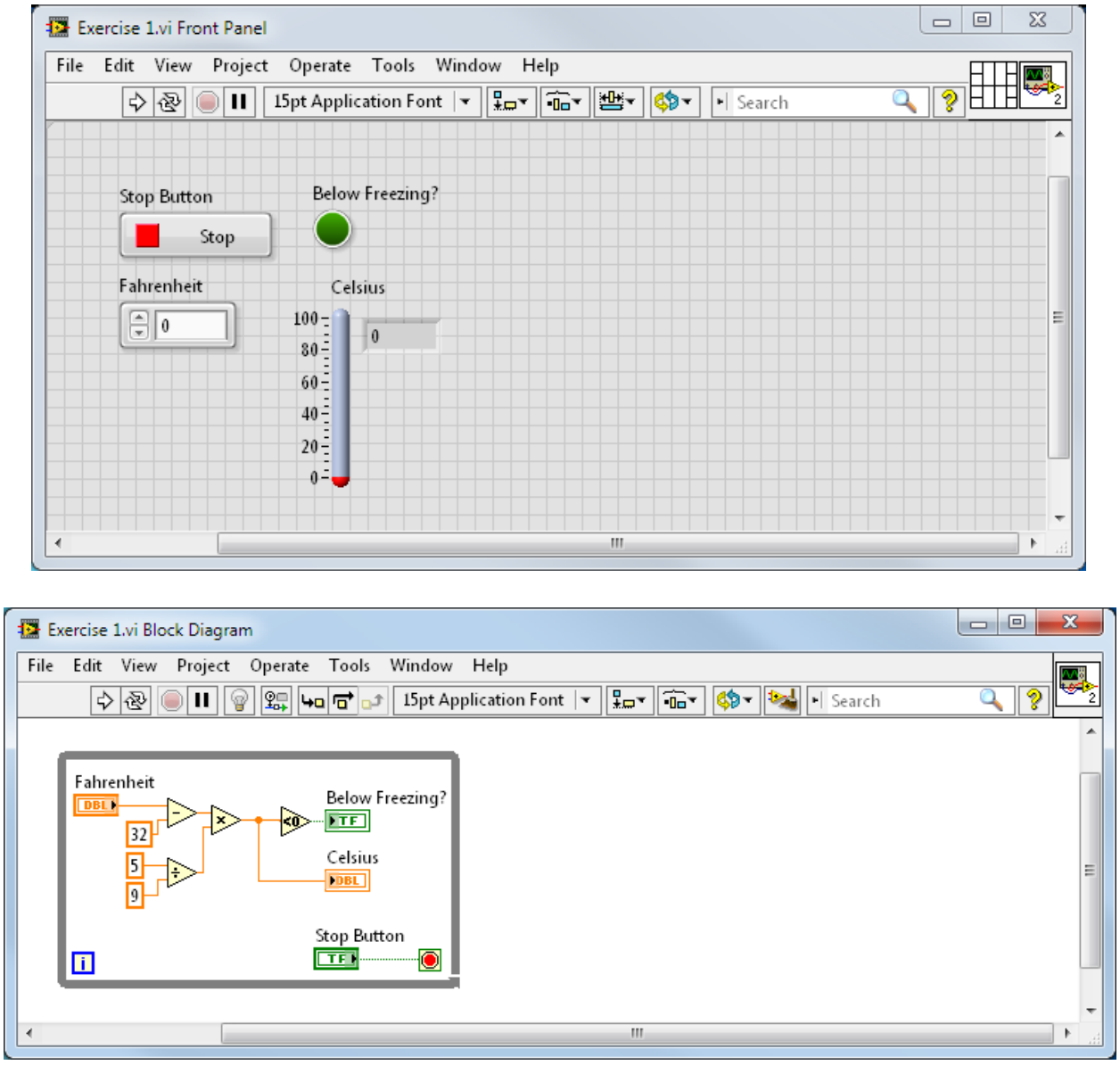


Figure 1

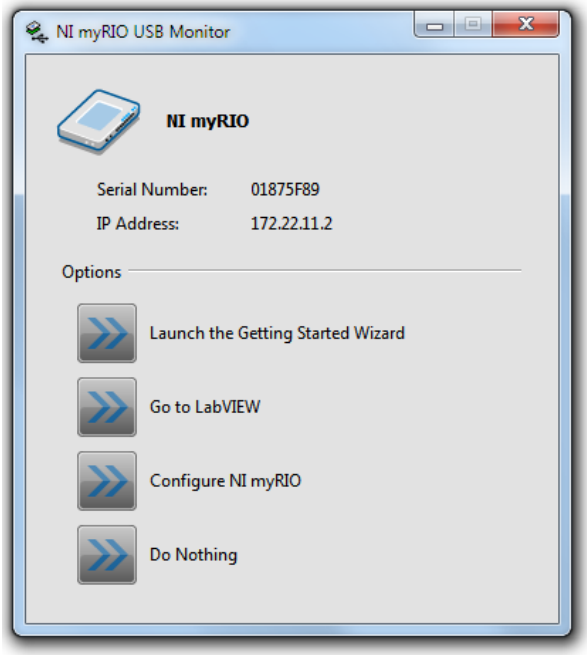
Hardware Setup: Connecting to the NI myRIO Device

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 recognize 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.

