



NI LabVIEW RIO Evaluation Kit

[Tutorial]



NI LabVIEW RIO Evaluation Kit

Tutorial

Welcome to the LabVIEW RIO Evaluation Kit tutorial. This document contains step-by-step instructions for experiencing embedded design using the NI LabVIEW system design software with the standard NI reconfigurable I/O (RIO) hardware architecture. Specifically in this evaluation the LabVIEW Real-Time and LabVIEW FPGA development modules and the NI Single-Board RIO hardware platform will be used to create an embedded control and monitoring system. The entire LabVIEW RIO Architecture, beyond the tools covered in this evaluation kit, includes a wide range of hardware and software products with a similar embedded system programming experience; visit ni.com/embeddedsystems for more information.

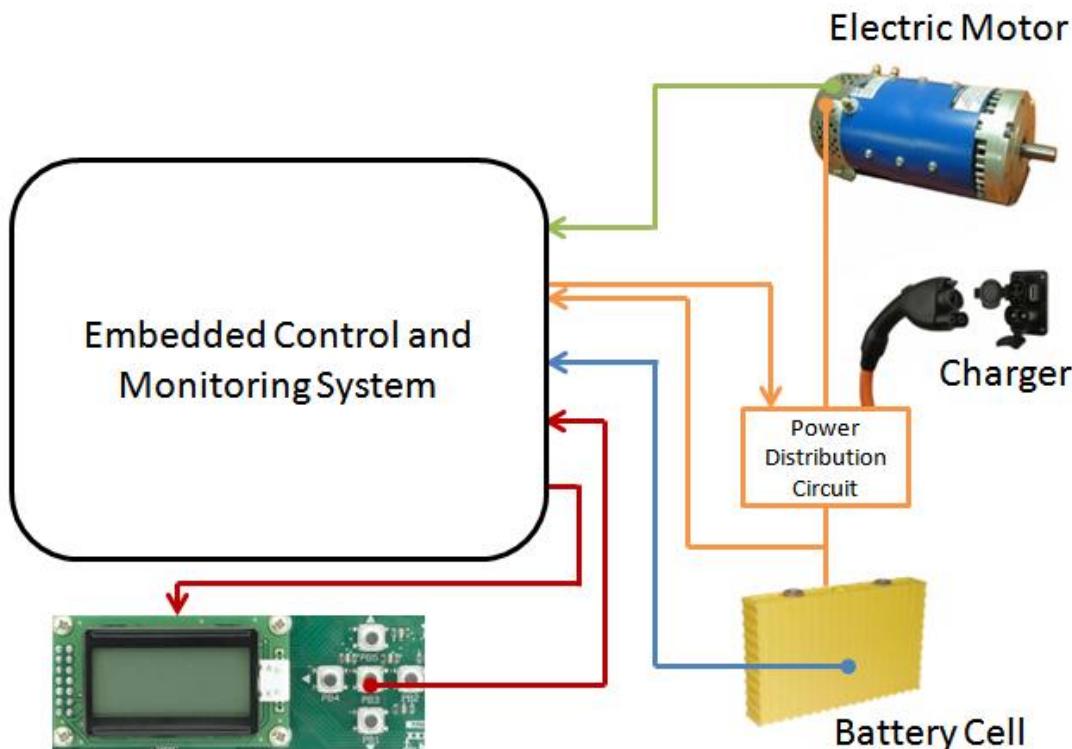
Note: *It may be easier to complete the step-by-step exercises if you print this document before proceeding. Otherwise, the Table of Contents contains links to the exercises for more convenient navigation of this document.*

Table of Contents

Tutorial Overview	4
Navigating Exercises	9
Using the Solutions	9
Troubleshooting	10
Activating LabVIEW	10
Getting Started – LabVIEW Programming Basics	11
Initial System Configuration	12
Exercise 1 – Open and Run LCD Application	15
Explore the Application	15
Deploy and Run Your First LabVIEW RIO Application	18
Exercise 2 – Create a Monitoring and Control FPGA Application	20
Acquire Button Presses	22
Monitor the Battery Cell Temperature	29
Acquire the Motor Encoder PWM and Convert to RPMs	31
Monitor the Battery Cell Voltage and Control Power Distribution	34
Exercise 3 – Develop a Real-Time Application	38
Communicate with the FPGA	42
Forward Data onto the Windows Target	47
Implement Decision Logic for LCD Screen	50
(Optional) Test the Real-Time and FPGA Applications	57
Exercise 4 – Create a Windows User Interface	62
Explore the Windows-Based Application User Interface	66
Finish Development of the Windows-Based Application User Interface	69
Run and Verify the Completed System	72
Exercise 5 – Application Deployment and Replication	74
Create and Deploy a Startup Real-Time Executable	75
Save a System Software Image and Deploy to Formatted Hardware	78
Appendix A – LabVIEW RIO Training Path	85
Appendix B - Changing the IP Address in the LabVIEW Project	90

Tutorial Overview

In this tutorial, you will complete five exercises that demonstrate how to develop an embedded system using LabVIEW system design software and NI reconfigurable I/O (RIO) hardware which includes a real-time processor, FPGA, and I/O. Using the LabVIEW RIO Evaluation Kit, your challenge will be to prototype an embedded control and monitoring system for an electric vehicle battery management system. Here is a system diagram of the battery management system that you will build up in these exercises:



Battery Management System Specifications

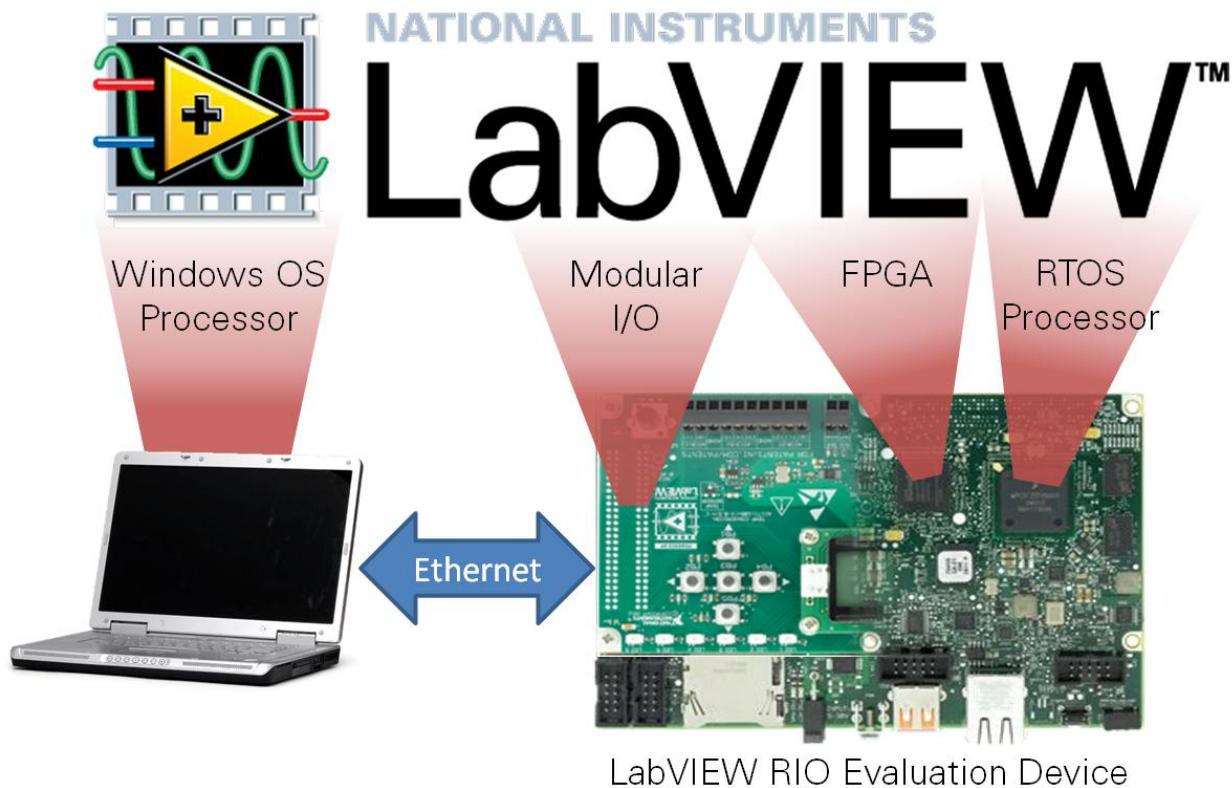
Monitoring Tasks

1. Battery cell temperature
2. Battery cell voltage
3. Motor encoder signal
4. HMI push buttons

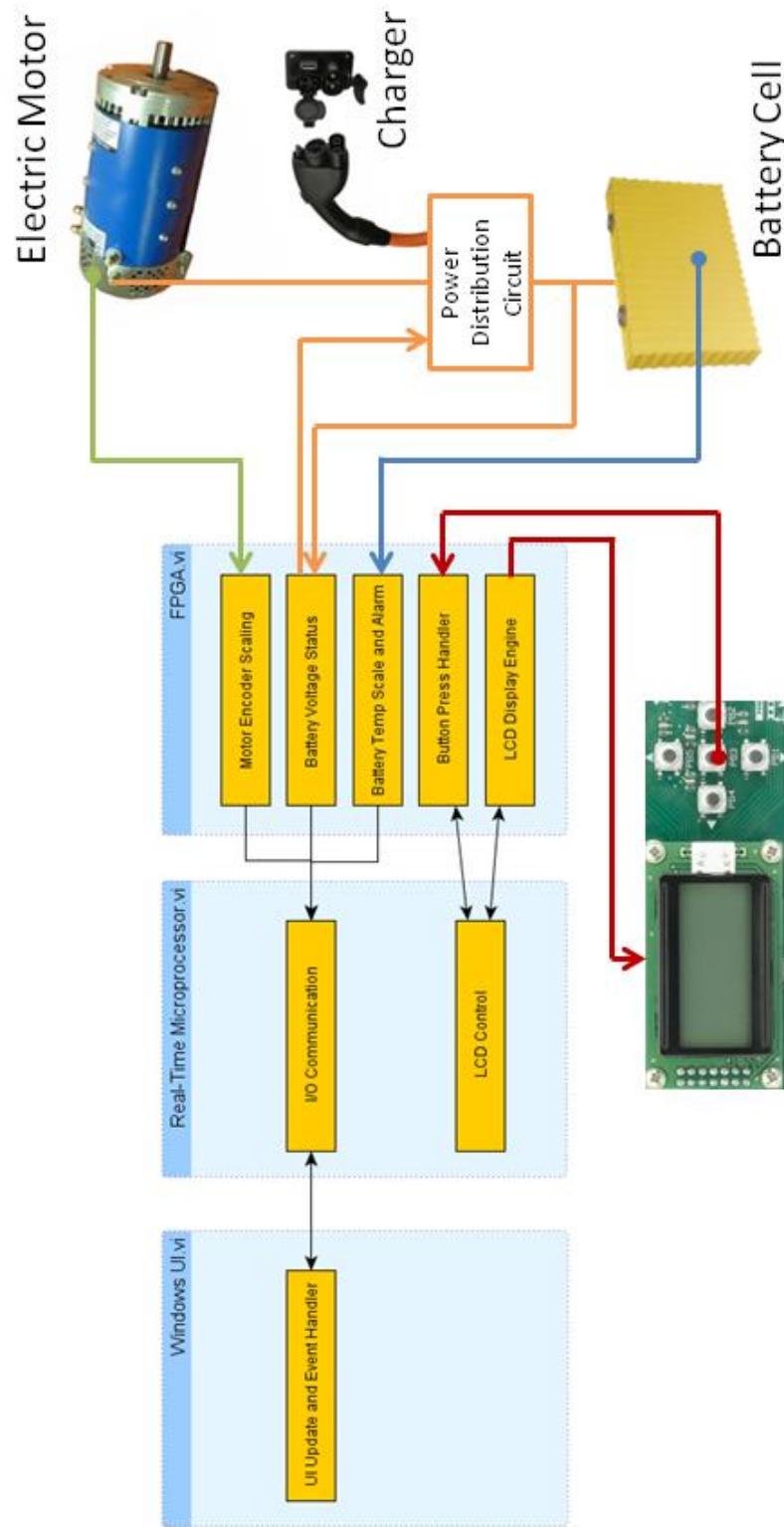
Control Tasks

1. Power distribution mode (i.e. charging, drawing, or overvoltage)
2. LCD screen

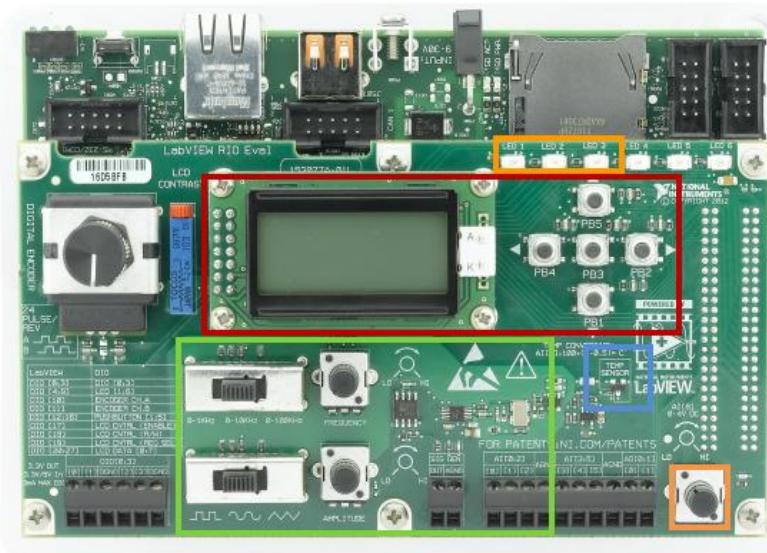
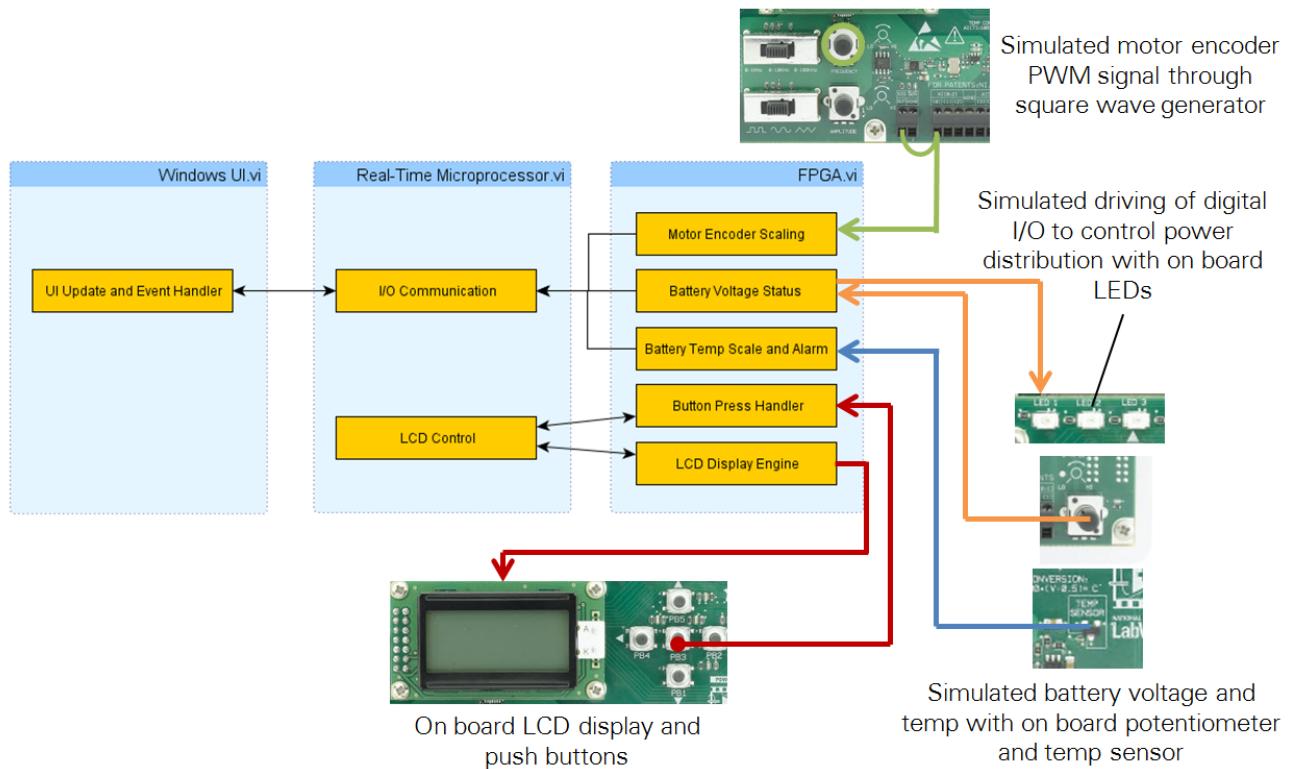
The LabVIEW RIO Architecture includes a standard hardware architecture that includes a floating point processor running a Real-Time Operating System (RTOS), an FPGA target, and I/O which can be programmed using a single development tool chain, LabVIEW. The system you are developing in this tutorial will include both a RIO board-level device and your Windows PC. Since LabVIEW includes a cross-compiler, it can be used to develop applications that will run on a floating point processor, an FPGA target, and a Windows PC.



Using the flexibility of the three different target types, the requirements of the electric vehicle battery management system outlined in the previous pages has been mapped to tasks on each of the targets of the system (Windows UI, Real-Time microprocessor, and FPGA):



Finally, since you do not have an actual electric car battery system to control and monitor here is how you will simulate it using the on-board I/O of the NI-RIO Evaluation Device:



Note: If you would like to open and modify the system diagram shown above you can download the free editor at <http://www.yworks.com/yed> and load the diagram files from .\Tutorials\Additional Resources\LabVIEW RIO Eval Kit – SysDiagram.

To build up the control and monitoring system outlined, you will design the components of the system through each of the tutorial exercises:

Exercise 1: Open and Run LCD Application

Open and run a precompiled embedded system to control the LCD screen and review the documentation of the source code to understand how the application works.

Exercise 2: Create a Monitoring and Control FPGA Application

Create an FPGA application on your own to acquire data from and control your I/O for the battery management system. While this application is compiling, complete Exercise 3.

Exercise 3: Develop Real-Time Application

Design a real-time application running on a processor which communicates with the FPGA controlling the LCD screen and coordinates network communication back to your Windows user interface.

Exercise 4: Create a Windows User Interface

Extend the embedded system to include a user interface running on a Windows computer to display the current status of your battery management system.

Exercise 5: Application Deployment and Replication

Now that your battery management system is complete, learn how to deploy and replicate the system.

After completing these exercises, explore a variety of LabVIEW Real-Time and LabVIEW FPGA applications, getting started resources, more tutorials built for the kit, and the RIO platform products on an online Community for LabVIEW RIO Evaluation Kit owners at ni.com/rioeval/nextstep.

Navigating the Exercises

The LabVIEW RIO Evaluation Kit installs several helpful exercises on your development machine that you can explore to learn more about LabVIEW Real-Time and LabVIEW FPGA software. By default, these exercises install on your computer's C: drive at

Windows 7

C:\Users\Public\Documents\National Instruments\LabVIEW RIO Evaluation Kit\Tutorials

Windows XP

C:\Documents and Settings\All Users\Documents\National Instruments\LabVIEW RIO Evaluation Kit\Tutorials

-  [1- OpenRun LCD Screen](#)
-  [2- Create FPGA Application](#)
-  [3- Create Real-Time Application](#)
-  [4- Create Windows UI](#)
-  [5- DeployReplicate System](#)
-  [Additional Resources](#)

Alternatively, you may locate these files from the Start menu at **All Programs » National Instruments » LabVIEW RIO Evaluation Kit » Tutorial Folder**. The original files are also located on the DVD included in the evaluation kit.

Using the Solutions

In addition, each exercise has a **_Solution** folder within it so if for any reason you are not able to complete an exercise successfully feel free to open the solution and/or use it to continue on to the next exercise. You will need to modify the solutions in the following ways to work with your specific evaluation hardware:

1. Update the LabVIEW project with your device's IP address. Reference **Appendix B** for more details on this process.
2. For Exercise 4 and 5 the solution requires that you update the IP address on the Windows UI.vi front panel to match the IP address of your LabVIEW RIO Evaluation device. Reference **Step 1-4 of Exercise 4 on Page 66** for more details on how to do this.

Troubleshooting

If you have any questions or run into any configuration issues while exploring this evaluation kit, please review the LabVIEW RIO Evaluation Kit Frequently Asked Questions online at ni.com/rioeval/faq. This document contains information on how to change your IP address or reconnect to your device, and includes answers to basic FPGA compilation questions.

Activating LabVIEW

The evaluation kit installs a 90-day full evaluation version of LabVIEW; you will be prompted to activate LabVIEW when you open the environment. To continue in evaluation mode, click the “Launch LabVIEW” button.

Getting Started – LabVIEW Programming Basics

If you are new to LabVIEW, this section will help you learn more about the LabVIEW development environment and graphical programming language. You can also use the web links in your Additional Resources folder to view Getting Started with LabVIEW videos and online demonstrations.

This tutorial assumes you ran the LabVIEW RIO Evaluation Setup utility upon reboot. If you have not done so, run it now. You can access the utility from your Windows Start menu, select All Programs » National Instruments » LabVIEW RIO Evaluation Kit » Setup Utility.

The hardware device included in your kit is based on the NI Single-Board RIO (reconfigurable I/O) platform and presents a hardware architecture found on other NI RIO devices which consist of two processing devices: a real-time processor that you can program with the LabVIEW Real-Time Module and an FPGA that you can program with the LabVIEW FPGA Module. These devices are connected by a PCI bus and the LabVIEW development environment includes built-in interfaces for communicating between them. The hardware device in your kit will be referred to as your “NI-RIO Evaluation HW” in this tutorial.

A LabVIEW application is called a “VI”, or virtual instrument, and is composed of two primary elements: a front panel and a block diagram, which you can program using the LabVIEW Functions Palette.

- **Front panel** – The front panel is what you use to create a LabVIEW user interface (UI). For embedded applications, such as FPGA applications, you either create subfunctions, or subVIs, where controls and indicators are used to pass data within the target application or you use the front panel to define sockets/registers that are exposed to other elements of your system (such as the real-time processor) with read/write access.

***Note:** If you close the front panel, it will also close the block diagram, so be sure to minimize it instead if you wish to use the block diagram.*

- **Block diagram** – The block diagram is where you program LabVIEW applications using a combination of graphical and textual notations. To program the block diagram, right-click anywhere on the diagram (blank white window) to bring up the Functions palette. Objects on the front panel window appear as terminals on the block diagram. Terminals are entry and exit ports that exchange information between the front panel and block diagram. Terminals are analogous to parameters and constants in text-based programming languages.
- **Functions palette** – The Functions palette contains components for creating FPGA, RT, and host interface applications. To do this, place the components on the block diagram and wire them together by left-clicking on a terminal and dragging the wire to your destination, completing this wire segment with another left-click.

Using the functions palette – In this tutorial, **bold** text denotes an item to select from the Functions palette. To access the Function palette, right-click anywhere on the LabVIEW block

diagram. You can also “pin” the functions palette (in the upper left corner of its window) so that it is always present on the block diagram.

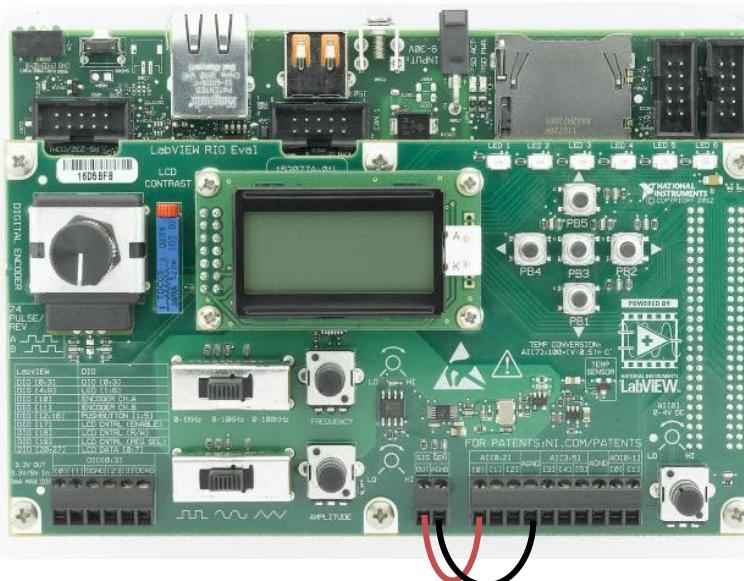
More on Using LabVIEW - To learn more about the LabVIEW graphical programming environment including syntax, deployment, debugging, and more, reference

<http://www.ni.com/gettingstarted/labviewbasics/>.

Initial System Configuration

NI-RIO Evaluation Hardware

1. If you haven’t already, complete the **Setup Wizard** located at Start»All Programs»National Instruments»LabVIEW RIO Evaluation Kit»Setup Utility
2. Use the included NI screw driver and wire to connect the Signal Generator OUT and AGND terminals to the AI0 and AGND terminals as shown below.

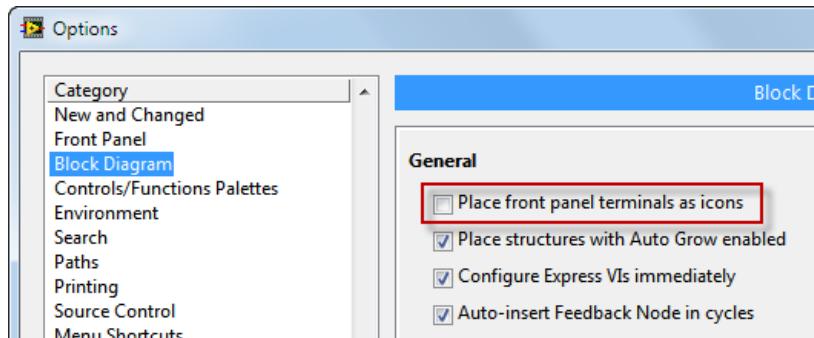


3. In preparation for application development verify that the function generator is set to **0-1KHz** and **Square Wave** generation.

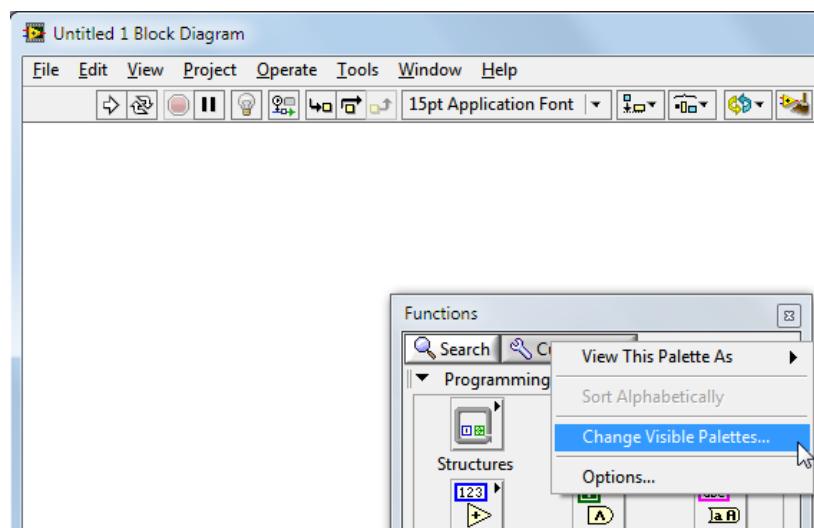


LabVIEW Environment

1. Launch LabVIEW and navigate to the **Tools»Options...** menu. Click on the Block Diagram category and uncheck the **Place front panel terminals as icons** check box. Click OK. This will conserve space on the block diagram.



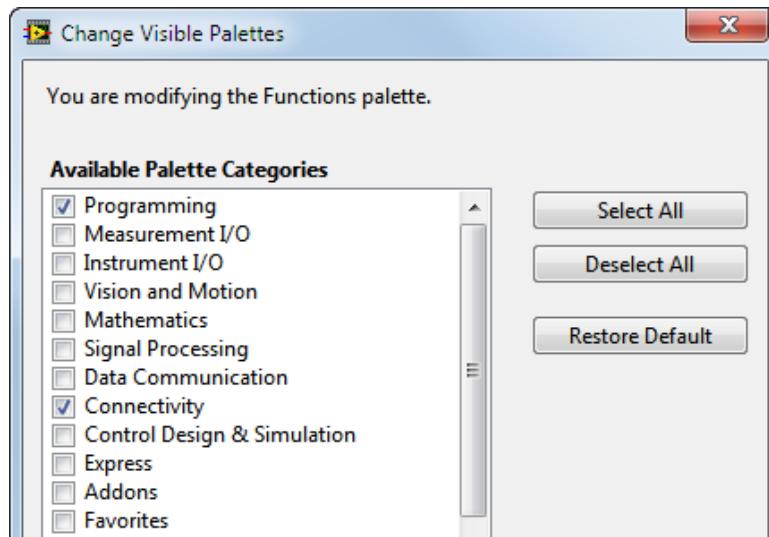
2. Create a new application (called a VI) by selecting **File»New VI**. Click on the Block Diagram window to bring it to the front and right-click it to make the Functions Palette appear. Pin down the Functions Palette (upper left corner of its window) and click on **Customize»Change Visible Palettes...**



3. Click **Deselect All** and then check the following palettes that you will use in this tutorial. Once you are done click OK.

Palettes Needed

Programming
Connectivity
Real-Time
FPGA Interface



4. Close the **Untitled 1.vi** without saving it.

Exercise 1 | Open and Run LCD Application

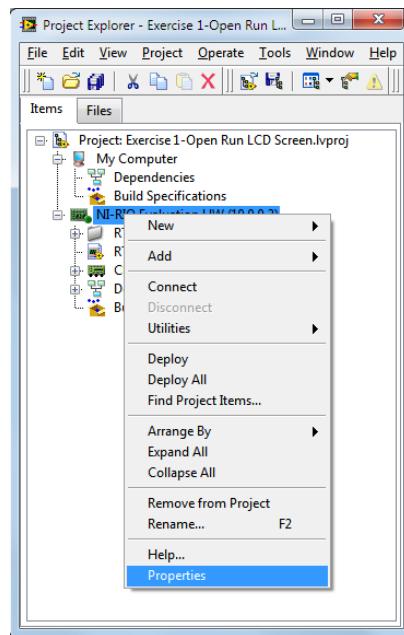
Summary

In this exercise you are going to open and run a preconfigured application that communicates between the Real-Time Microprocessor and FPGA to scroll text across the hardware's LCD screen. To deploy the application, you will complete the following tasks:

1. Explore the Application
2. Deploy and Run Your First LabVIEW RIO Application

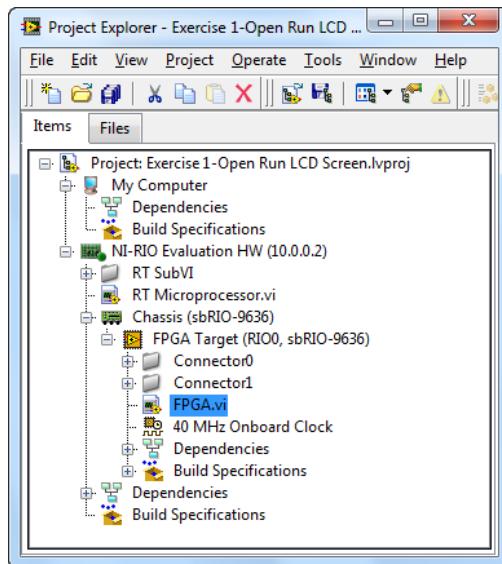
Explore the Application

1. Open up the Exercise 1 LabVIEW project file by navigating to .\1- OpenRun LCD Screen\Exercise 1-Open Run LCD Screen.lvproj.
2. Change the IP address of the *NI-RIO Evaluation HWtarget* in the Project Explorer window to match the IP address of your evaluation board.
 - a. Right-click the *NI-RIO Evaluation HWtarget* in the Project Explorer window and select **Properties** from the menu to display the General properties page.
 - b. In the **IP Address / DNS Name** box, enter the IP address you wrote down from the National Instruments LabVIEW RIO Evaluation Kit Setup utility and click OK.



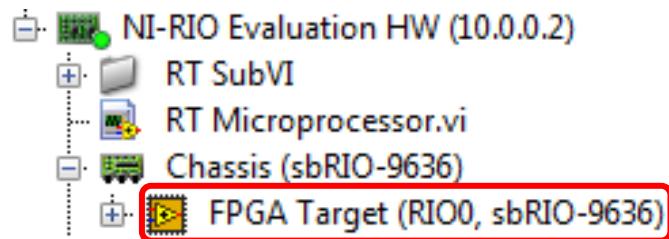
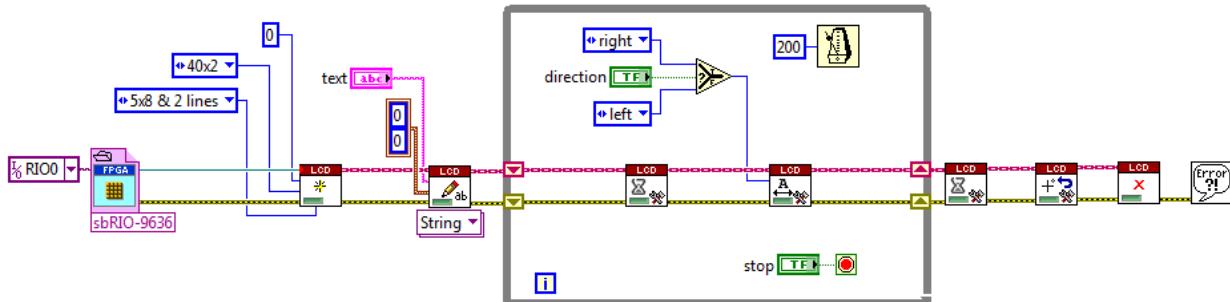
Note: If you forgot to write down the IP address from the setup utility and need to determine the IP address of your RIO device, see Appendix B - Changing the IP Address in the LabVIEW Project at the end of this tutorial for further instructions.

3. In the project, expand out the *NI-RIO Evaluation HW* target, Chassis, and then the FPGA Target. Note there are separate VIs for the Real-Time Microprocessor and the FPGA. Double-click on the **FPGA.vi**.



4. Note that the front panel of an FPGA application is simple as it is not intended to be used as a User Interface (UI) but instead the controls/indicators represent the FPGA registers that are accessible for communication between the FPGA and the real-time processor.
5. Open the block diagram by pressing **CTRL+E** or navigating to **Window»Show Block Diagram** and observe that the FPGA VI uses a low-level LCD driver API to interface directly with the digital I/O lines that drive the LCD screen. Press **CTRL+H** to open the **Context Help** window, which gives details about the code your cursor interacts with.
6. Close the FPGA VI.
7. In the project, navigate to and double-click on the **RT Microprocessor.vi** that will execute on the Real-Time Operating System (RTOS) running on the processor.
8. Note that this front panel is also very simple because the real-time operating system is headless, without graphics support. As a result, the front panel of a real-time application is useful for development and debugging but should not be used for final deployment.

9. Open the block diagram by pressing **CTRL+E**.
10. Observe on the left side of your block diagram that the real-time application opens a reference to a pre-compiled bitfile that will run on your FPGA Target with the resource name *RIO0* from your project. Then commands are sent to the FPGA to drive the LCD screen based on those commands through digital I/O lines.



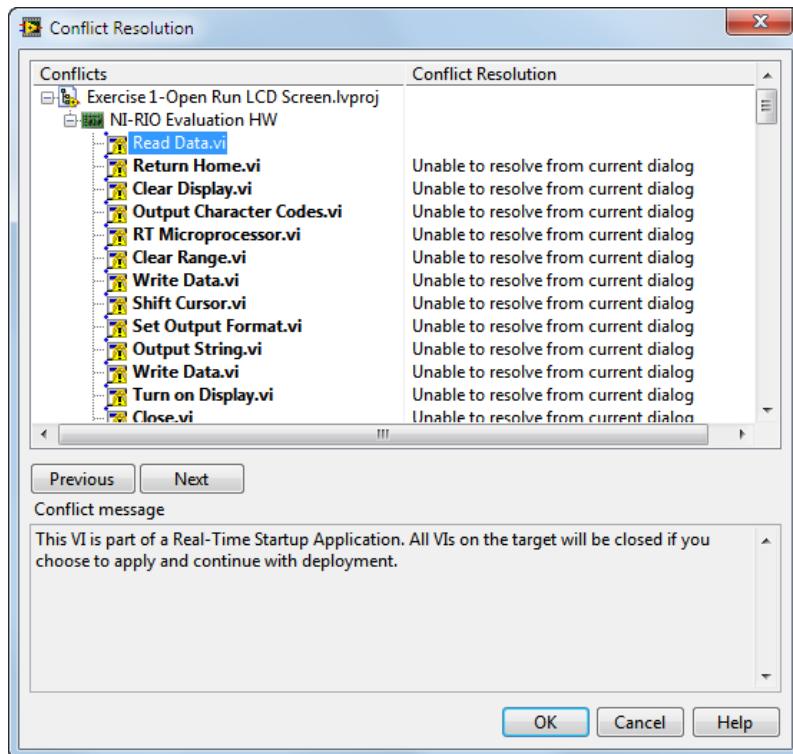
You will be writing text to the LCD screen and then scrolling it across the display moving right or left.

Deploy and Run Your First LabVIEW RIO Application

1. Deploy and run the real-time application by pressing the **Run** button  on the toolbar for the RT Microprocessor.vi. Select **Save** if it prompts you.

Once you kick off the deployment, a dialog box will appear showing the current compilation process and then will deploy the application down to the microprocessor on the NI-RIO Evaluation Device. In this case, the FPGA VI is deployed through the real-time VI so it is not required to deploy it separately.

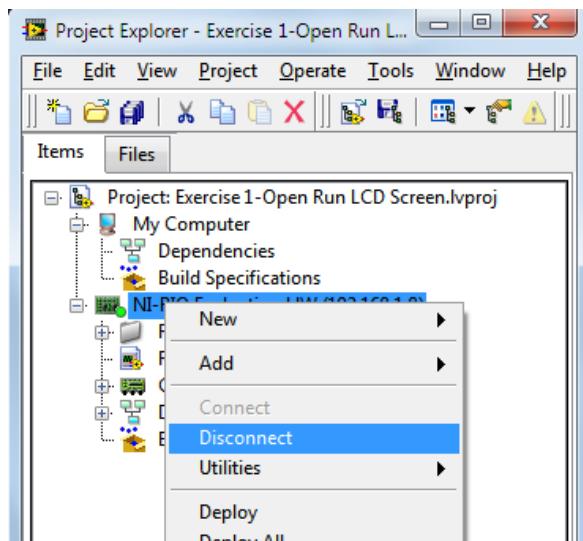
Note: When the following conflict dialog appears click **OK** as you want to overwrite the initial application that was loaded on the real-time processor during the wizard configuration.



2. Keep the real-time application running. View the LCD screen on your target and the scrolling text. Click on the **direction** horizontal toggle switch on the front panel of your VI to scroll the text in the opposite direction. Click on the **STOP** button.
3. Modify the **text** string control with a word or short phrase of your choosing to display on the LCD screen. Press the **Run** button and select **Save** if it prompts you.
4. View your text scroll on the LCD screen. When finished, click on the **STOP** button.

Note: If you are unable to see your text make sure the LCD Contrast is set appropriately (left of the LCD screen). Otherwise consult the *Troubleshooting* section on [page 10](#).

5. When you are finished exploring the project, right-click on the *NI-RIO Evaluation HW* target in the LabVIEW project and select **Disconnect**. Note the bright green Boolean turns off, indicating the target is disconnected. Finally, close out all the LabVIEW files and save the files if prompted.



Tip: If you do not disconnect your current LabVIEW project from the target, then any subsequent LabVIEW projects that you attempt to deploy files from will present an error. To release this reservation, press the reset button to reboot the device.

Congratulations, you have deployed and run your first LabVIEW built embedded application on RIO hardware!