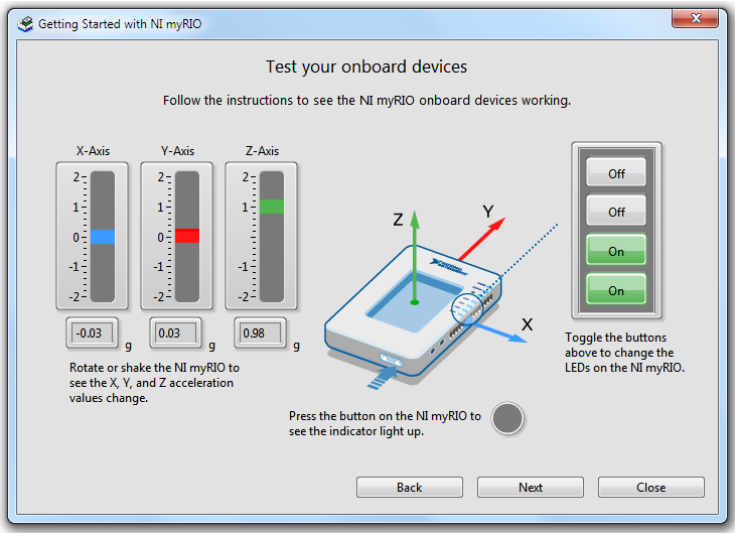
2. Go to LabVIEW
Selecting this option launches the LabVIEW Getting Started window.

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. Next, create a default project and explore the functionality of the basic architecture.

Running Real-Time Code on the NI myRIO Device

Now set up an NI myRIO project from the project templates included in LabVIEW 2013 for NI myRIO. After creating a default project, open the default "Main.vi" and explore it in detail. Then complete an exercise on the functionality of the default Main.vi.

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

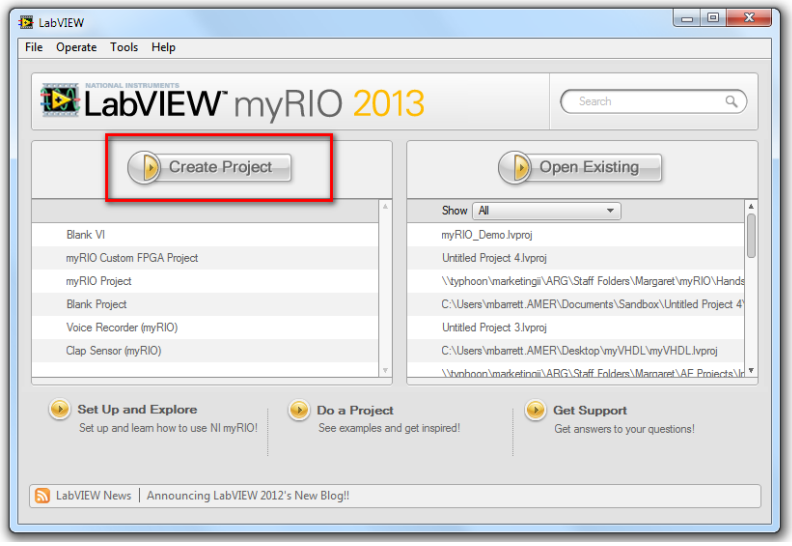
FPGA "personality." The inputs and outputs of the NI myRIO expansion connectors and the NI miniSystems port communicate with the processor through the FPGA. The FPGA is covered in more detail later; for now, remember that the NI myRIO unit is shipped with an FPGA personality that is configured by default to pass all of the input data and output data to and from the connector pins and onboard devices (buttons, accelerometer) to the processor that is running the real-time code.

Now that you have connected and configured the NI myRIO device, you can create an NI myRIO project and start developing code for the device. Using the default FPGA is a fast way to get started and create simple stand-alone code or even proof-of-concept code for more complicated projects that require an FPGA. For code that requires customizing how the NI myRIO device handles I/O, a custom FPGA project is necessary for the fastest possible code execution. Again, customizing the FPGA is unnecessary for basic applications, so this section uses a non-customized FPGA project.

Exercise 2: Investigate Main.vi

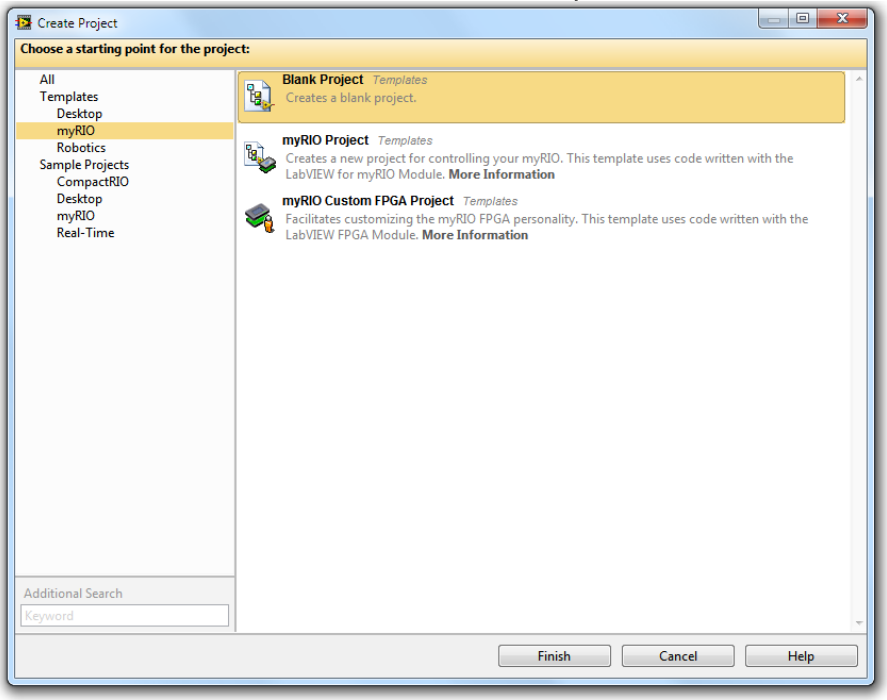
Follow these steps to set up an NI myRIO project from a template:

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.

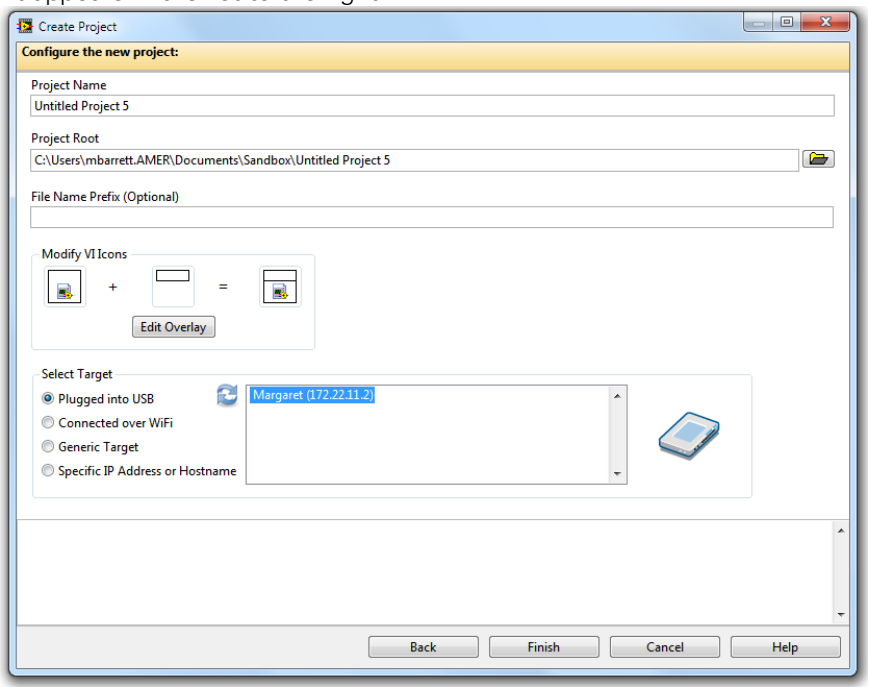


NI myRIO Project Templates

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 1.
    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 to save the project in. 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.



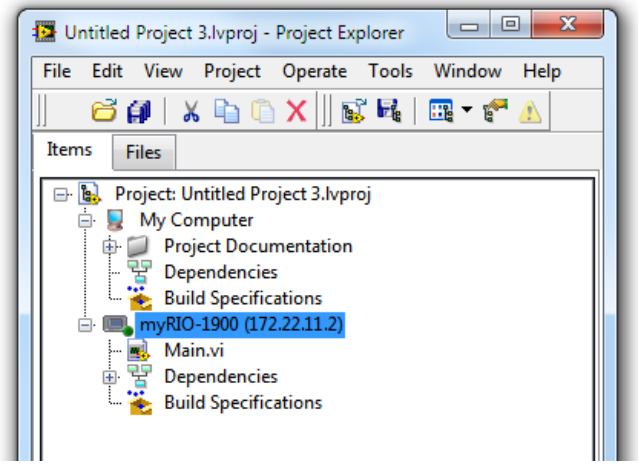
Configuring a LabVIEW Project

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. In the final form of the application, any front panel controls and indicators within VIs on the RT target are inaccessible. Code is eventually *deployed* to the NI myRIO unit and run without a USB connection to the computer. You can use network-published shared variables or some form of network streaming to a VI on the development computer to receive and process data from the NI myRIO device. 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.

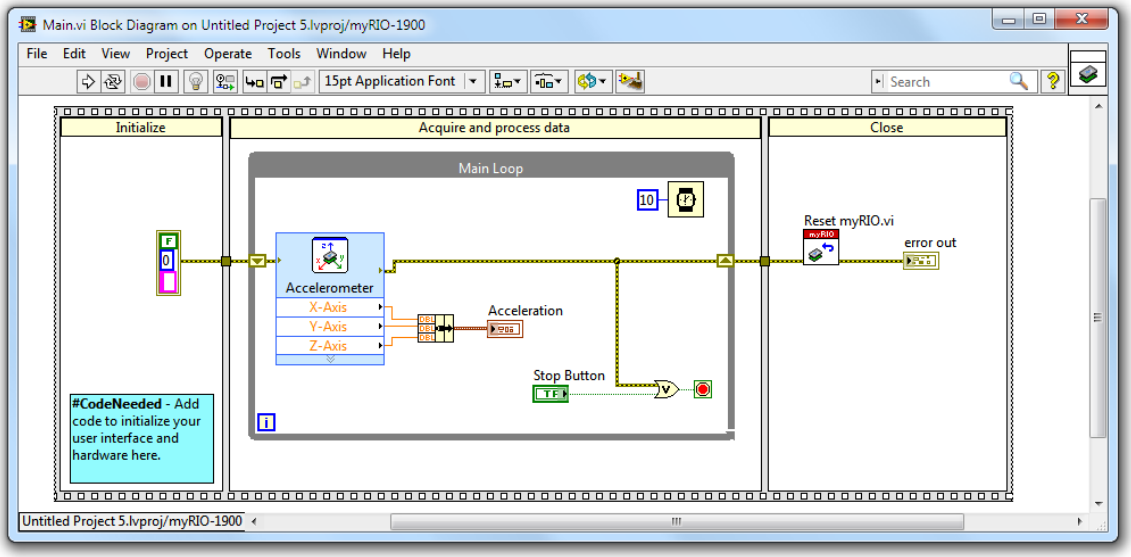
For this exercise, the NI myRIO unit is tethered to the development computer, but you still want to run the code on its processor. This means that you can interact with the front panel of the VI running on the RT processor of the NI myRIO device, and the NI shared variable engine takes care of the necessary 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. Open Main.vi from the LabVIEW Project Explorer 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.