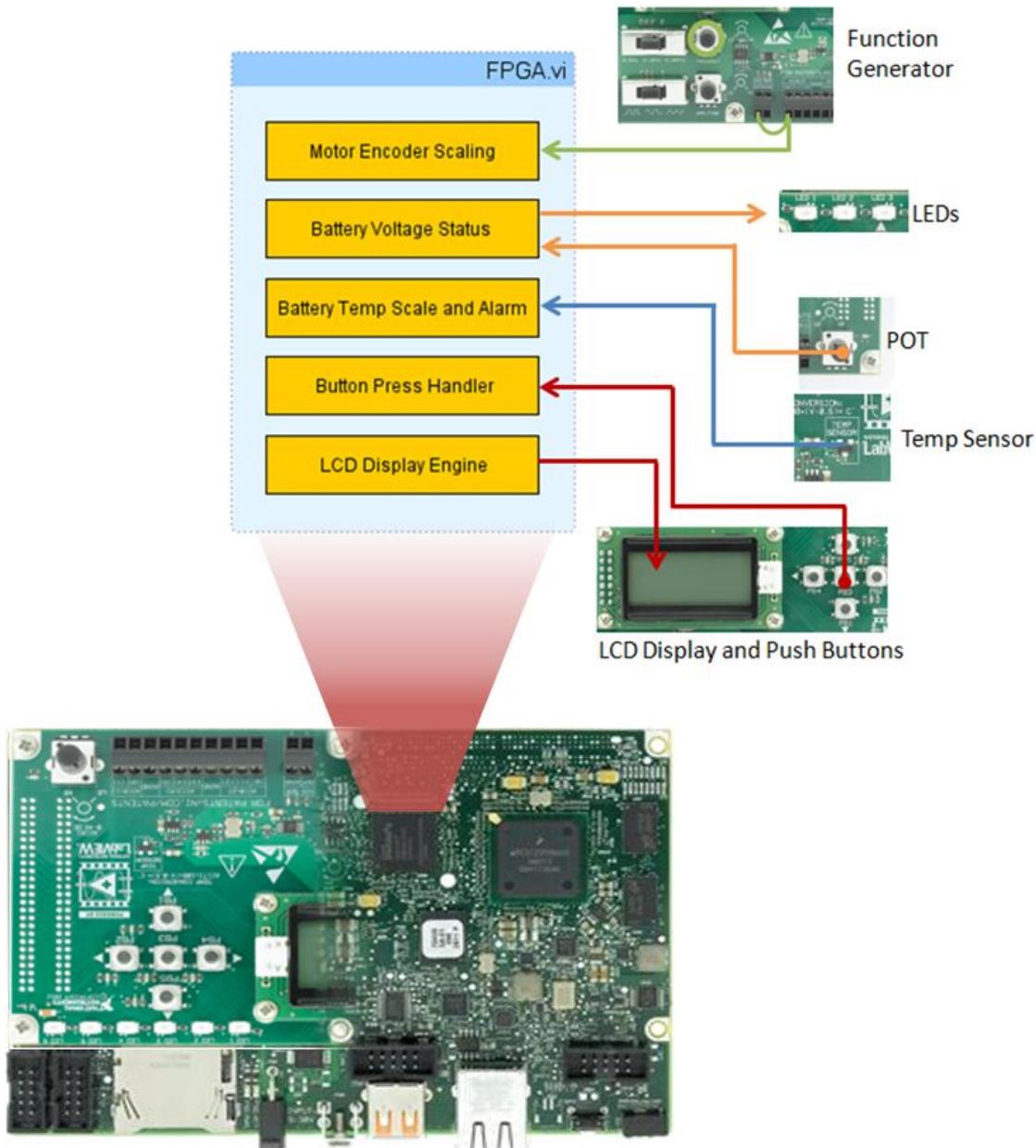
Exercise 2 | Create a Monitoring and Control FPGA Application

Summary

To start the development of your embedded system you are going to create the LabVIEW FPGA application which acquires data from and controls your I/O for the Battery Management System. In this exercise you will implement these four tasks:

1. Acquire Button Presses
2. Monitor the Battery Cell Temperature
3. Acquire the Motor Encoder PWM and Convert to RPMs
4. Monitor the Battery Cell Voltage and Control Power Distribution

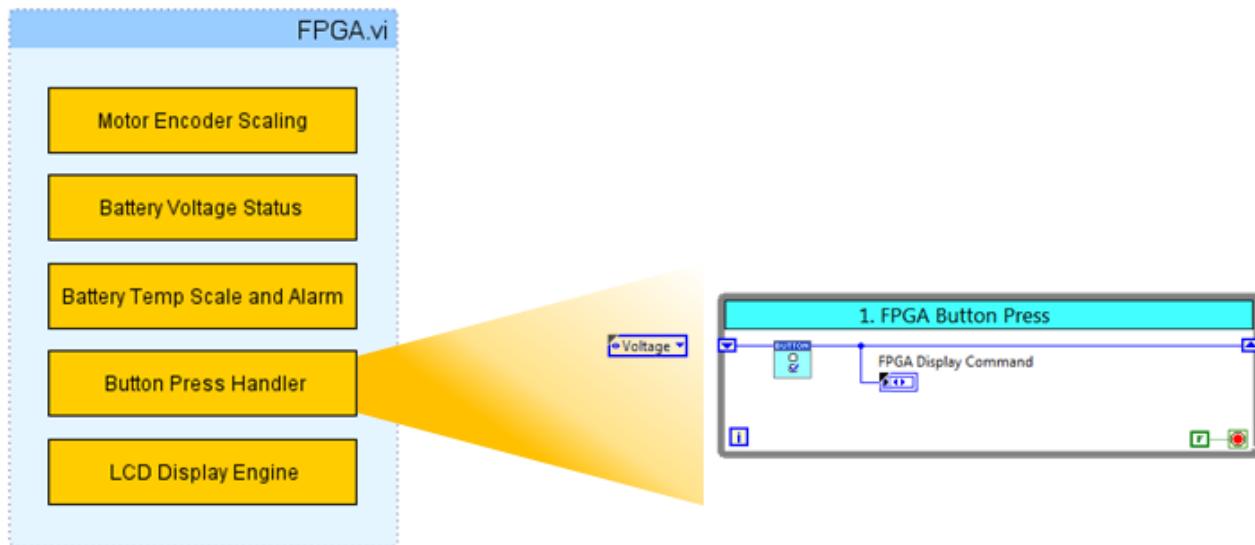
What am I going to accomplish in this exercise?



In this exercise you will develop the FPGA application which has direct access to the hardware's I/O, to acquire samples on inputs and drive control through outputs. You will implement monitoring logic for the battery cell voltage, battery cell temperature, motor encoder, and on-board push buttons. Based on the current voltage and temperature your FPGA logic will then control digital outputs to the system – in this case controlling on-board LEDs.

Acquire Button Presses

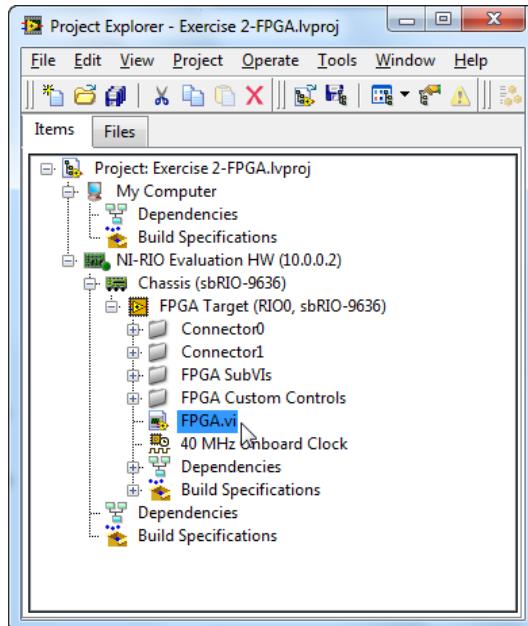
In this section you will develop logic to acquire the push button presses on the NI-RIO Evaluation Device and communicate them to other parts of the FPGA application and to the real-time application.



1. In LabVIEW select **File»Open Project...** and open up the Exercise 2-FPGA project file in the Exercise 2 folder at **.\2- Create FPGA Application**.
2. Change the IP address of the *NI-RIO Evaluation HWtarget* in the Project Explorer window to match the IP Address of your evaluation board.
 - a. Right-click the *NI-RIO Evaluation HWtarget* in the Project Explorer window and select Properties from the menu to display the General properties page.
 - b. In the **IP Address / DNS Name** box, enter the IP address you wrote down from the National Instruments LabVIEW RIO Evaluation Kit Setup utility and click OK.
 - c. Right-click on the *NI-RIO Evaluation HWtarget* in the Project Explorer window and select **Connect** to verify connection to the device.
 - d. Click **OK** if a dialog appears requesting to overwrite the files that are still running on the device from Exercise 1.

Note: If you forgot to write down the IP address from the setup utility and need to determine the IP address of your RIO device, see Appendix B - Changing the IP Address in the LabVIEW Project at the end of this tutorial for further instructions.

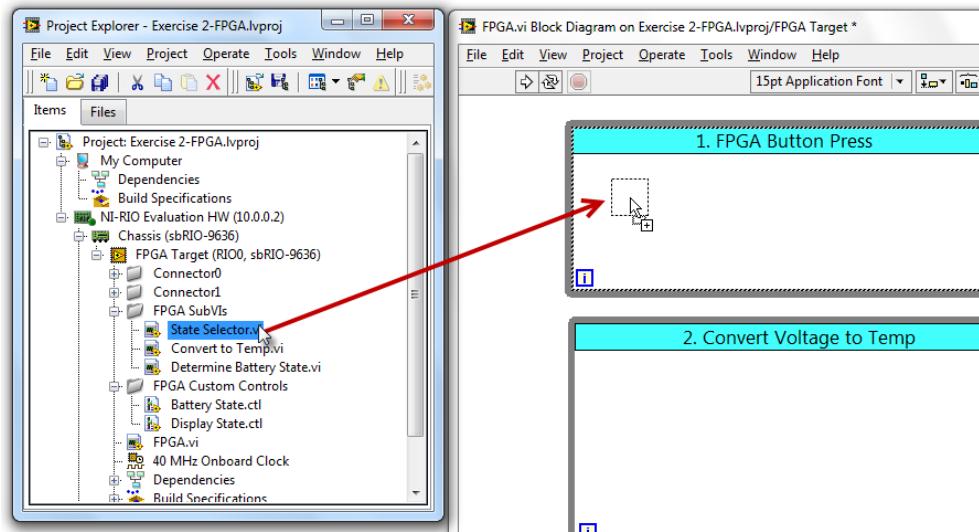
3. Expand out the *NI-RIO Evaluation HW* target and Chassis in the Project Explorer to expose the FPGA target.
4. Double-click on **FPGA.vi** to open up the existing LabVIEW FPGA application.



5. Select the block diagram by pressing **CTRL+E** or navigating to **Window»Show Block Diagram**.
6. Observe that the **LCD Display from FPGA/RT** while loop has already been inserted from the Functions palette (Connectivity»LCD»Hitachi HD44780»Command Handling Loop) as well as a while loop for each of the tasks that will be programmed.

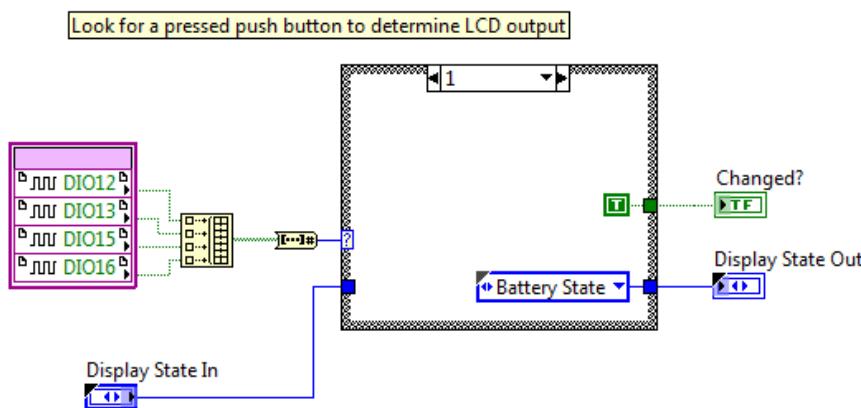
Note: Each of the while loops on a LabVIEW FPGA VI executes in true parallelism since each task is mapped to dedicated logic on the FPGA. Also because the loops are implemented in dedicated hardware logic the stop conditions are wired to false constants.

7. To re-use Intellectual Property (IP) already built for LabVIEW FPGA button selection navigate back to the Exercise 2-FPGA project window, expand out the FPGA SubVIs folder, and drag the **State Selector** VI into the **FPGA** VI.



8. Once the subVI is inserted into the **FPGA Button Press** while loop, double-click on it to bring up its front panel and press **CTRL+E** to view the block diagram code.

Note: As shown in the figure below, the block diagram of the **State Selector** VI reads in the digital input lines from the four outer onboard push-buttons and uses a case structure to select the appropriate display state. DIO12 corresponds to Push Button 1 (PB1 on the board), DIO13 corresponds to PB2, and so on.

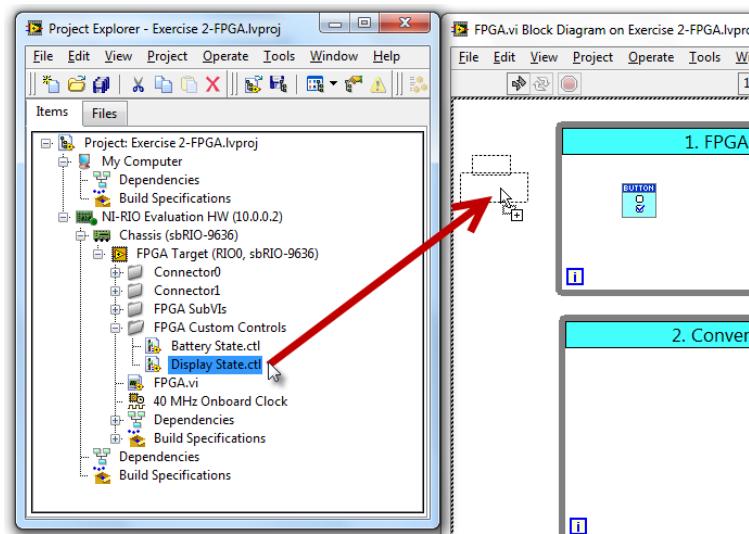


9. Close out the **State Selector** front panel and block diagram to return back to the **FPGA.vi**

Note: Save the VI if prompted when closing it.

10. To control and keep track of the state of the LCD Screen, an enumerated type control (enum) constant has been created. An enumerated type control lets you create a numbered list of named items from which to select.

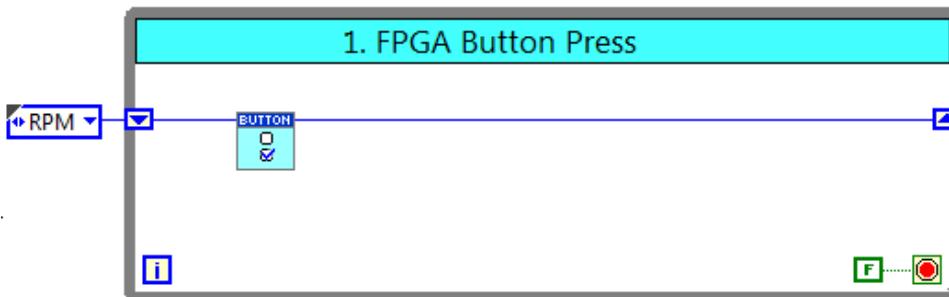
To insert the enum, back in the Project Explorer, expand out the **FPGA Custom Controls** folder and drag the **Display State.ctl** to the left of the FPGA Button Press while loop as shown.



11. Wire the enum constant through the edge of the while loop and into the **Display State In** input of the **State Selector** subVI. Wire the **Display State Out** output of the **State Selector** subVI to the right side of the while loop, creating a tunnel on the loop border.

Tip: Reference the *Getting Started – LabVIEW Programming Basics* section on [page 11](#) for more information about how to wire and other LabVIEW environment fundamentals.

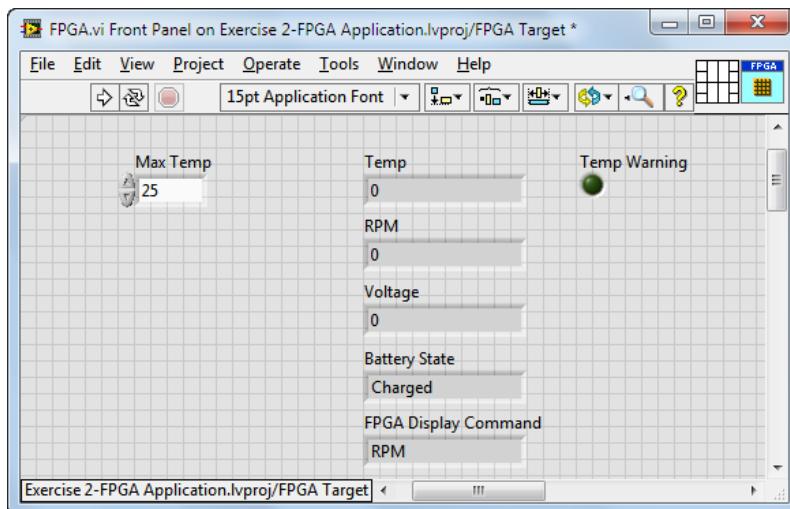
12. Right-click on the left-hand tunnel created on the while loop and select **Replace with Shift Register** from the drop down menu. LabVIEW replaces the tunnel you right-clicked with a shift register terminal, and the cursor becomes a shift register icon (▣). Hover over the tunnel on the opposite side of the loop until it flashes, then click the tunnel to replace it with a shift register. A shift register enables the passing of data from one loop iteration to the next.



13. To communicate with the real-time application, switch to the front panel of the FPGA VI and drag in the following controls and indicators from the directed location and rename with the exact spelling, capitalization, and spacing as shown in the table below:

Control/Indicator Type	Palette Location	Name
Boolean Indicator	Modern»Boolean»Round LED	Temp Warning
Numeric Indicator	Modern»Numeric»Numeric Indicator	Temp
Numeric Indicator	Modern»Numeric»Numeric Indicator	RPM
Numeric Indicator	Modern»Numeric»Numeric Indicator	Voltage
Numeric Control	Modern»Numeric»Numeric Control	Max Temp
Custom Indicator	From Project Explorer»FPGA Custom Controls»BatteryState.ctl	Battery State
Custom Indicator	From Project Explorer»FPGA Custom Controls»DisplayState.ctl	FPGA Display Command

Note: For the custom indicators navigate to the Project Explorer window, expand the NI-RIO Evaluation HW»Chassis»FPGA Target»FPGA Custom Controls folder and drag in the specified items.

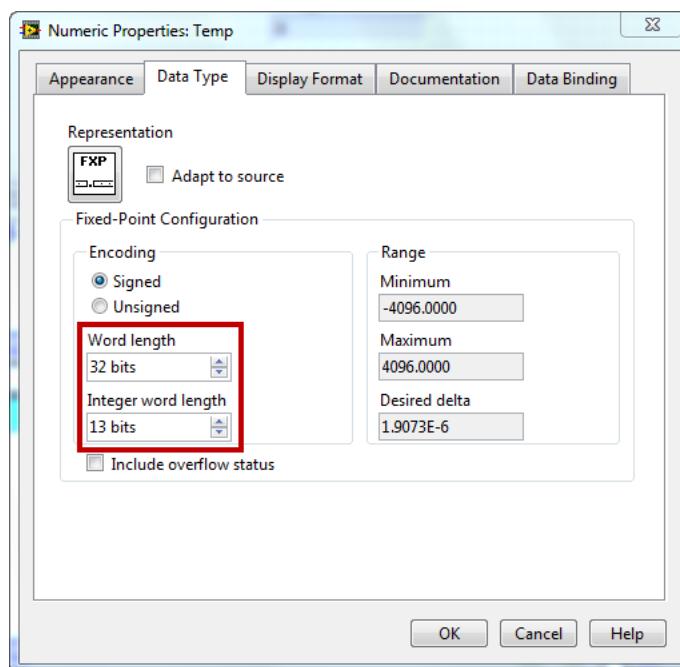


14. By default the custom controls are dropped down as controls on the front panel. Right-click on **Battery State** and select **Change to Indicator**. Repeat this for the **FPGA Display Command** control.
15. Right-click on the **Temp** indicator, select **Properties**, and click on the **Data Type** tab. Click on the Representation icon that shows I16, and in the menu that appears select **FXP**.

Note: The FPGA will be acquiring data in decimal format, however in contrast to microprocessors, FPGAs do not inherently have floating point processing units on-chip, so instead the fixed point data type is commonly used to represent non-integer values. To learn more about fixed-point numbers search *Fixed Point Numbers* on ni.com.

16. Enter the following values to define the **Encoding** value, **Integer word length** and the **Word length** which communicates to LabVIEW the maximum size of the integer portion of the data and the maximum overall integer plus decimal representation required.

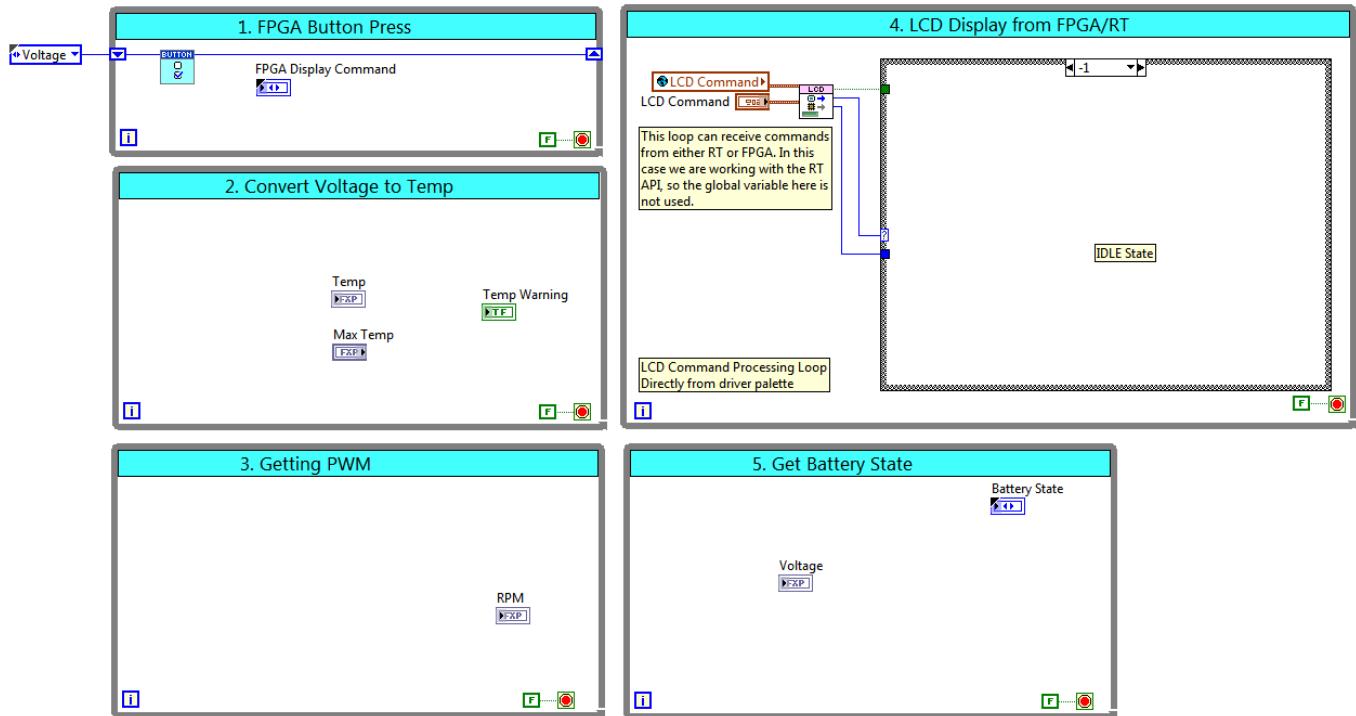
Encoding: Signed
Word length: 32 bits
Integer word length: 13 bits



17. Repeat steps 15 and 16 for the two remaining numeric indicators and one numeric control with the following **Encoding** values, **Word lengths**, and **Integer word lengths**:

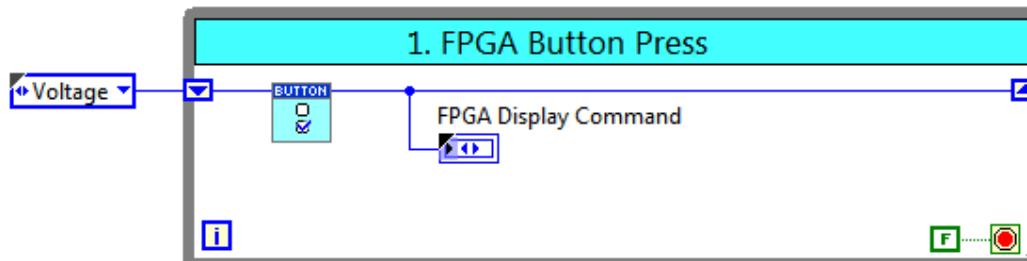
Control/Indicator Name	Encoding	Word length (bits)	Integer word length (bits)
RPM	Unsigned	32	16
Voltage	Signed	24	5
Max Temp	Signed	32	13

18. Toggle back to the block diagram and arrange the controls and indicators as shown in the figure below.



19. Branch the **Display State Out** output wire of the **State Selector** subVI and connect it to the **FPGA Display Command** indicator.

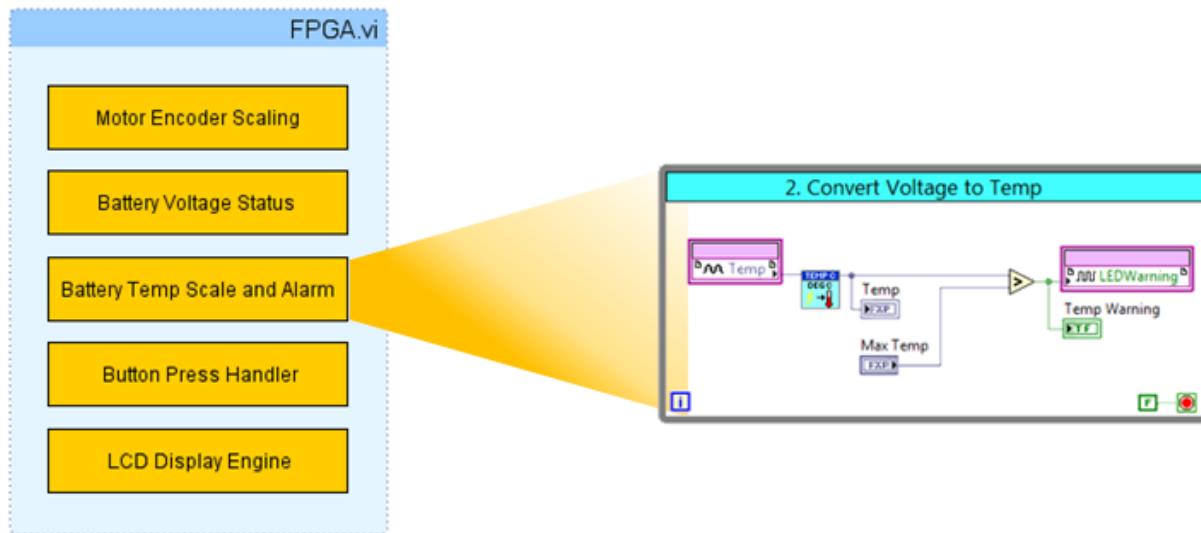
The **FPGA Button Press** handling loop is now complete and should look as follows:



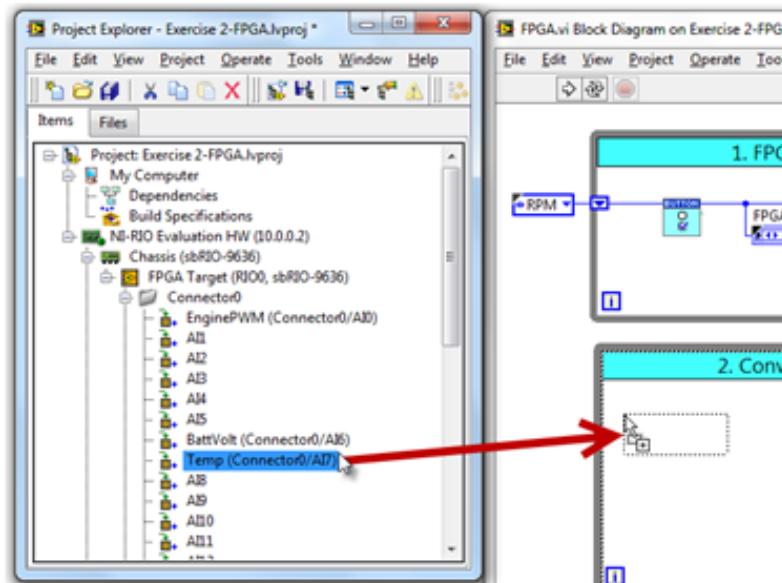
20. Save the FPGA VI by selecting **File»Save** or pressing **CTRL+S**.

Monitor the Battery Cell Temperature

The next while loop will acquire the battery cell temperature voltage, scale it to degrees Celsius and compare it against a maximum temperature for alarming purposes.

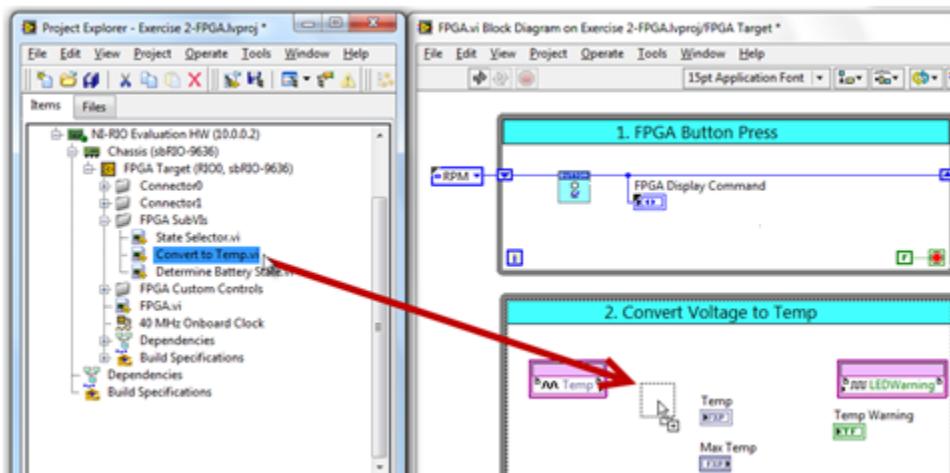


1. In the Project Explorer window under the FPGA Target expand out **Connector0** which exposes the I/O channels available from that connector. Locate the **Temp** (**Connector0/AI7**) channel and drag it into the left-hand side of the **Convert Voltage to Temp** while loop.

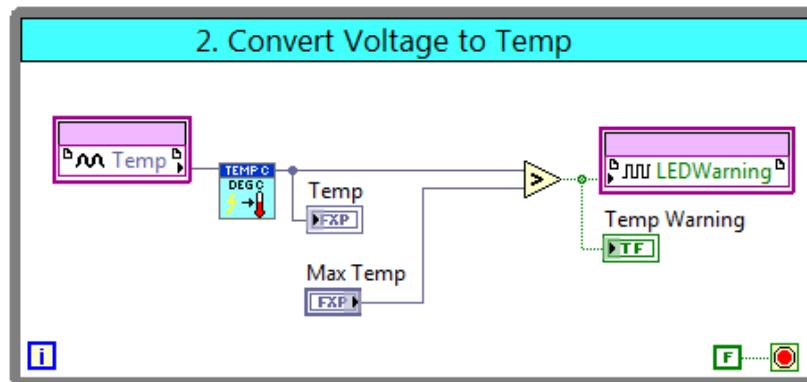


2. Still in the Project Explorer window, expand out **Connector1** and locate the **LEDWarning** (**Connector1/DIO9**) digital channel. Drag and insert it on the right-hand side of the while loop.

3. Right-click on the center of the **LEDWarning** I/O node and change it to write mode by selecting **Change to Write** from the menu that appears.
4. To re-use IP already generated to scale the temperature units, expand the **FPGA SubVIs** folder under the **FPGA Target** in the Project Explorer, and select the **Convert to Temp.vi**.
5. Drag the **Convert to Temp.vi** into the middle of the **Convert Voltage to Temp** while loop.



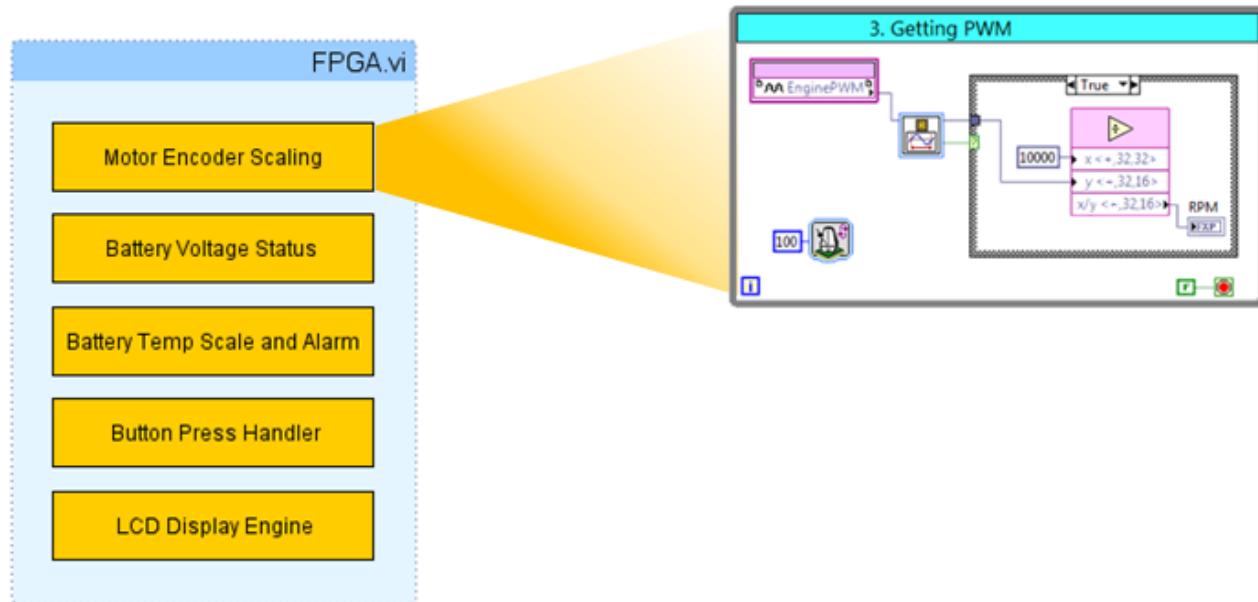
6. Insert a **Greater?** logic comparison (Functions»Programming»Comparison) into the while loop and wire up all the components in the loop as shown below.



7. Save the FPGA VI by selecting **File»Save** or pressing **CTRL+S**.

Acquire the Motor Encoder PWM and Convert to RPMs

Next the motor encoder PWM signal (simulated by a square wave signal generator) will be captured on an input channel then converted to number of periods using a built-in **Analog Period Measurement VI**. To convert the simulated signal to RPMs, the acquisition rate is then divided by the number of periods calculated to get an RPM value.



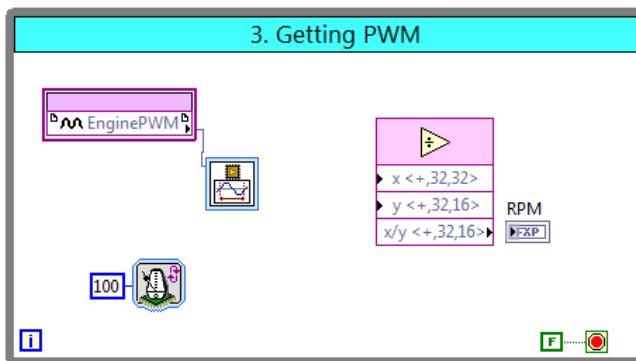
1. In the Project Explorer window under the FPGA Target expand out **Connector0** and drag **EnginePWM** (Connector0/AI0) into the left-hand side of the **Getting PWM** while loop.
2. Locate the **Analog Period Measurement VI** from the **Functions»Programming»FPGA Math & Analysis** palette and place it to the right of the **EnginePWM** I/O node.

After insertion, the Analog Period Measurement VI configuration dialog box will appear, click **Cancel**. Wire the **EnginePWM** output to the **input data** input terminal on the **Analog Period Measurement VI**.

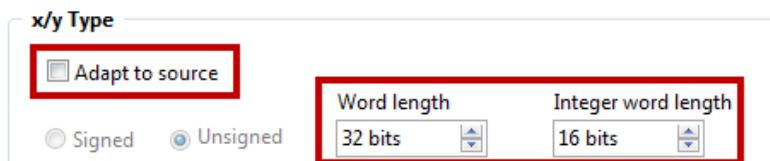
3. Now double-click on the **Analog Period Measurement VI** to again bring up the configuration dialog box. Configure it with these settings:

Level: 1
Hysteresis: 0.5
Interpolate crossings: Checked
Direction: Rising
Number of periods: 1

4. Drop down a **Loop Timer** VI (Functions»Programming»Timing), and in the dialog box that appears, specify a Counter Unit of **usec**. Leave the size of internal counter at its default
5. Right-click on the **Count (usec)** input of the **Loop Timer** and select **Create»Constant**. Enter a value of **100**, so that the acquisition loop for the EnginePWM will run at 100 microseconds (10kHz rate).
6. Insert the **High Throughput Divide** VI (Functions»Programming»FPGA Math & Analysis »High Throughput Math) on the right-hand side of the **Getting PWM** while loop. This will divide the 10 kHz acquisition rate by the number of PWM periods that occurred during that timeframe, in order to get the RPM frequency.

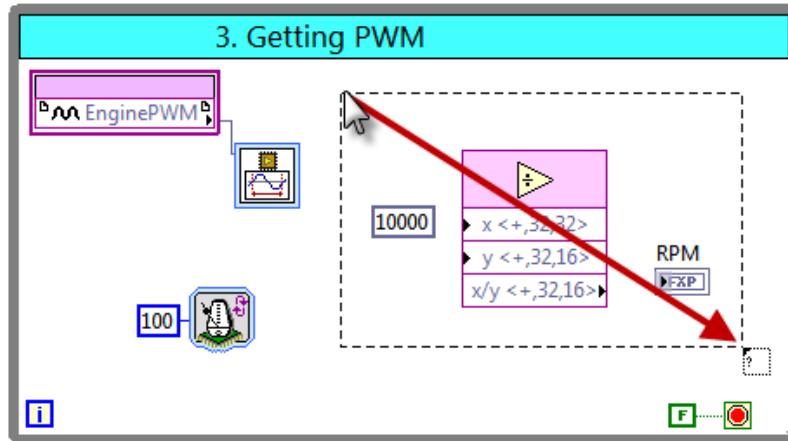


7. Double-click on the **High Throughput Divide** VI to bring up the function's configuration and set the Fixed-Point Configuration of the x/y output to Word length: **32 bits** and Integer word length: **16 bits**.



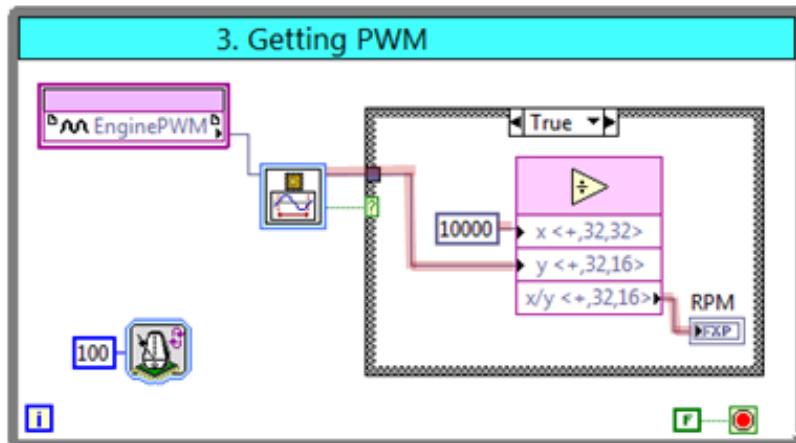
8. Drop down a **Numeric Constant** (Functions»Programming»Numeric) next to the x input of the High Throughput Divide VI and set its value to be **10000**.
9. To change the default numeric to fixed-point:
 - ✓ Right-click on the constant, select **Properties** from the shortcut menu, and then select the **Data Type** tab.
 - ✓ Click on the icon underneath the Representation heading, select **FXP**, and uncheck the **Adapt to entered data** checkbox.
 - ✓ Enter **32 bits** as the Word length and **32 bits** as the Integer word length (in this case there will not be any decimal represented). Set the Encoding to be **Unsigned**. Click **OK** to exit the properties dialog.

10. Drag a **Case Structure** around the constant, High Throughput Divide and RPM indicator.



11. Wire the **output valid Boolean** output of the **Analog Period Measurement VI** to the **case selector** of the case structure (input terminal with a question mark on it), so that the division and RPM value update occurs only when the period measurement calculation is valid. Ensure there is nothing in the false case of the Case Structure by left-clicking on the Case Selector label that currently says **True**.

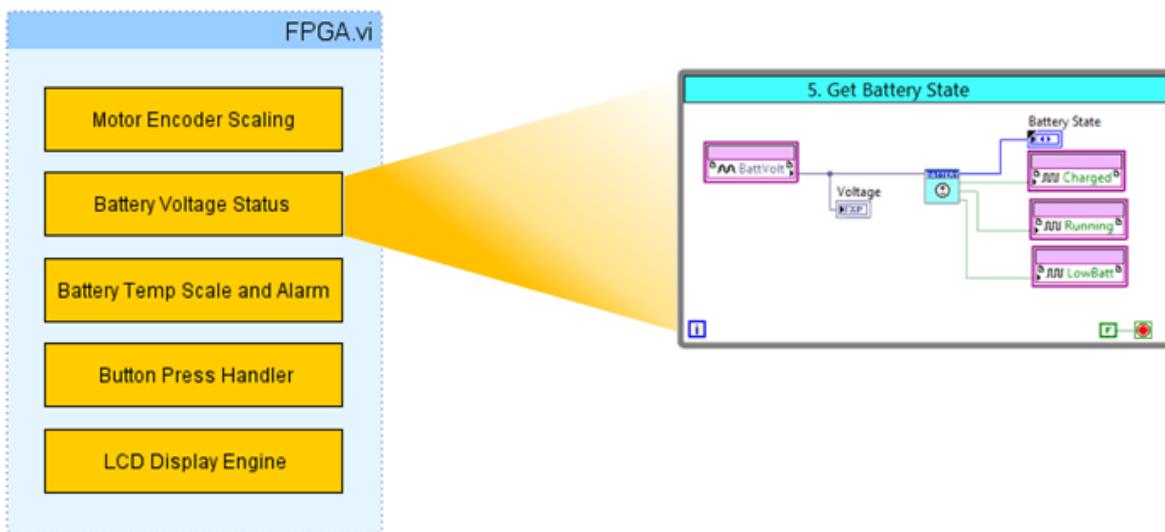
12. Wire the rest of the components in the loop as highlighted in red below.