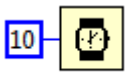


The two types of sequence structures are flat and stacked. The sequence structure in Main.vi is flat. Sequence structures force code execution into a sequence. All of the code in each frame of the structure (from left to right) must execute before the next frame can start. You can use *tunnels* to pass data across the frames. In this case, the sequence structure in Main.vi is being implemented to force an open-acquire/process-close architecture. Using a sequence structure in this manner is the simplest way to guarantee code execution in a particular fashion, though you can 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 getting quickly passed to the next frame via a tunnel.
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 being used to retrieve data from the onboard accelerometer on the NI myRIO device. Then that data is passed from the x-axis, y-axis, and z-axis output terminals through the orange wires (double precision type) and bundled into a cluster to be 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 as long as all of the code in the loop already executes in less than 10 ms; otherwise, the loop executes only as fast as the code inside it, but it is not forced to wait an additional 10 ms. This basically gives the loop a frequency of 100 Hz (again, with the caveat that the code in the loop can execute that fast). Since the waveform chart shows the last 100 samples of accelerometer data, you view an entire second of accelerometer data on the chart at a time. As with most software-timed structures, you always have some level of jitter to the timing of a loop. For more importantly timed tasks, you can improve determinism in LabVIEW through *Timed Loops* and FPGA programming.

   You can choose from two stop conditions for the main loop: the stop button, which you can press at anytime while the code is running to stop the While Loop and continue the execution of the sequence structure, and a stop-on-error condition, which you can create by wiring the error cluster out of the Accelerometer Express VI to an OR node with the stop button. The stop-on-error condition ends the While Loop and continues the execution of the sequence structure if any error occurs in a VI that the error cluster is wired to. You can combine errors from multiple VIs to create safer execution protocols so that any physical devices attached to the system are not damaged by flaws in code execution or missing sensor information.
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 you add more 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 or press <Ctrl-R> to start this VI running on the NI myRIO device.
6. When you have successfully deployed the VI to the NI myRIO unit, the waveform chart starts displaying the samples from the accelerometer. Shake the NI myRIO device to see the accelerometer readings to change in real time.
7. Press the stop button to allow the VI to exit the While Loop and finish executing the sequence structure.
8. Keep this project and VI open.

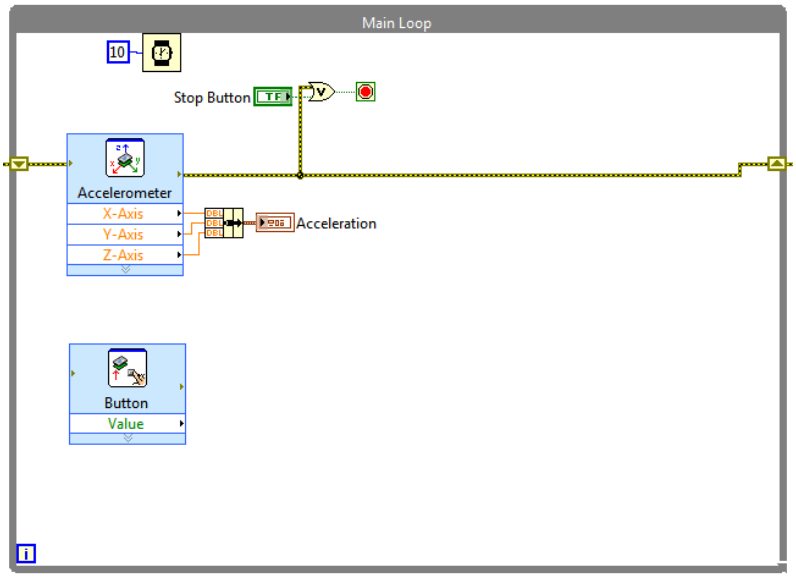
Setting up a new NI myRIO project is similar to setting up any NI real-time target. You can change any project into a real-time project by adding the RT device as a target in the project from the project root. You also can create an RT project from a template using the process from the last section. With the explanation of the default "Main.vi," its architecture and functionality should now be easy to understand. You now can add functionality to an existing process (an existing While Loop) or even add more While Loops to process data in parallel. However, the RT processor can execute processes in parallel only for as many cores as it has. When more loops are present, the processor handles parallelism the same as other programming languages—with threads and swapping between tasks at run time.

Exercise 3: Create Real-Time Code to Run on the NI myRIO Device

Using the NI myRIO project template, write a LabVIEW VI to run on the NI myRIO device to explore some of the built-in functionality of the device. In this exercise, you will create a stand-alone application using the push button and onboard LEDs.