13. Save the FPGA VI by selecting **File»Save** or pressing **CTRL+S**.

Monitor the Battery Cell Voltage and Control Power Distribution

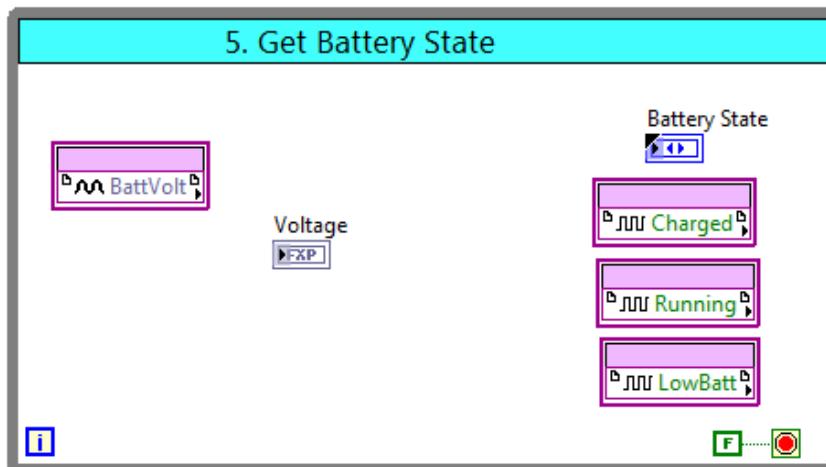
The last FPGA-based task is to acquire the current battery cell voltage, and based on its level assert digital output lines in order to control power distribution to charge the cell, draw off the cell, or throw an error if it has reached the voltage limit.



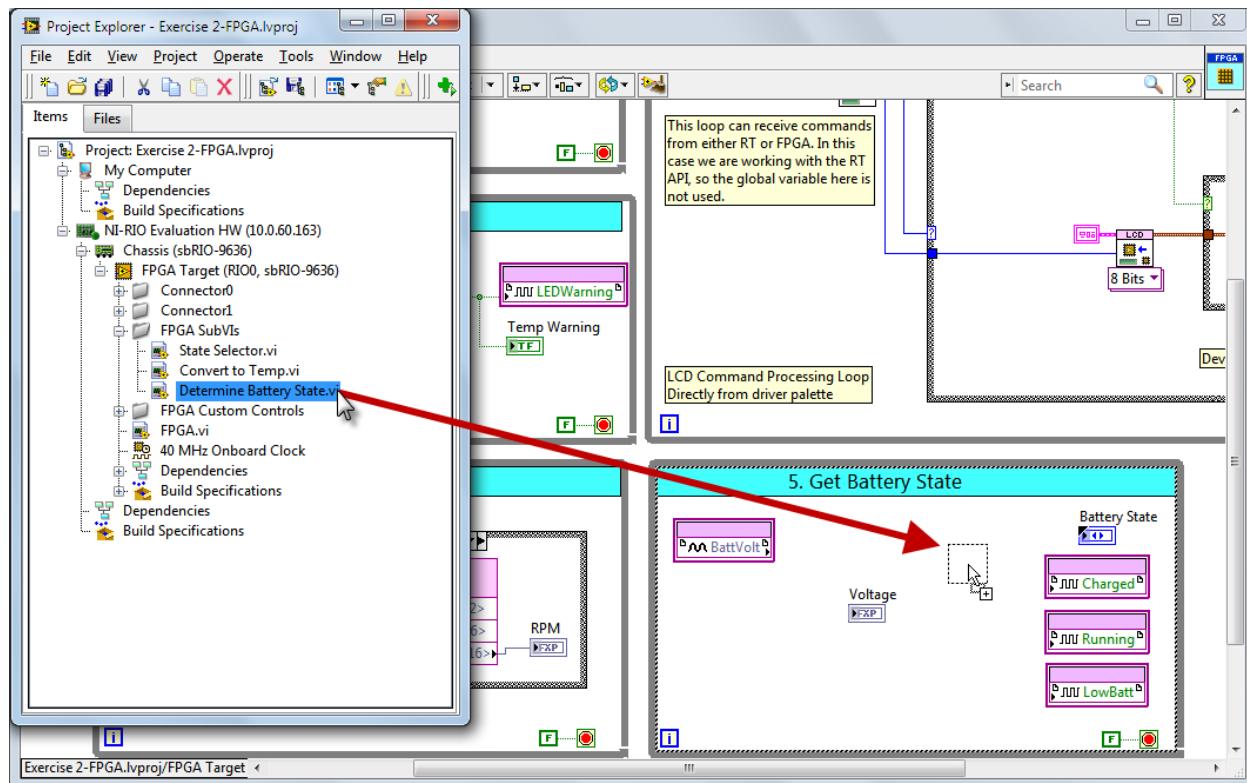
1. In the Project Explorer window under the FPGA Target expand out **Connector0** and drag **BattVolt (Connector0/AI6)** into the left-hand side of the **Get Battery State** while loop.
2. To assert the digital lines, expand out **Connector1** in the Project Explorer and drag over the following digital lines into the right-hand side of the same loop.

Digital Lines

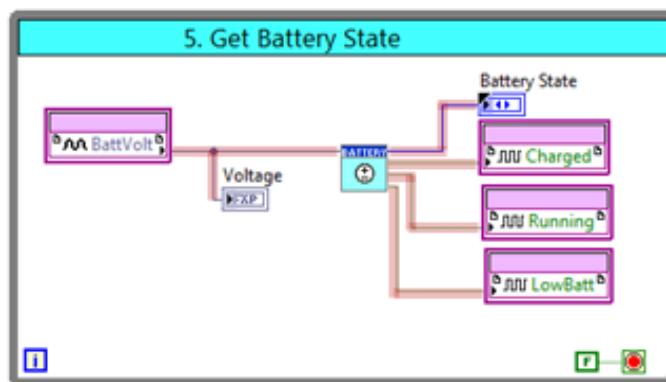
Connector1»Charged (Connector1/DIO6)
 Connector1»Running (Connector1/DIO5)
 Connector1»LowBatt (Connector1/DIO4)



3. Hover over the **Charged** digital I/O node that was inserted in the last step until the cursor changes to a hand. Then right-click on the node and select **Change to Write**.
4. Repeat step 3 for the remaining two digital I/O nodes.
5. In the Project Explorer window, under the **FPGA Target** expand out the **FPGA SubVIs** folder and drag the **Determine Battery State.vi** into the middle of the **Get Battery State** while loop.



6. Wire up the components in the **Get Battery State** while loop as highlighted in red below, with the Fully Charged, Running, and Low Battery subVI outputs wired to the appropriate I/O node.



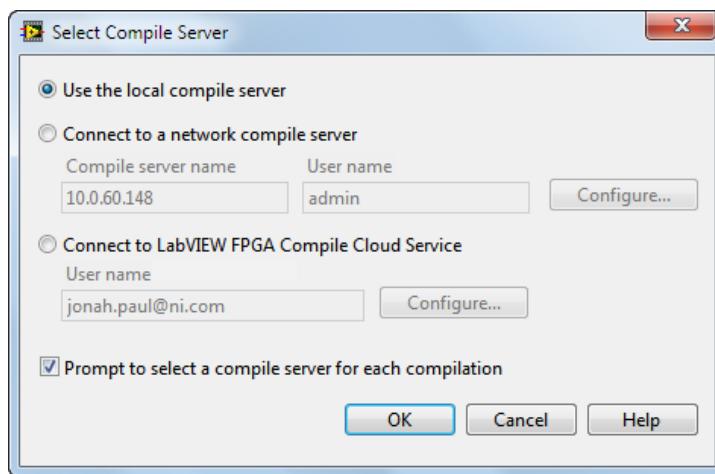
7. Save the FPGA VI by selecting **File»Save** or pressing **CTRL+S**.

Start LabVIEW FPGA Compilation Process

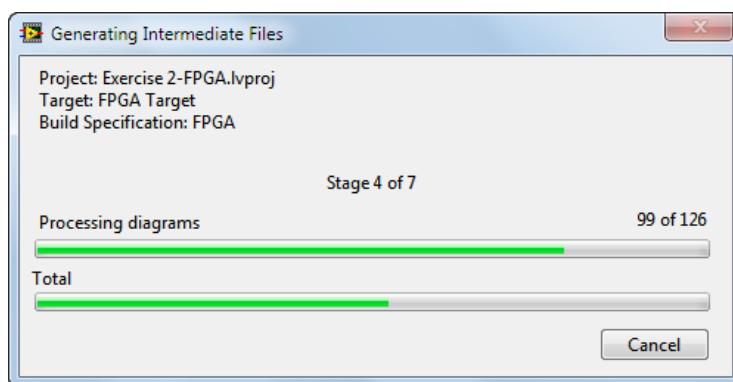
1. Save the FPGA VI and click the **Run** button to start the compilation process.

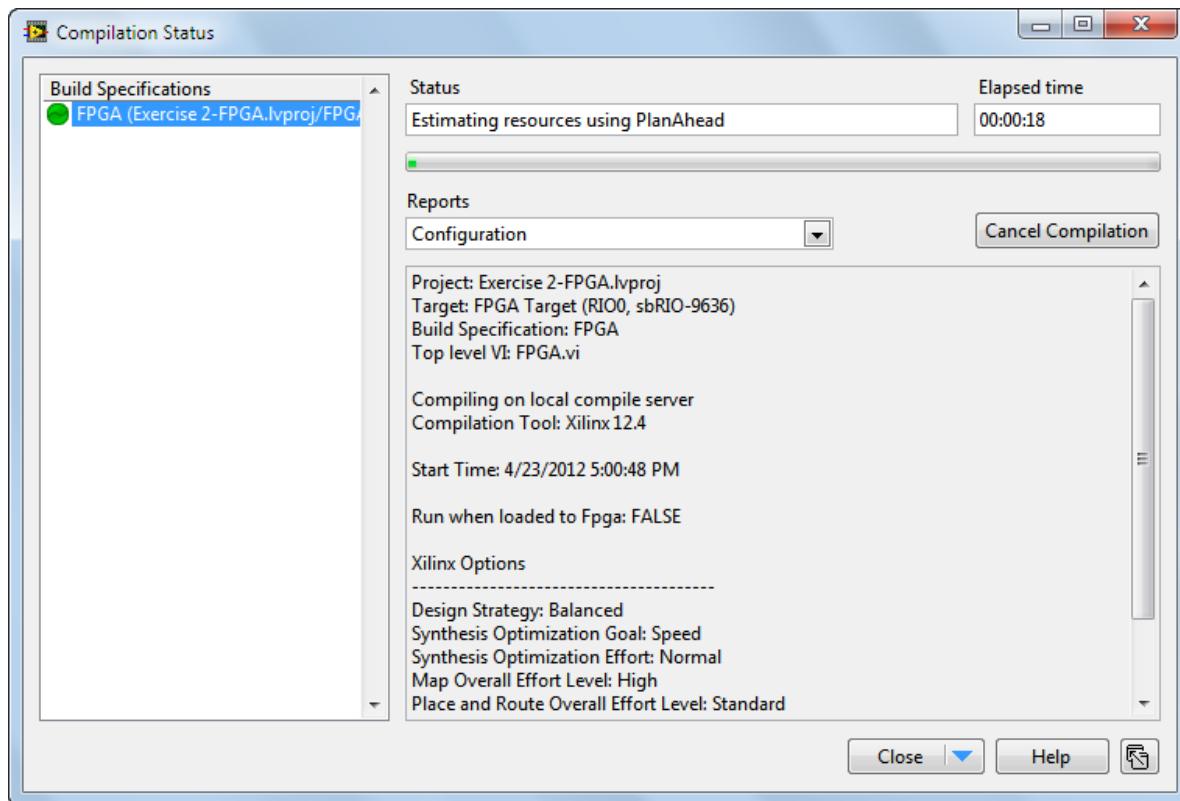
Note: In contrast to LabVIEW applications running on a Windows OS, an FPGA-based VI is compiled down to a bitfile, which is loaded onto the FPGA chip configuring its logic cells and I/O blocks to implement the requested logic. This is a two-step process. First LabVIEW generates intermediate files and then it transfers them to the Xilinx compilation tools for the final compile stage.

2. In this case, select **Use the local compile server** option. The FPGA compilation process can also be offloaded onto a remote machine, farm servers, or to the cloud.



3. The intermediate file generation process will then start, and once complete, will kick off the Xilinx compilation. In total, the process should take about 20 minutes to finish compilation.





Note: If a communication error occurs when the Compile Server starts, manually start the Compile Worker by navigating in the Windows start menu to **All Programs»National Instruments»FPGA»FPGA Compile Worker**. Then, once it starts up, click the **Run** button on your LabVIEW FPGA VI again to re-establish communication.

You have now finished development of a LabVIEW FPGA application to monitor and control the prototype battery cell, motor, and attached display!

Learn more about FPGA Fundamentals at ni.com/fpga

Next Step

While you are waiting for your FPGA application to compile, continue on to the development of your real-time application in Exercise 3.

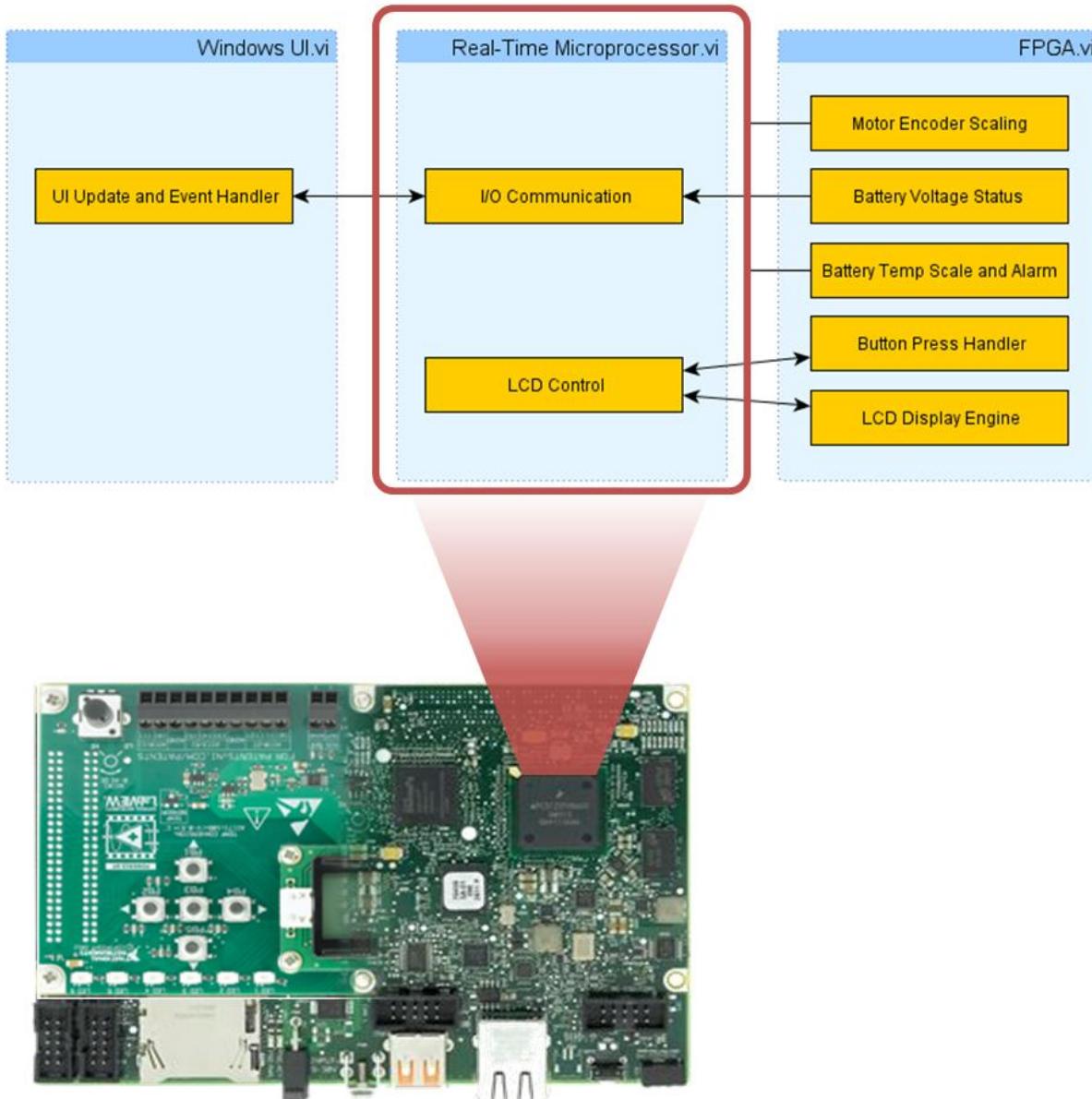
Exercise 3 | Develop Real-Time Application

Summary

In this exercise you are going to create a real-time application running on a processor which communicates with the FPGA controlling the LCD screen. It will also acquire data captured by the FPGA from the I/O and coordinate network communication back to your Windows user interface.

1. Communicate with the FPGA
2. Forward Data onto the Windows Target
3. Implement Decision Logic for LCD Screen
4. (Optional) Test the Real-Time and FPGA Applications

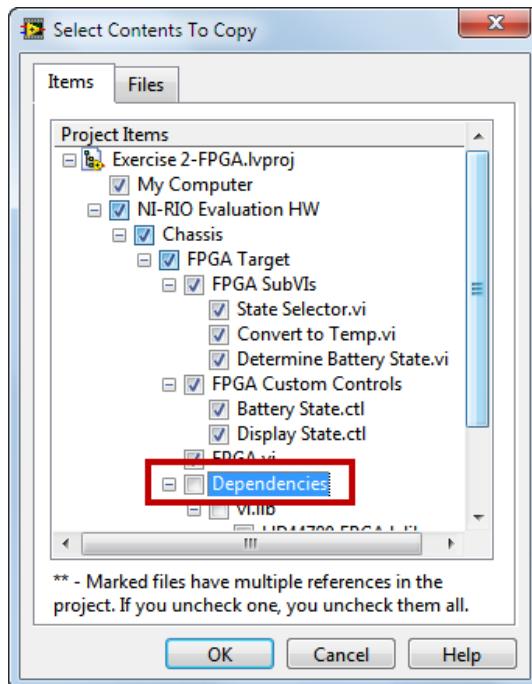
What am I going to accomplish in this exercise?



In this exercise you will complete two processes running on the real-time microprocessor. The first one will transfer the values from the FPGA and then forward them on to the Windows user interface through network communication. The second process, LCD Control, will decide what to write down to the LCD screen based on push button presses.

Project Setup

1. With your Exercise 2 project still open in the Project Explorer Window, select **File»Save As...** to copy the project files into the Exercise 3 folder.
2. In the Save As... dialog box that appears select **Duplicate .lvproj file and contents » Select contents to copy** and click **Continue...**

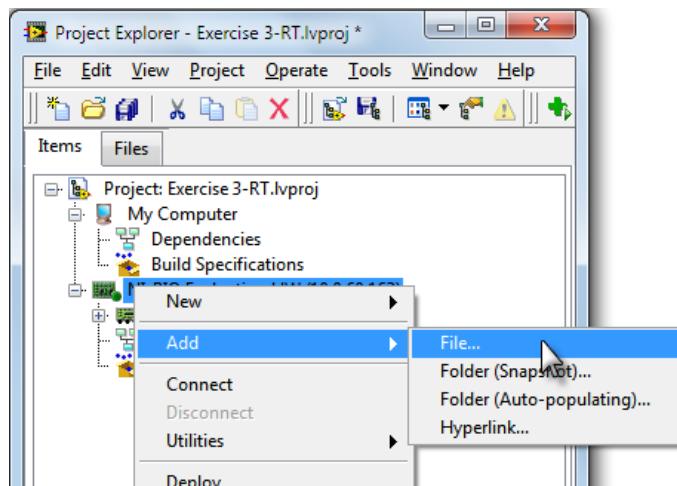


3. Uncheck the *Dependencies* category for the FPGA target hierarchy and select **OK**. These LabVIEW dependency files are not necessary since you will still be saving the files on the development machine.
4. Navigate to the **\3- Create Real-Time Application** folder and rename your project as ***Exercise 3- RT.lvproj***. Click **OK** to save the project and supporting files.

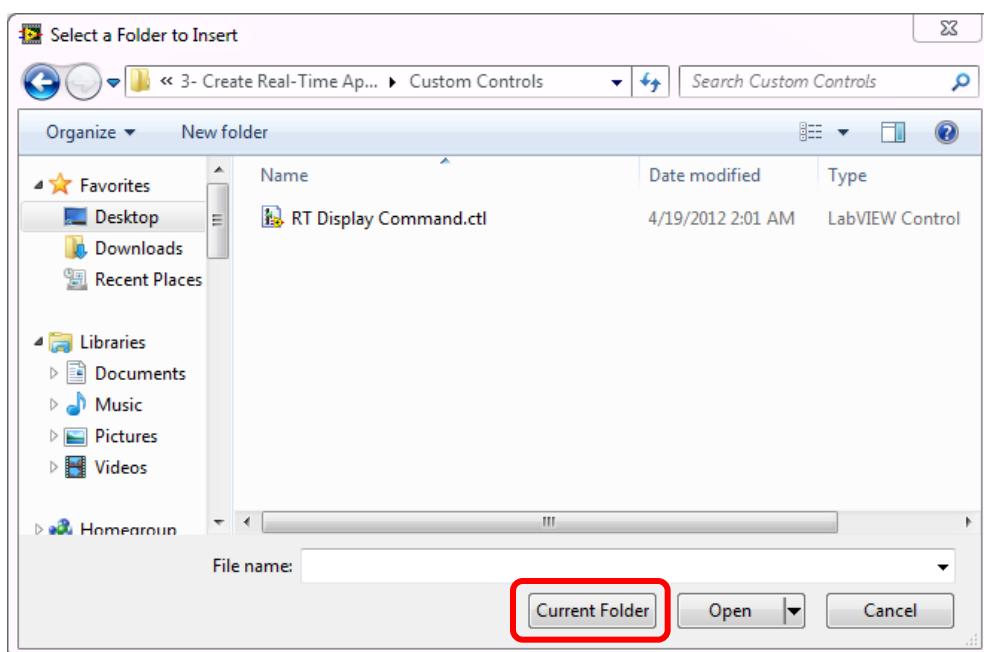
Note: Click **OK** if asked to save over existing files

5. Leave the Exercise 2 project open so that the FPGA VI can compile and open the ***Exercise 3- RT.lvproj*** from **\3- Create Real-Time Application** that was just created.

6. In the Project Explorer window, right-click on *NI-RIO Evaluation HW* and select **Add»File...** to add the RT application starting template, Navigate to the *.\3- Create Real-Time Application\VI*s folder and select the **RT Microprocessor.vi**.



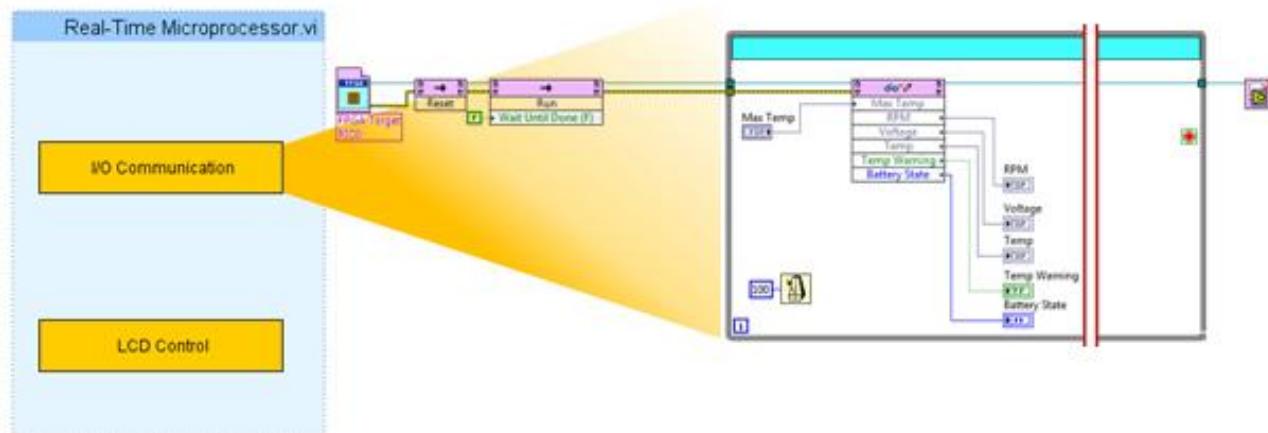
7. To include the network published shared variables used to communicate with the Windows UI, right-click on *NI-RIO Evaluation HW* and select **Add»File...** Navigate to the *.\3- Create Real-Time Application* folder and select **RT-Host Shared Variables.lvlib**.
8. Right-click again on *NI-RIO Evaluation HW* and select **Add»Folder (Snapshot)...** Navigate into the *.\3- Create Real-Time Application\Custom Controls* folder and select **Current Folder**.



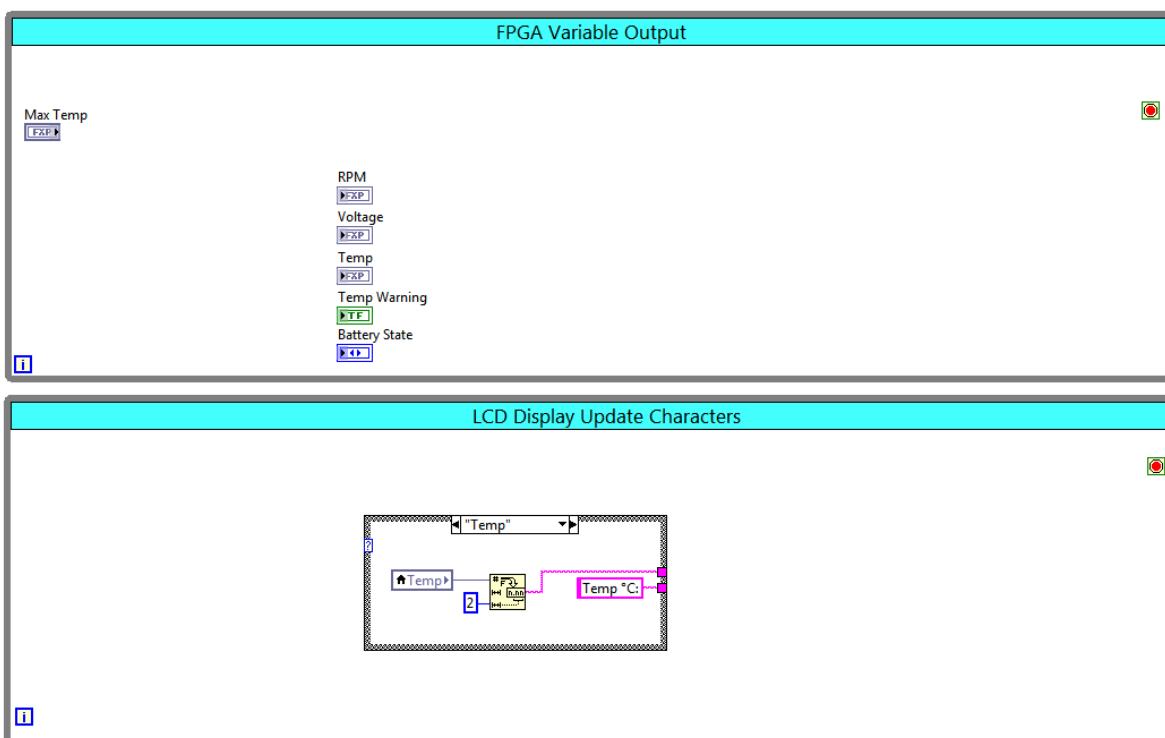
9. Select the **Custom Controls** folder that was just added to the project, press **F2**, and rename it **RT Custom Controls**.

Communicate with the FPGA

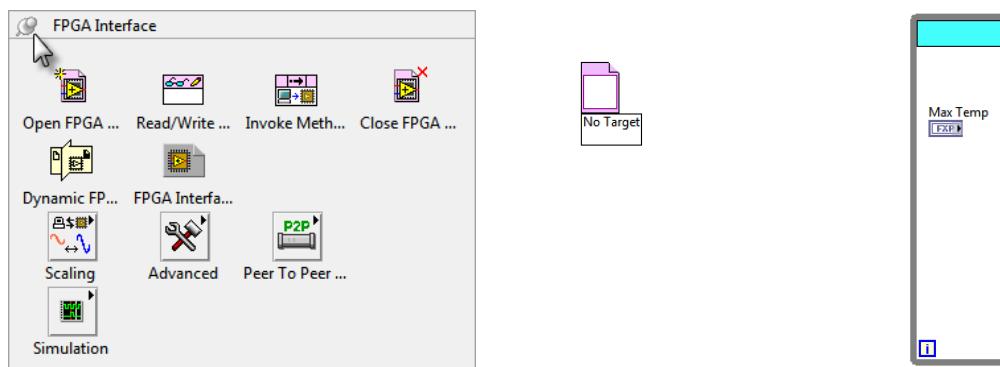
In this section you will open a reference to the FPGA target, implement communication with the target across the backplane bus and close the reference.



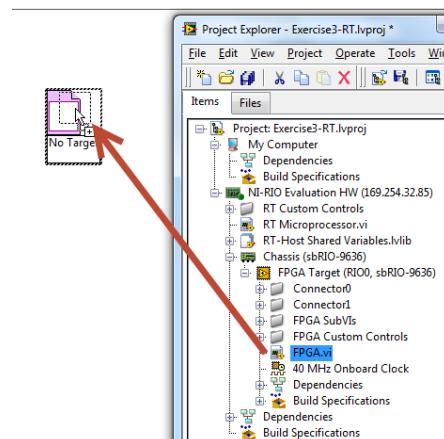
1. Open up the RT Microprocessor VI from the project and toggle to the block diagram (press **CTRL+E**).



2. Navigate to the FPGA Interface palette and pin it to the block diagram. From this palette insert an **Open FPGA VI Reference** (Functions»FPGA Interface»Open FPGA VI Reference) on the block diagram as shown below.

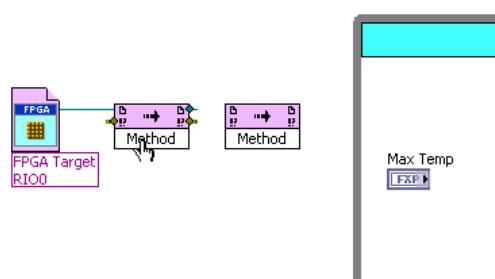


3. From the project find the **FPGA VI** under Chassis»FPGA Target. Drag it into the unconfigured **Open FPGA VI Reference** to the left of the top while loop.

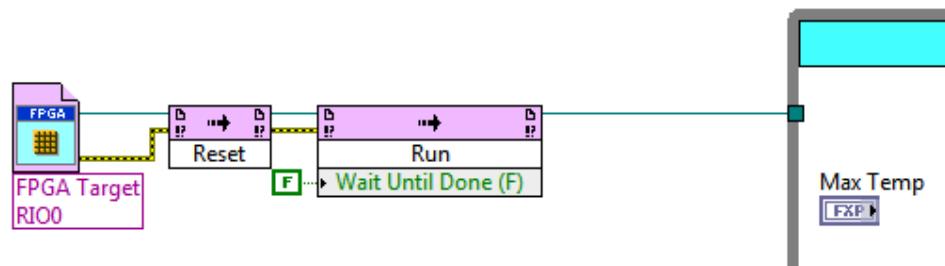


Note: The Run Arrow will be broken because the FPGA VI that you copied over to the project is not yet compiled. In testing you will reference the compiled bitfile from Exercise 2.

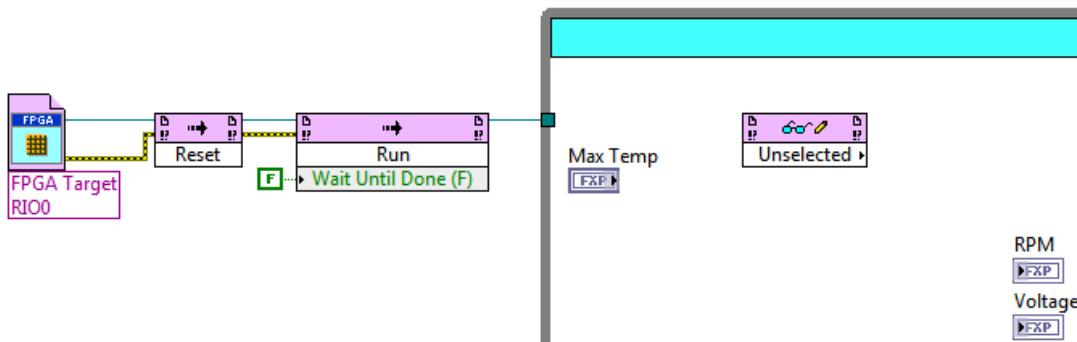
4. Insert two **Invoke Method** nodes (Functions»FPGA Interface) on the block diagram between the Open FPGA VI Reference and the while loop to reset and load the FPGA application.
5. Wire the **FPGA Target reference out** of the Open FPGA VI Reference to the first **Invoke Method**, and then click on the white section titled **Method** and select **Reset**.



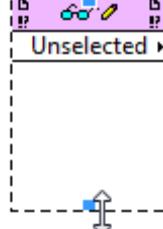
6. Wire the **FPGA VI reference** output of the first **Invoke Method** into the input of the second **Invoke Method** and select **Run** as the method for the second **Invoke Method**.
7. Next wire the FPGA reference output of the Run **Invoke Method** to the border of the **FPGA Variable Output** while loop
8. Right-click on the **Wait Until Done** input of the Run **Invoke Method** and select **Create»Constant**. This should be a False constant by default.
9. Complete the error and FPGA reference wires as shown below.



10. Now insert a **Read/Write Control** (Functions»FPGA Interface»Read/Write Control) inside the **FPGA Variable Output** while loop. This will define the data communicated between the FPGA and real-time application.

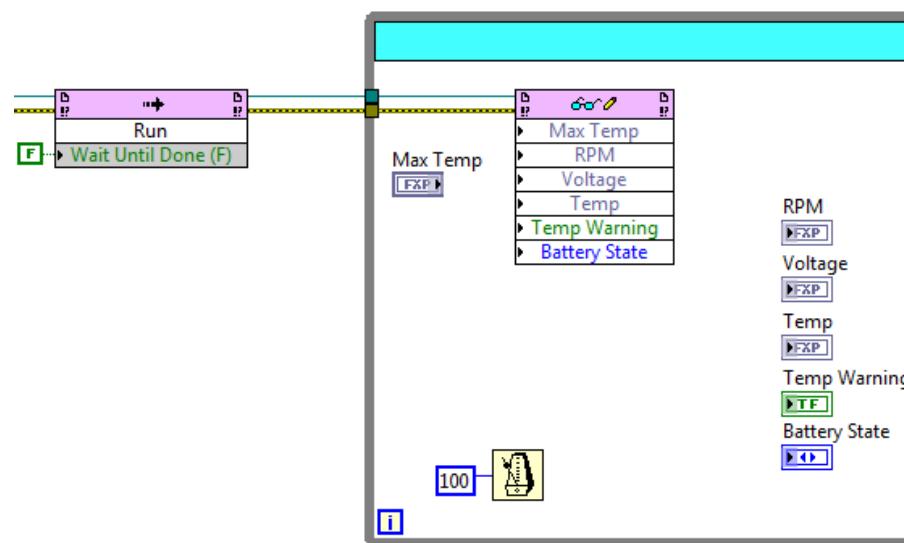


11. Wire the FPGA reference from the loop's tunnel to the **Read/Write Control** input along with the error wire.
12. Expand the **Read/Write Control** to include six terminals by clicking on the lower border and dragging it down as shown below.

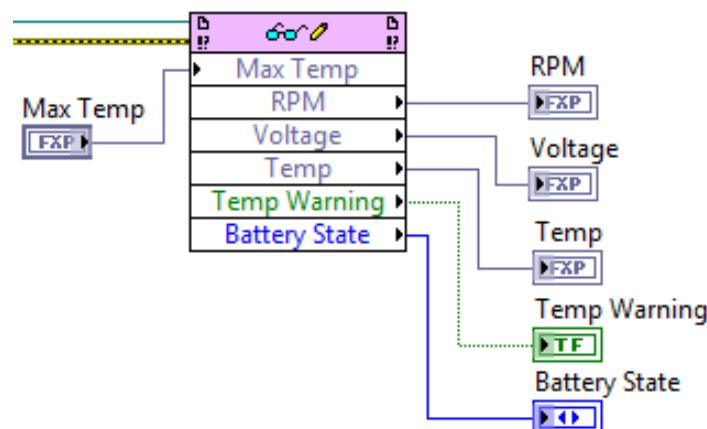


13. To configure the unselected terminals, left-click on each of them and select the control or indicator name from the FPGA front panel that should be polled. Configure the terminals as shown below to include **Max Temp**, **RPM**, **Voltage**, **Temp**, **Temp Warning**, and **Battery State**.
14. Without setting timing in the while loop it will execute as fast as it can, potentially starving resources allocated to the other loop. To avoid this insert a **Wait Until Next ms Multiple** (Functions»Programming»Timing) function inside the loop. Right-click on its **millisecond multiple** input and select **Create»Constant**.

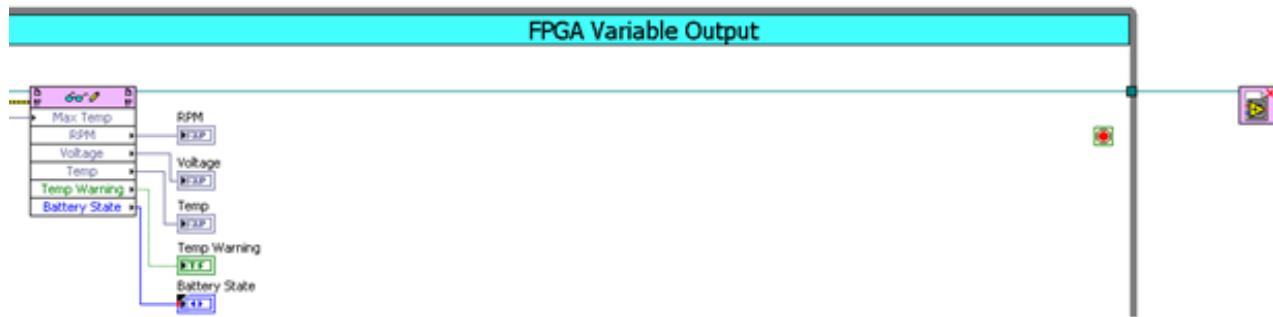
Enter a value of **100** in the constant to execute the loop every 100 milliseconds.



15. By default the **ReadWrite Control** terminals should all be inputs. Right-click on terminals 2-6 and select **Change to Read** to change them to outputs.
16. Wire the terminals on the **ReadWrite Control** to the matching front panel control and indicators on the real-time application VI.



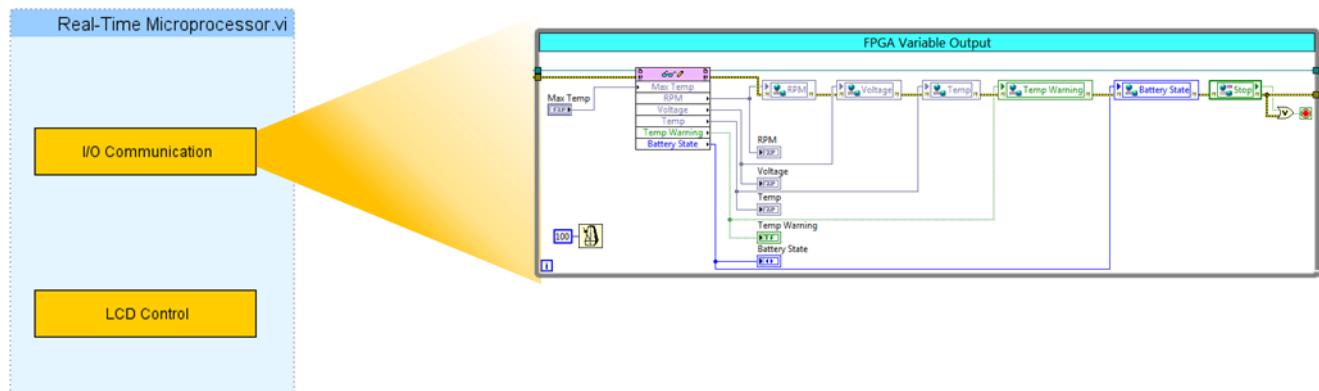
17. Finally, to close out the reference to the FPGA target, insert a **Close FPGA VI Reference** function (Functions»FPGA Interface) to the right of the **FPGA Variable Output** while loop and wire up the **FPGA VI reference** as shown.