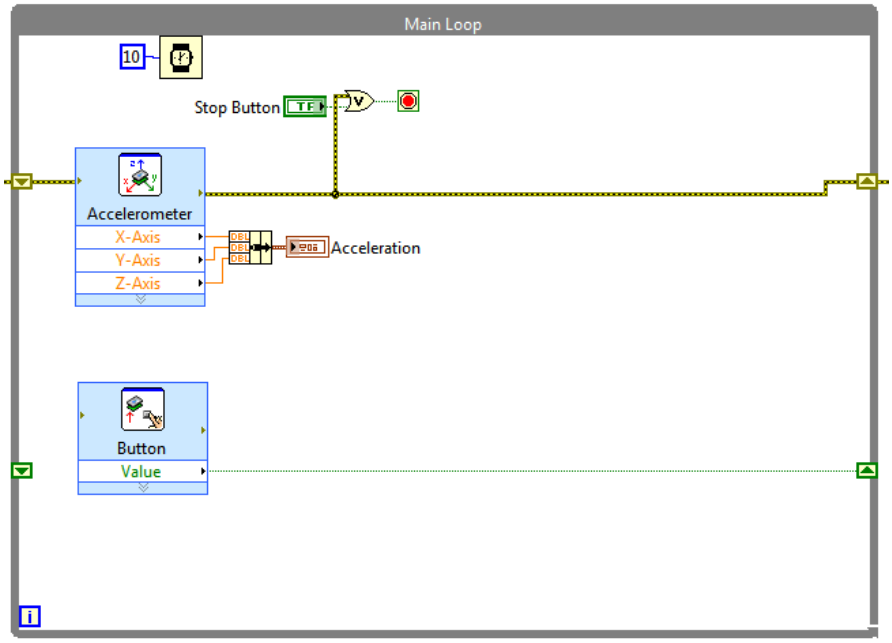
If the project from the last exercise is still open, follow the steps below to create the RT VI. If the project from the last section is not open, go back to the last section and follow the instructions to create a new NI myRIO project.

1. Open **Main.vi** under the myRIO-1900 (xxx.xx.xx.x) target device in the LabVIEW Project Explorer.
2. Open the block diagram.
3. Enlarge the center frame of the sequence structure and the While Loop within.
    a. Move the mouse cursor to the bottom edge of the sequence structure and locate the blue resizing node that appears in the center of the bottom border of the structure. Click and drag the node down to increase the space available for placing icons.
    b. Move the cursor to the right edge of the center frame of the sequence structure and locate the blue resizing node. Click and drag this node to the right to increase the area in the center frame.
    c. Move the cursor to the bottom right corner of the While Loop. Click and drag the resizing node to enlarge the While Loop to fill the space created in steps a and b.
4. Create a **rising-edge trigger** to detect when the button on the NI myRIO unit is pressed (ignoring the output when the button is held).
    a. Place a **Button Express VI** on the block diagram from the NI myRIO palette.
        i. **Right-click** and navigate to **myRIO»Onboard»Button.** Place the Express VI inside the While Loop just below the accelerometer VI.
        ii. Press **OK** in the configuration dialog box to keep the default settings for the Express VI. This Express VI handles opening a reference to the physical hardware through the default FPGA personality on the NI myRIO device and reading the current value from the button. Since you are placing this VI in the While Loop, all the code in the VI executes during each iteration of the loop. This might seem wasteful because the code has to open a reference to the particular data channel you are reading from on the FPGA every cycle. However, the Express VIs for the NI myRIO device have been written with a "smart open" feature that avoids opening the reference on every iteration. So, after the first iteration of the code, the VI executes faster because it needs to only read data in from the button.



b. Wire the Value output of the button Express VI to the right edge of the While Loop.

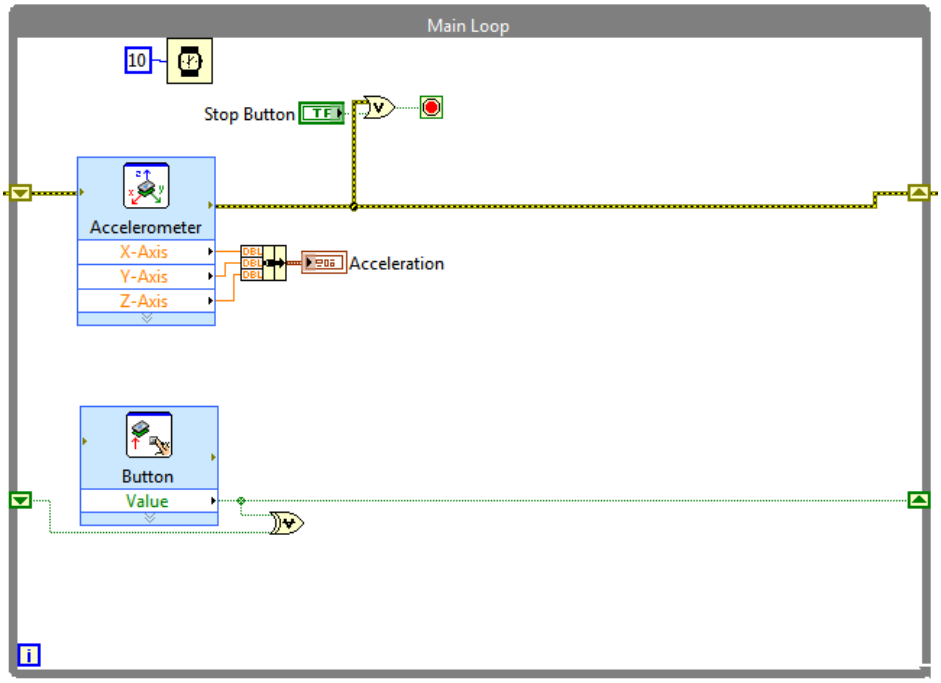
c. Right-click on the tunnel created at the right edge of the While Loop by the button value Boolean data and select **Replace with Shift Register**. This changes the icon of the tunnel and generates another icon in the same location on the left side of the While Loop. Both of these tunnels create a shift register. The right side of the register stores data from the current iteration, while the left side of the register calls the data from the previous iteration. With this register, you can compare the value of the button on the current iteration to the value of the button on the previous iteration (on the first iteration the previous value defaults to FALSE).