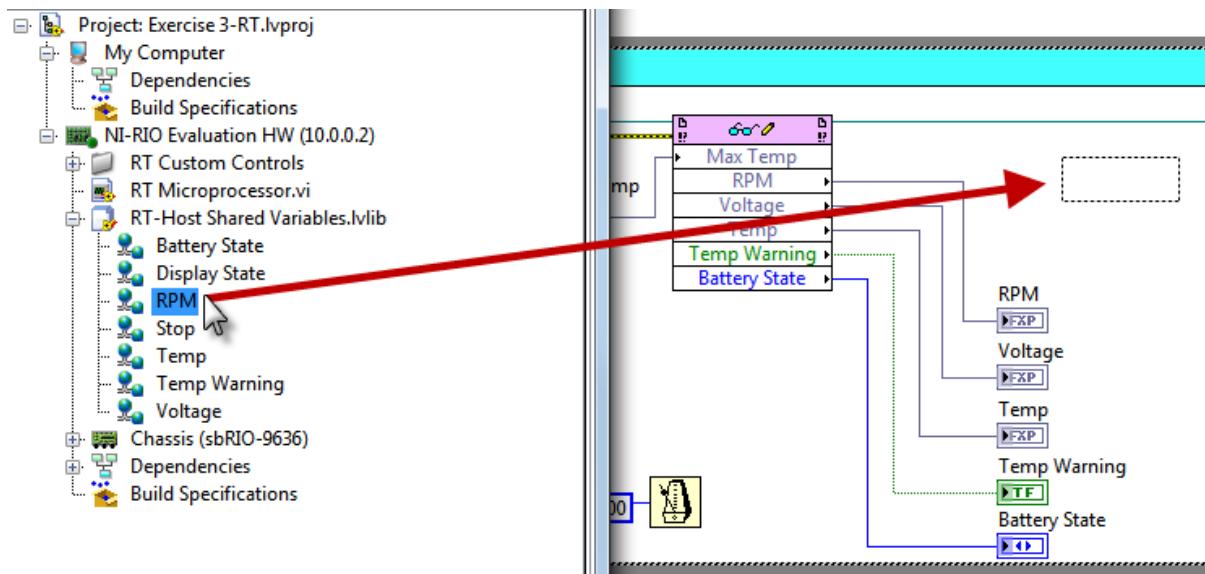
18. Save the Real-Time VI by selecting **File»Save** or pressing **CTRL+S**.

Forward Data onto the Windows Target

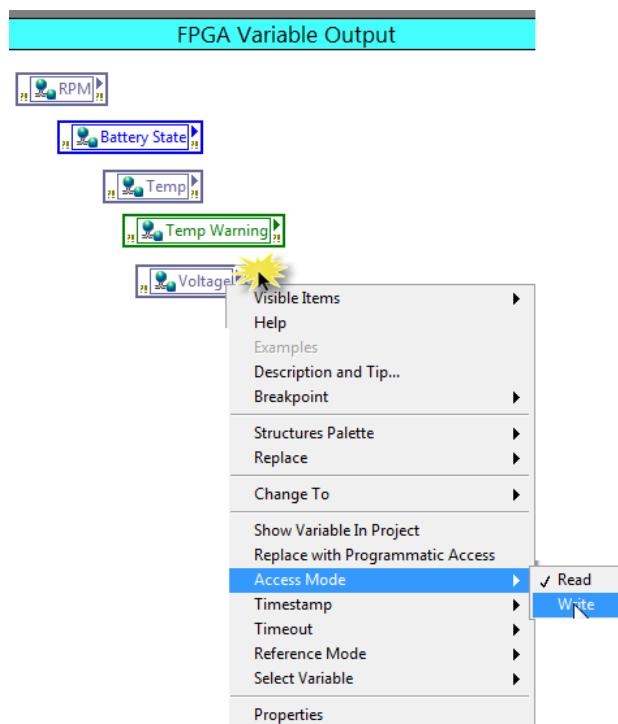
Next you will send the data collected from the FPGA and publish it across the Ethernet connect so that it will be accessible to the Windows user interface that you will develop in the next exercise.



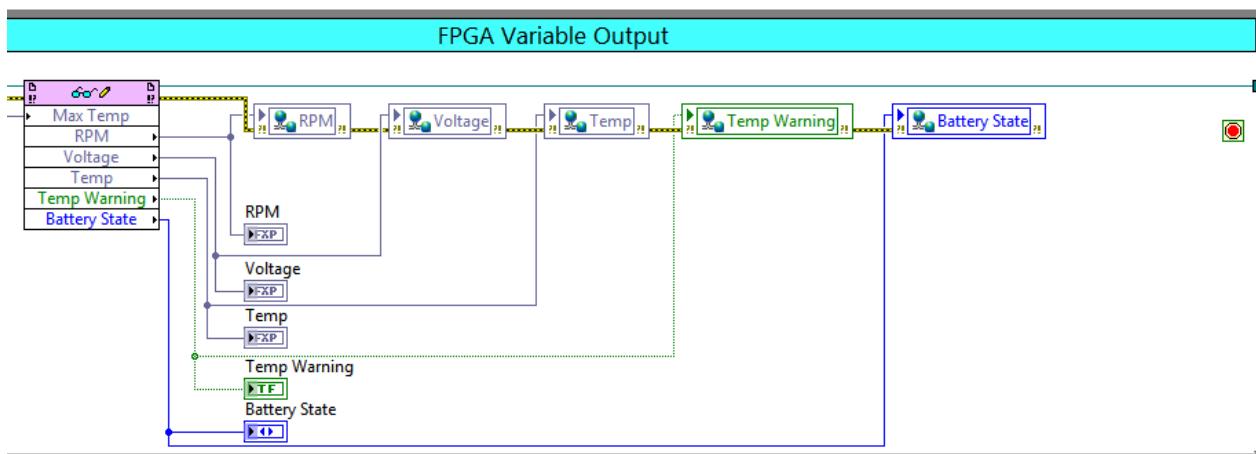
1. Inside the project under the *NI-RIO Evaluation HW* target, locate the **RT-Host Shared Variables.lvlib**. This library contains the **Network Published Shared Variables** that have been created for communication.
2. Expand the library (.lvlib) to show the **Network Published Shared Variables**. Drag an instance of each variable, excluding **Display State** and **Stop**, into the block diagram.



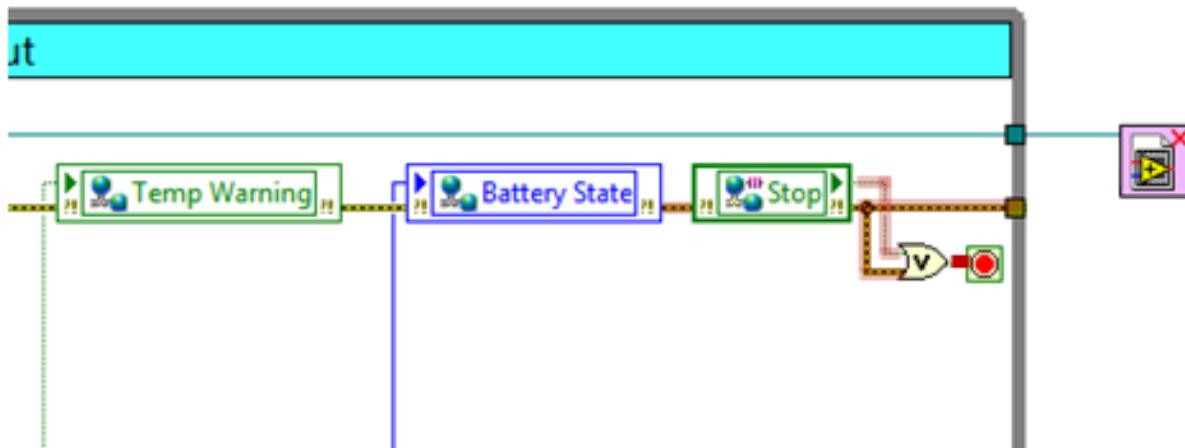
3. By default the shared variable static nodes are in read mode; change the variables to write by selecting one of them, right-clicking, and going to **Access Mode»Write**. Change all the shared variables to be write access mode.



4. Create a wire branch from the FPGA Read/Write control outputs and wire to their respective shared variables as shown in the next screenshot.
 5. In order to enforce dataflow and ensure correct error propagation, wire the **error out** from the **FPGA Read/Write Control** through each of the **Shared Variable** static nodes, in any order.



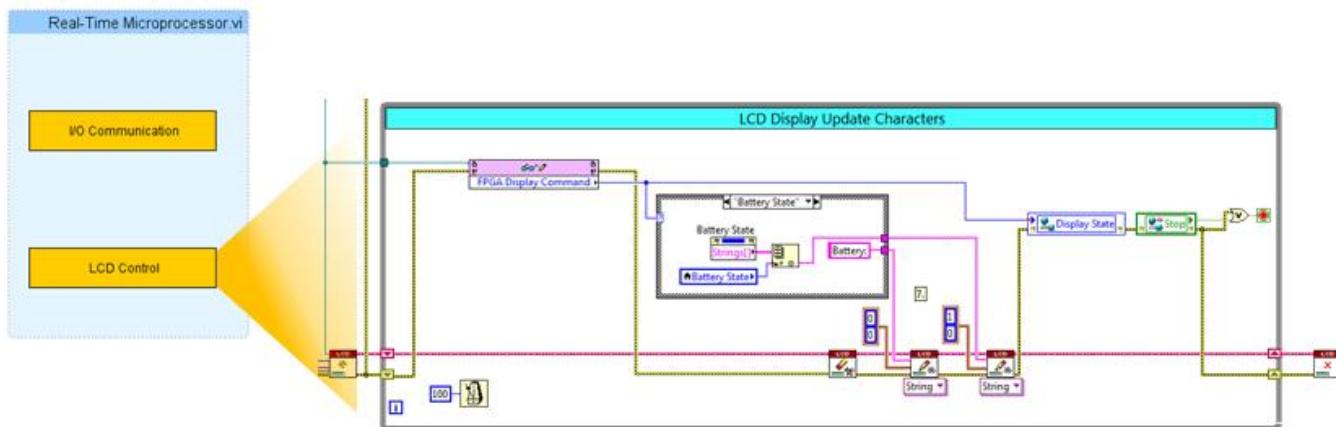
6. Inside the project under the *NI-RIO Evaluation HW* target, locate the **Stop** shared variable inside the **RT-Host Shared Variables.lvlib**. Drag this variable onto the block diagram and insert it to the right of the existing shared variables, as shown below.
7. Insert an **Or** logic function (Functions»Programming»Boolean) to the right of the Stop shared variable. Wire the Error line and Boolean output of the Stop shared variable into the **Or** input terminals and the Boolean output of the **Or** terminal into the red Conditional Terminal as highlighted in red below. This will stop the loop if either the Stop shared variable is true or there is an error.



8. Save the Real-Time VI by selecting **File»Save** or pressing **CTRL+S**.

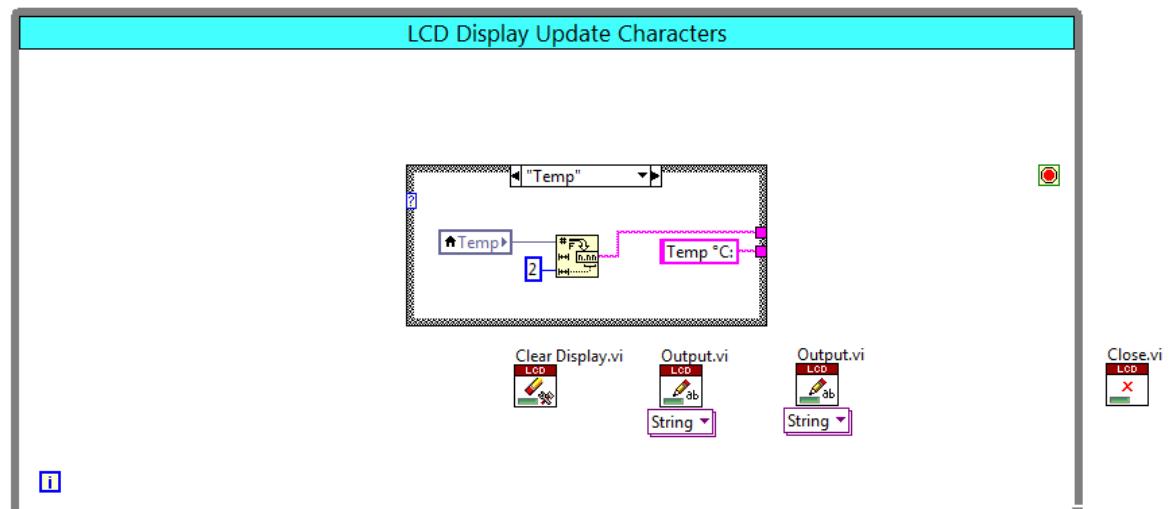
Implement Decision Logic for LCD Screen

The second process, LCD Control, will decide what to write to the LCD screen based on push button presses. You will implement the logic in the next section to control this behavior.

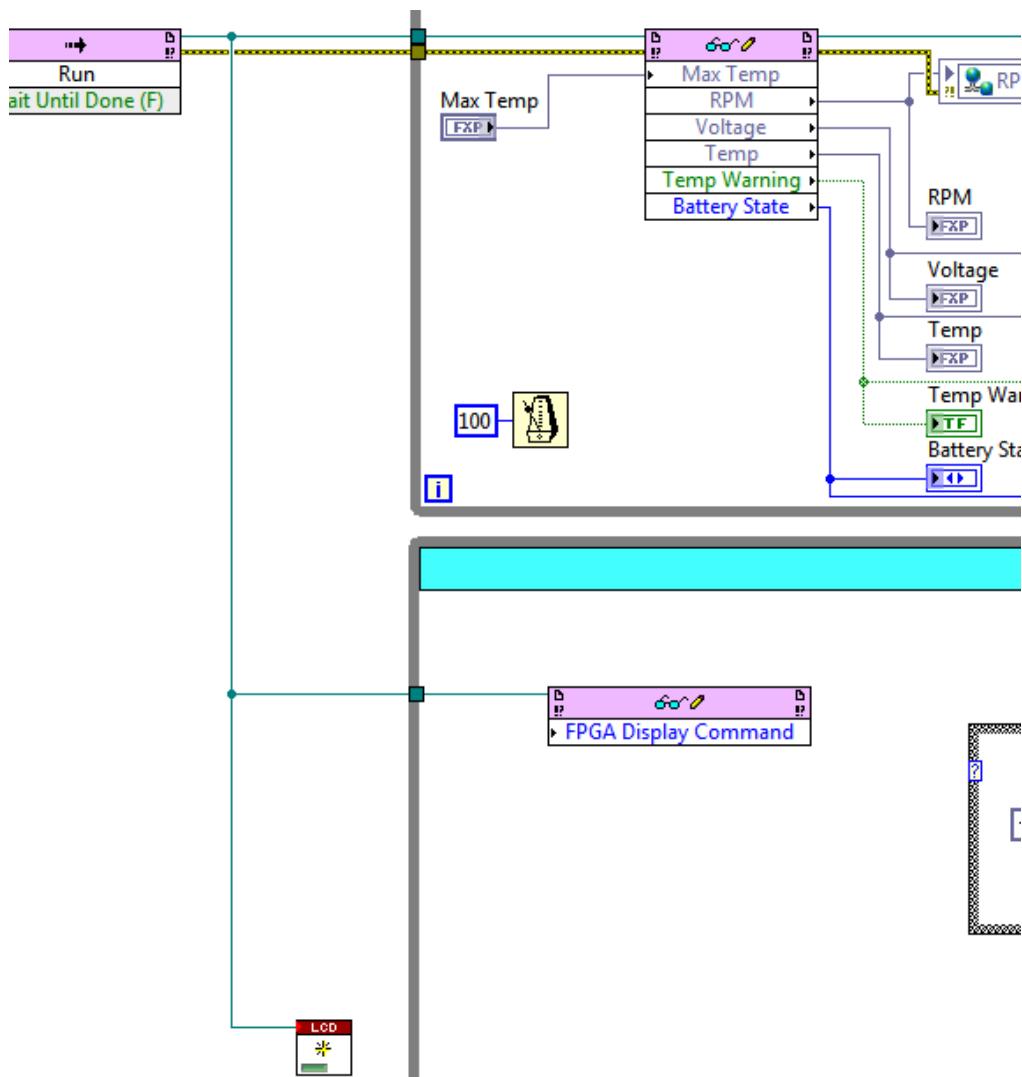


1. To begin, navigate on the Functions palette to the Hitachi HD44780 LCD screen API (Functions»Connectivity»LCD» Hitachi HD44780) and insert the following functions:

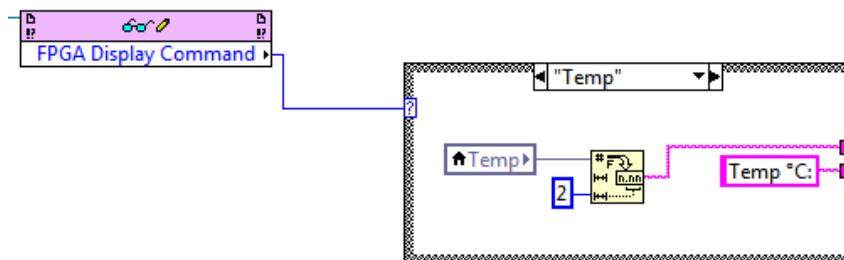
Initialize HD44780.vi
Clear Display.vi (in the Advanced palette)
Two Output.vi
Close.vi



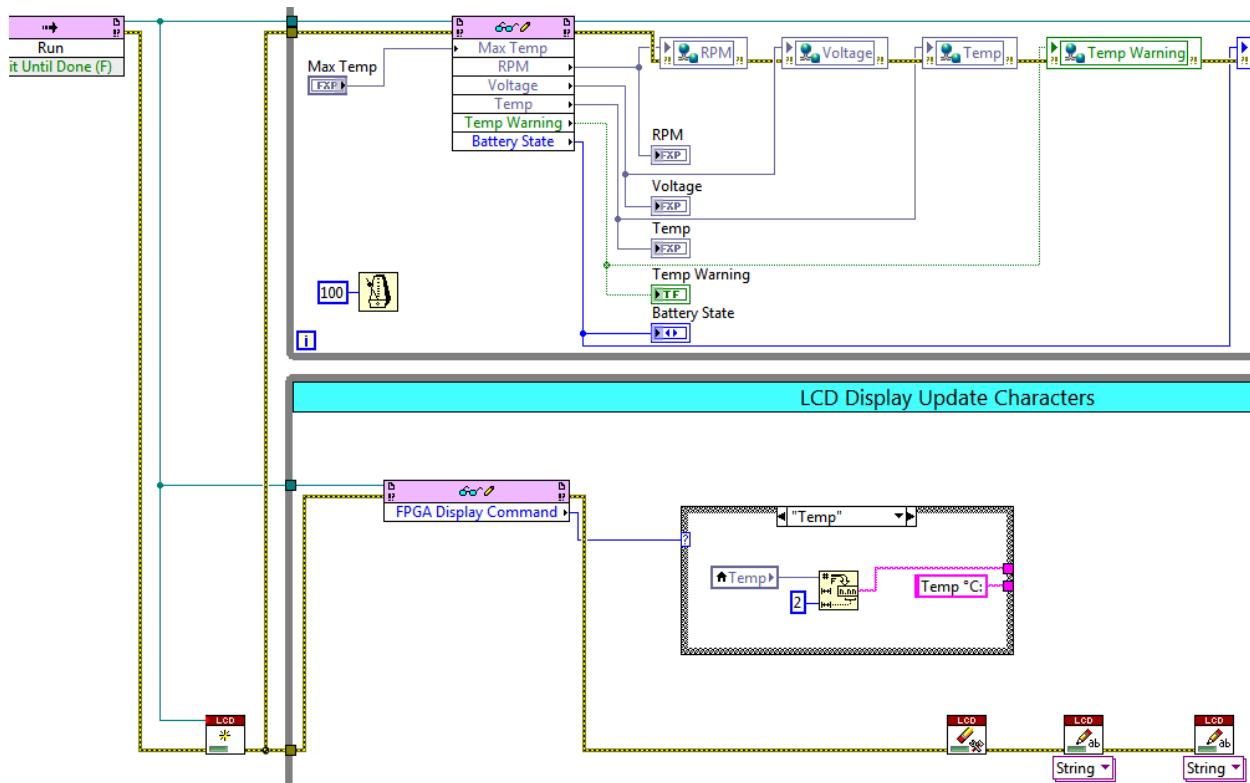
2. Now insert a **ReadWrite Control** (Functions»FPGA Interface) inside the **LCD Display Update Characters** while loop. This will be used to receive the push button display command from the FPGA to decide which string to write to the LCD screen.
3. Branch the FPGA VI Reference wire from after the Run Method next to the top while loop and wire it to the Read/Write Control inserted in the previous step. Also wire it to the **FPGA VI Reference in** input of the Initialize HD44780 VI.
4. To configure the **unselected** terminal of the **ReadWrite Control**, left-click on it and select the name of the control or indicator from the FPGA front panel that should be polled. In this case select the **FPGA Display Command**.



5. By default the terminal should be an input. Right-click on the **FPGA Display Command** terminal and select **Change to Read** to change it to an output.
 6. Wire the FPGA Display Command output to the case selector of the nearby case structure to select the appropriate text to display on the LCD screen. The logic in the case structure polls the chosen shared variable for the most recent value and generates the value and name string to write to the LCD screen.