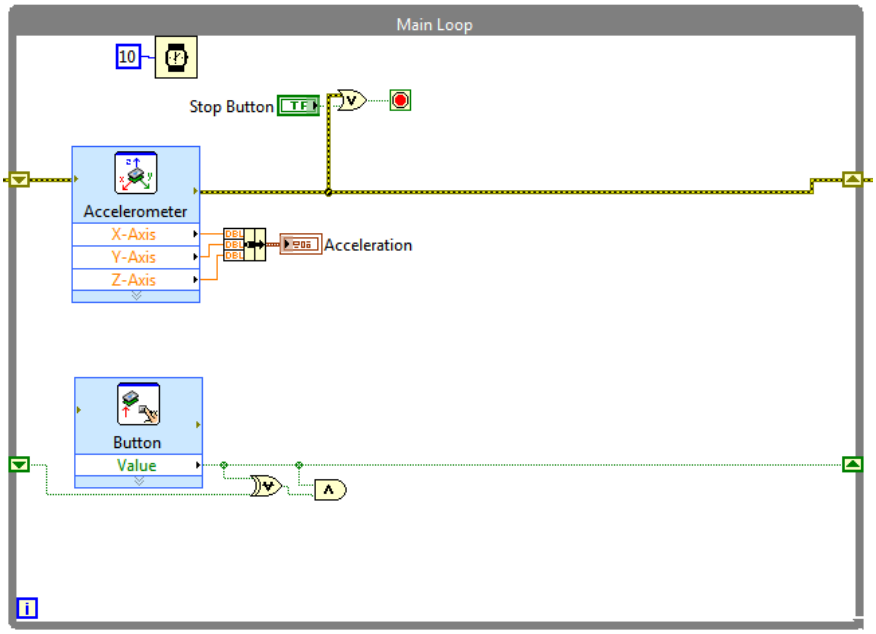
d. Place an **Exclusive Or** node on the block diagram.

   i. Right-click and navigate to **Programming»Boolean»Exclusive Or**. Place the Exclusive Or node on the block diagram just below and to the right of the button Express VI.

ii. Wire the value output of the button Express VI to the top terminal of the Exclusive Or node.
iii. Wire the shift register that stores the value of the button to the bottom input of the Exclusive Or terminal.

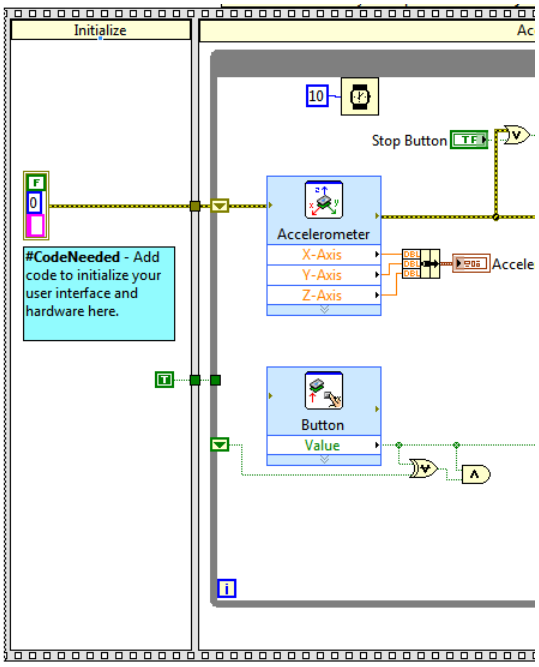


e. Place an **And** node on the block diagram.
i. Right-click and navigate to **Programming»Boolean»And**. Place the And node on block diagram to the right of the Exclusive Or node.
ii. Wire the output of the button Express VI to the top terminal of the And node.
iii. Wire the output of the Exclusive Or node to the And node bottom terminal.



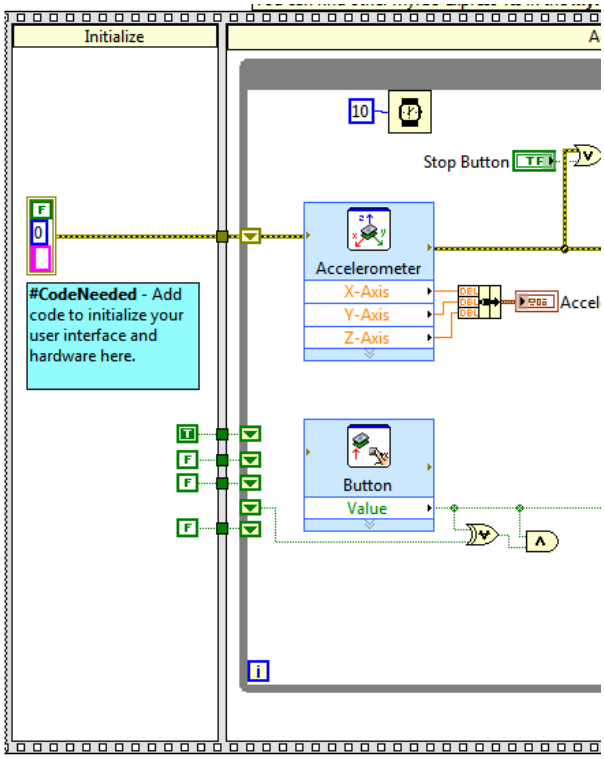
f. Use the keyboard shortcut <Ctrl-H> to open Context Help to aid in understanding the functionality of the And and Exclusive Or nodes. With Context Help open, you can hover over any front panel or block diagram item to learn more about the function of that item. Now that the logic is in place to detect the rising or positive edge of the signal from the button on board the NI myRIO unit, you can use the logic to provide some functionality.

5. Create a cycle of lighting the onboard LEDs when the button is pressed.
   a. Place a **True Constant** on the block diagram.
      i. Right-click and navigate to **Programming»Boolean»True Constant**. Place the constant in the *initialize frame* of the sequence structure.
      ii. Wire the true constant to the left edge of the While Loop (it automatically tunnels through the frame of the sequence structure).



   b. Place three **False Constants** on the block diagram.
      i. Right-click and navigate to **Programming»Boolean»False Constant**. Place three constants in the *initialize frame* of the sequence structure—the first below the true constant and the subsequent two below one another.
      ii. Wire the false constants to the left edge of the While Loop.

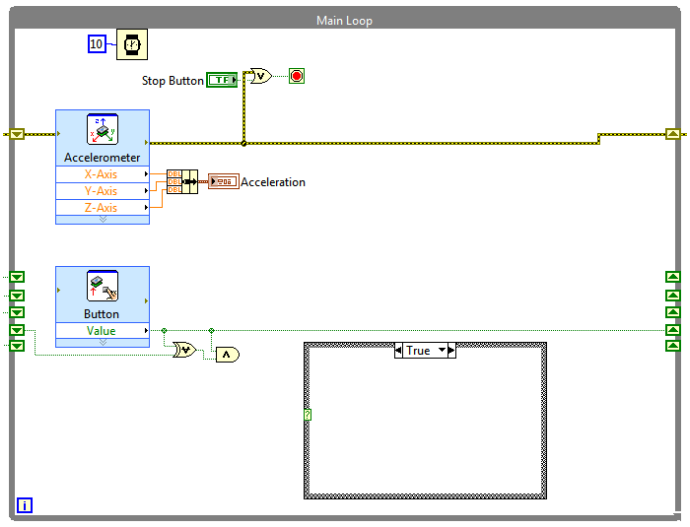
c. Replace the constant tunnels with shift registers.
   i. Right-click on each Boolean data tunnel just created on the left side of the While Loop and select **Replace with Shift Register**.



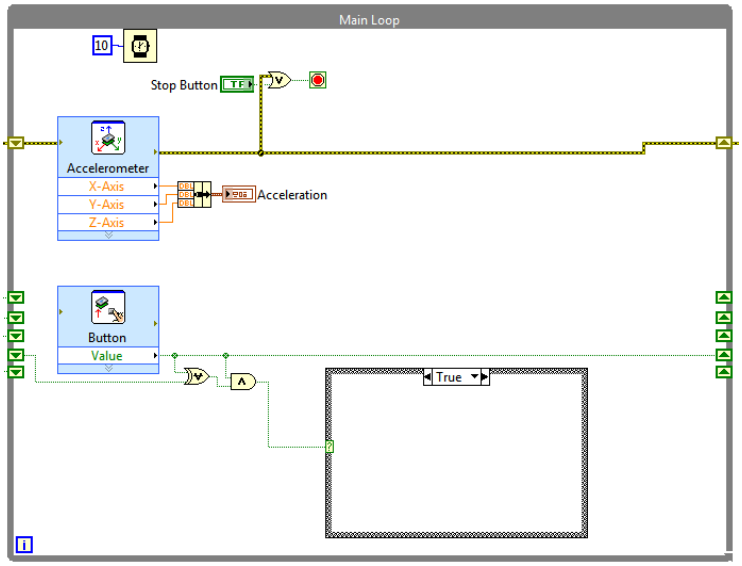
d. Each of the four Boolean constants (one true, three false) just placed on the block diagram is used to track the state of one of the four LEDs on board the NI myRIO device. This code is a very simple and straightforward implementation of the concept of shifting the value of the LED to the next LED in the set. Another method could use a counter and modulus function to take the remainder of the count and compare it to an assigned LED value (0, 1, 2, or 3) and light up the corresponding LED. This approach could use one shift register but requires placing more function blocks down on the block diagram.

e. Place a **case structure** on the block diagram.

i. Right-click and navigate to **Programming»Structures»Case Structure**. Place a case structure in the While Loop just below and to the right of the rising-edge trigger logic (Exclusive Or and And nodes).

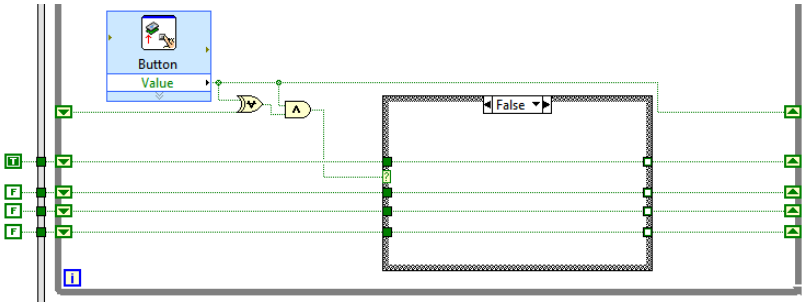


ii. Wire the output of the And node to the case selector on the case structure.



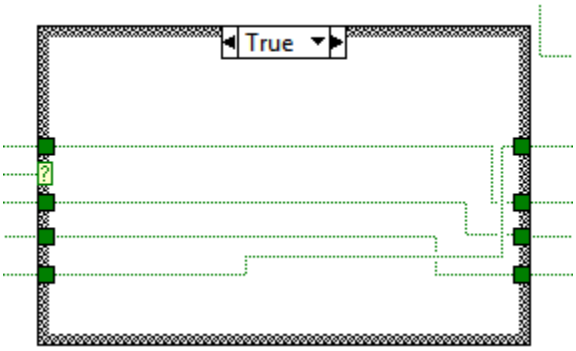
iii. Switch the case structure to the **false** case by pressing the small arrows to the left or right of the case indicator box or by selecting the drop-down arrow just to the right of the case name and choosing **false** from the drop-down menu.

iv. Wire the shift registers straight through the case structure by wiring from the left registers to the left edge of the case structure, and then wiring straight to the right side of the register on the While Loop. If the value isn't first wired to the left edge of the case structure, the automatic wiring tool passes the wire around the case structure rather than going through it.



v. Switch the case structure to the **true** case.

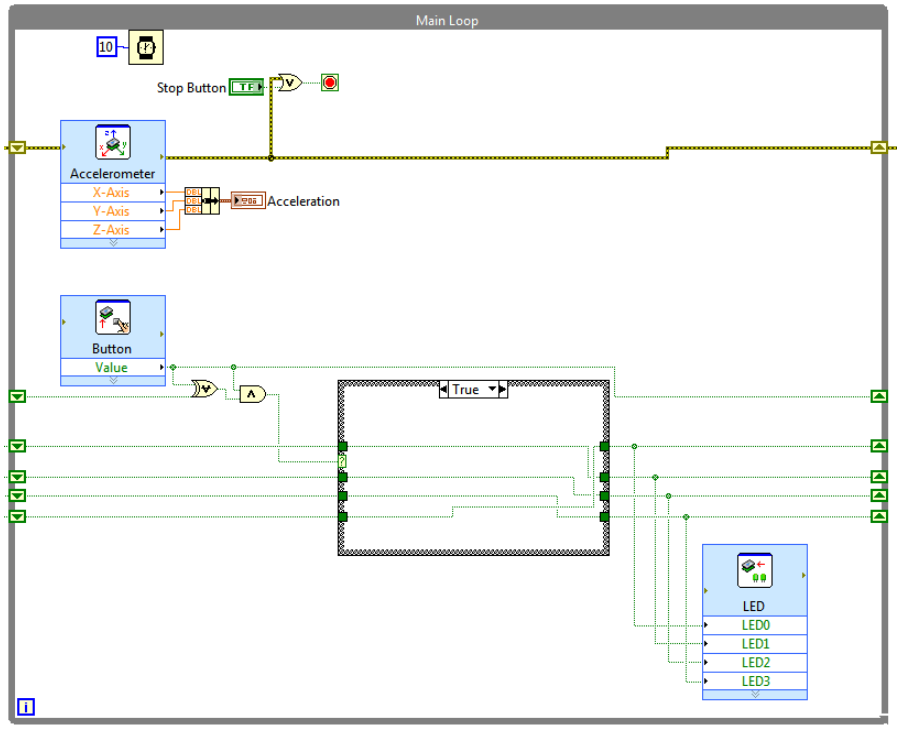
vi. **Wire** the tunnel for each Boolean value to the next tunnel below it by wiring the last tunnel on the left to the first tunnel on the right.



f. The logic here is that when the rising edge of the signal from the button is detected (in other words, the moment someone presses the button), the **true** case of the case structure shifts the value of each LED to the next LED. Then the LED that is on shifts to the next LED and the others are off. Each time the button is pressed this happens, and when the fourth LED is lit and the button is pressed, the first LED lights up and the fourth turns off. However, none of this code works if you don't tell the NI myRIO device to interpret the Boolean data as a signal for the LEDs.

6. Signal the onboard LEDs to turn on and off based on the logic that you have created.
   a. Place an onboard **LED Express VI** on the block diagram.
      i. Right-click and navigate to **myRIO»Onboard»LED.** Place the LED Express VI in the lower right corner of the While Loop. Select **OK** on the Configure LED Express VI dialog box to keep the default settings. By default, the Express VI allows all four LEDs to be written to.

ii. Wire each of the Boolean values created in the previous step to the inputs LED0, LED1, LED2, and LED3 on the LED Express VI.



7. **Run** the VI by pressing the run arrow on the toolbar. Verify the functionality of the code by pressing the button on the NI myRIO device and observing the lit LED shift to the next one over, and loop back from the fourth to the first on the fourth button press.
8. To verify that the code is running on the NI myRIO device and not on the development computer, right-click on NI myRIO in the LabVIEW project and select **Disconnect**.
9. Unplug the USB cable from the NI myRIO device and verify that the code is still running.
10. When finished, reconnect the USB cable.
11. In the LabVIEW project, select **Connect**.
12. Click the stop button on the LabVIEW front panel to stop your code.

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 controls, specialized 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

1. Start from the LabVIEW Getting Started window and select **Create Project**.
2. 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**.
3. 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.
4. Select **Next** to configure the project.
    a. Give the project an appropriate name, select a location to save the project, and select the NI myRIO device plugged into USB.
5. Select **Finish** to create the project.
6. When the LabVIEW Project Explorer loads the new project, my computer and myRIO – 1900 (xxx.xx.xx.x) appear as targets.
7. Expand the NI myRIO target and notice the new "chassis" tree. This tree appears on devices containing a targetable FPGA. Logically, the FPGA target falls under the chassis tree. The FPGA's first job is to handle 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 organized 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.
8. Open "myRIO-1900 Customized FPGA.vi" to view the shipping code placed on the FPGA.
9. 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 exemplifies safely handling I/O data from the FPGA and preparing it to be passed up to the Real-Time VI. The NI myRIO device is shipped with a default FPGA personality that handles all of the inputs and outputs on both the MXP connectors and the MSP connector. The default FPGA has PWM, UART, I^2C, SPI, and quadrature encoder I/O, all of which more than suffice for most of the 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).

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/myrio/project-guide

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 and view projects and 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.