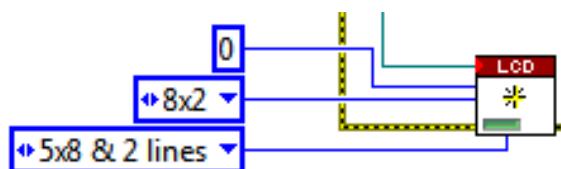
- To propagate any errors through both loops appropriately, wire the Error line through the Initialize HD44780 VI and through both while loops as shown below.



8. Now focusing on the LCD screen write execution, locate the Initialize HD44780 VI to the left of the bottom while loop on the block diagram.

- ✓ Right-click on its **Device Index** input and select **Create»Constant**.
- ✓ Repeat for the **Screen Size** and **Font Configuration** inputs.
- ✓ Set these constants to the following values:

Device Index: 0
 Screen Size: 8x2
 Font Configuration: 5x8 & 2 lines



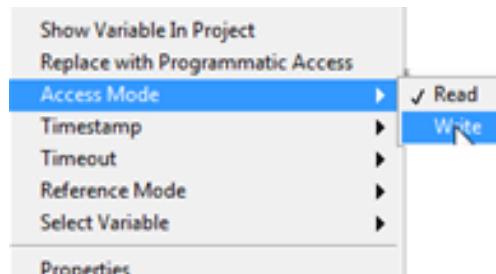
9. Wire the **LCD out** reference of the Initialize HD44780 VI to the **LCD in** of the Clear Display VI. Wire the **LCD out** of the Clear Display to the **LCD in** of the Output VI. Continue this wiring to carry the reference through to the Close VI.



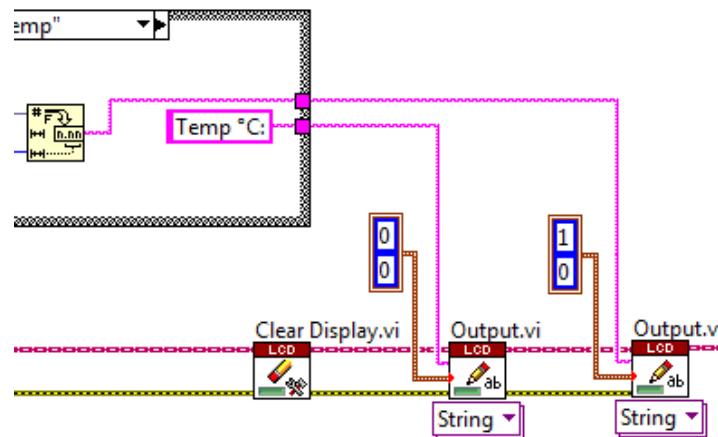
10. Next add network published shared variables to the LCD Display Update Characters while loop to communicate the current LCD display state to the Windows User Interface.

Inside the LabVIEW project under the *NI-RIO Evaluation HW* target, locate the **Display State** and **Stop** shared variables inside the **RT-Host Shared Variables.lvlib**. Drag these variables into the LCD Display Update Characters while loop and insert them to the right of the case structure.

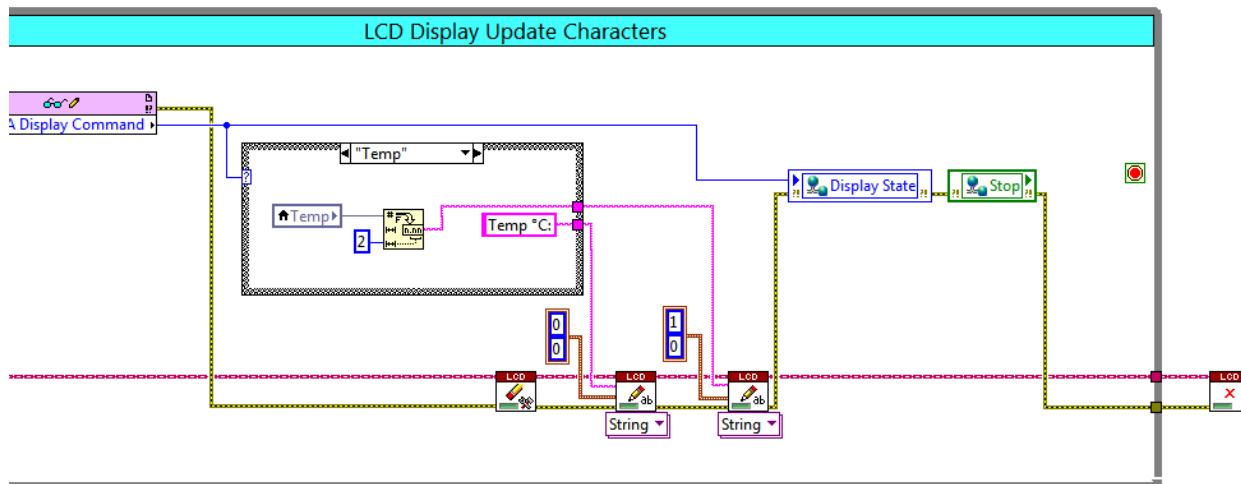
11. By default the shared variable static nodes are in read mode; change the **Display State** variable to write by selecting it, right-clicking, and choosing **Access Mode»Write**.



12. To control what is written to the LCD screen, wire the pink string values from the edge of the case structure to the **text** input terminals on each respective Output VI as shown below.
13. Right-click on the **position** input terminal on each Output VI and select **Create»Constant**. This is a cluster constant with two elements, row and column. Leave the first constant as default [0,0] and change the second Output VI cluster constant to [1,0] as shown below.



14. Create a branch off the **FPGA Display Command** output and wire to the **Display State** shared variable along with the error line from the second Output VI as shown below.

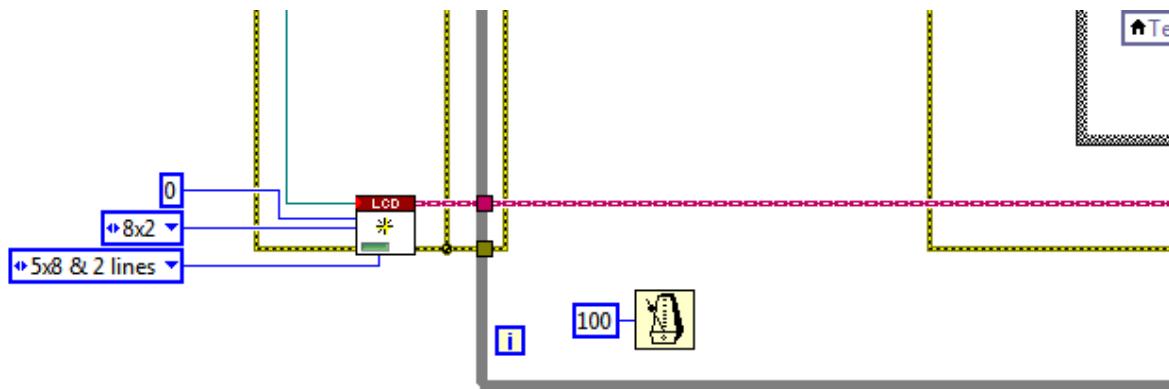


15. Insert an **Or** logic function (Functions»Programming»Boolean) to the right of the **Stop** shared variable. Wire the Error lines and output of the **Stop** shared variable as highlighted in red below, similar to what you did in the FPGA Variable Output loop.



16. Without setting timing in the while loop it will execute as fast as it can, potentially starving resources allocated to the other loop. To avoid this insert a **Wait Until Next ms Multiple** (Functions»Programming»Timing) function. Right-click on its **millisecond multiple** input and select **Create»Constant**.

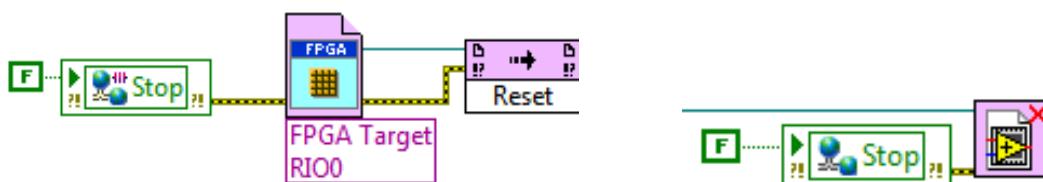
Enter a value of **100** in the constant to execute the loop every 100 milliseconds.



17. Inside the project under the *NI-RIO Evaluation HW* target, locate the **Stop** shared variable inside the **RT-Host Shared Variables.lvlib**. Drag an instance of this variable into the block diagram to the left of the Open FPGA VI Reference VI and one to the left of the Close FPGA VI Reference VI.
 18. Change the **Stop** variables to write mode by selecting them, right-clicking, and going to **Access Mode»Write**.

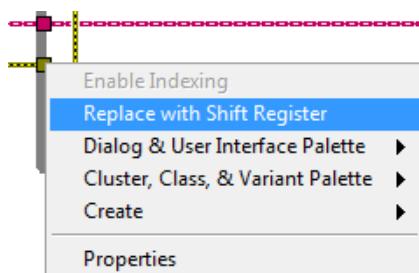


19. Right-click on the **Stop** input terminal of both shared variables and select **Create»Constant**. Leave the constant as the default False value.
 20. Wire the error line from the **error out** terminal of both Stop shared variables to the **error in** terminal of the Open FPGA VI Reference VI and Close FPGA VI Reference VI to enforce dataflow and ensure that initializing the Stop shared variable to False executes before those VIs every time you run the program.



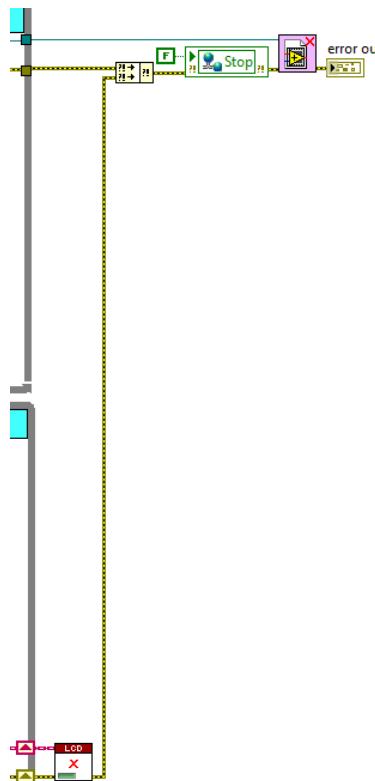
21. To finish the error propagation to the right of both while loops, wire the error lines as shown below into a **Merge Error VI** (Functions»Programming»Dialog and User Interface) before wiring to the **error in** terminal on the Stop shared variable.

22. Right-click on the left-hand error tunnel on the while loop and select **Replace with Shift Register** from the drop down menu. LabVIEW replaces the tunnel you right-clicked with a shift register terminal, and the cursor becomes a shift register icon (▣). Hover over the error tunnel on the opposite side of the loop until it flashes, then click the tunnel to replace it with a shift register. A shift register enables the passing of data from one loop iteration to the next.



23. Repeat this step to turn the LCD Driver Reference loop tunnels to shift registers.

24. Right-click on the **error out** terminal of the Close FPGA VI Reference VI and select **Create»Indicator** to display any errors to the user on the front panel.



25. Save the Real-Time VI by selecting **File»Save** or pressing **CTRL+S**.

(Optional) Test the Real-Time and FPGA Applications

At this point the FPGA VI should have completed its compilation process and now you can run the following tests on the FPGA and Real-Time target applications.

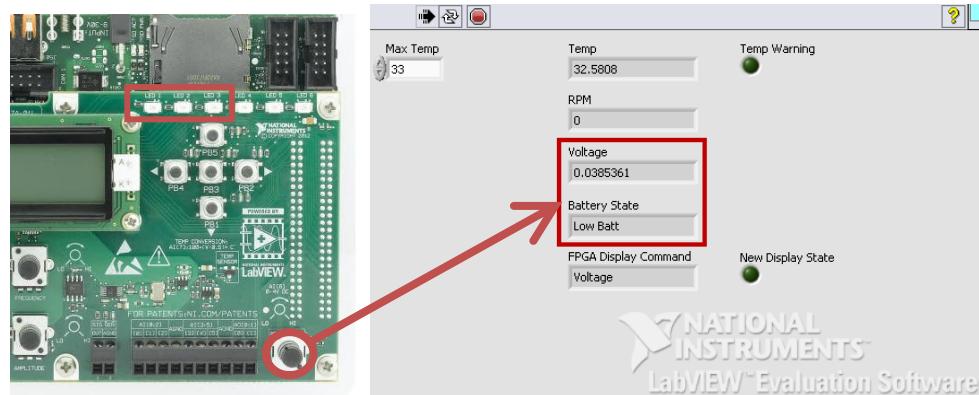
(Optional) FPGA Application Test

1. Bring up the Exercise 2-FPGA project that should still be open in the background and double click on the FPGA.vi if it is not already open.

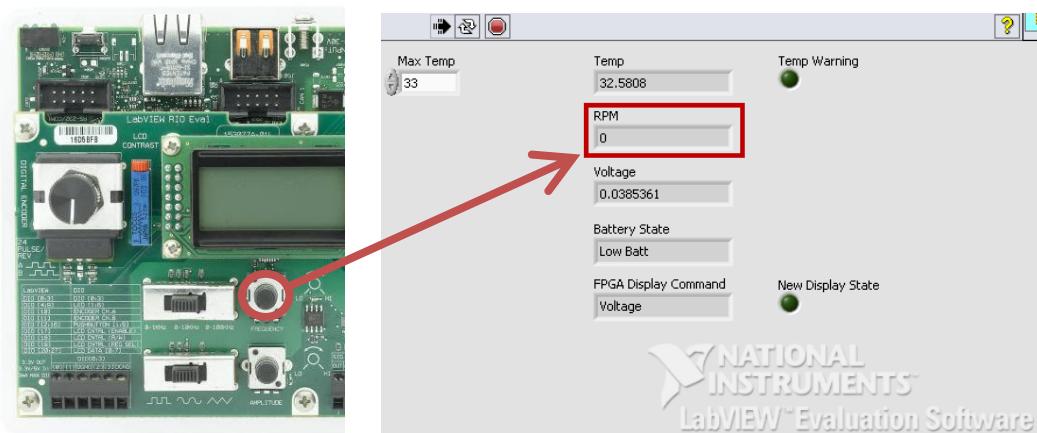
Note: If you did not complete the Exercise 2 FPGA application, reference the solution at .\2-Create FPGA Application_Solution and open the Exercise 2-FPGA.lvproj. The solution VI will need to be compiled for your specific target. Reference the *Using the Solutions* section on **Page 9** for more details on using the solution.

2. Click the run button if it is not already running and complete the following tests:

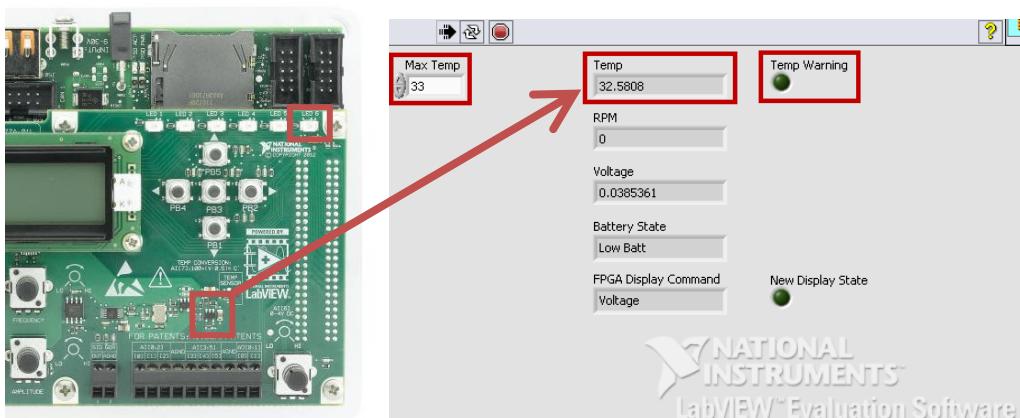
- ✓ Turn the potentiometer in the lower right-hand corner of the board and verify that on the FPGA front panel that the battery cell voltage indicator, called **Voltage**, increases and the **Battery State** indicator also changes. Also note the changes in the board LEDs representing the different digital signals that would be sent to the power distribution circuit based on the battery state (Low Batt, Running, and Charged).



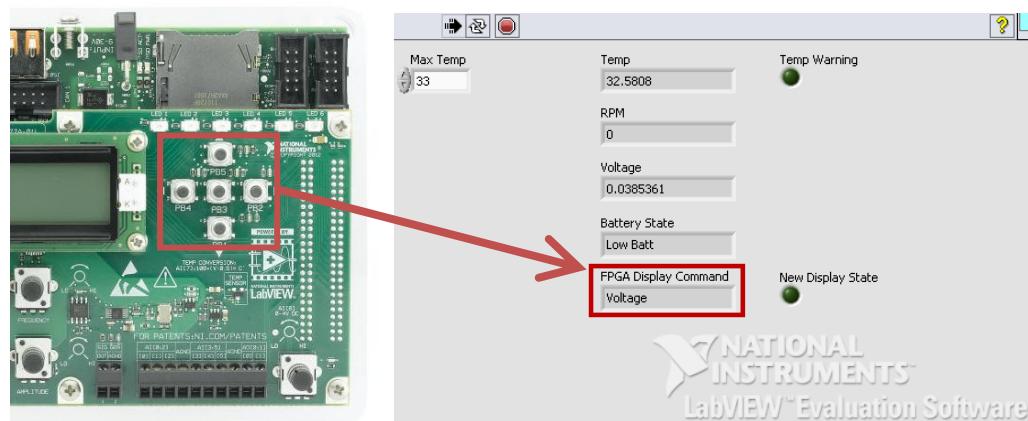
- ✓ Turn the potentiometer pictured below to vary the Motor PWM signal frequency and verify that the **RPM** signal responds accordingly on the FPGA front panel.



- ✓ Put your finger on the temp sensor shown below and verify that the **Temp** signal responds accordingly on the FPGA front panel. Reduce or increase the Max Temp so that you can test the Temp Warning and ensure that if the temp is above the maximum that the Temp Warning front panel alarm LED is on. Also verify that the on board LED turns on providing notice to the operator of an alarm.



- ✓ The final FPGA test is to verify the push button inputs. Assert each of the four outer push buttons (PB1, PB2, PB4, and PB5) and verify that the FPGA Display Commands update with a new menu item.



Note: Since the real-time application creates and controls the LCD screen, it will not yet work. The next testing section for the real-time application will exercise this functionality.

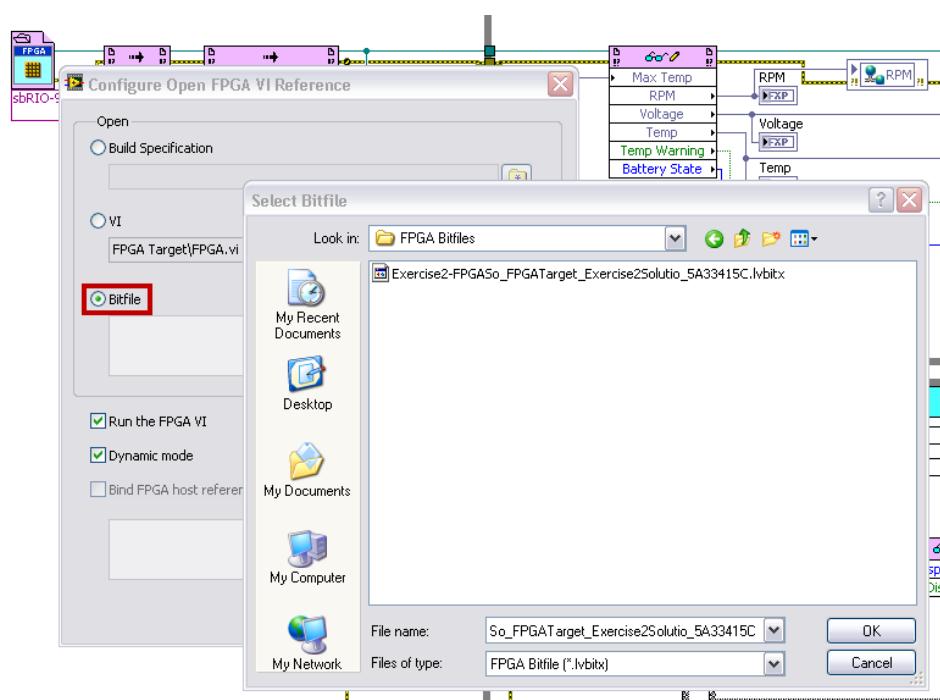
3. When you are finished testing the FPGA VI, click the **Abort** button .
4. Right-click on the *NI-RIO Evaluation HWtarget* in the LabVIEW project and select **Disconnect**. Note the bright green Boolean turns off, indicating the target is disconnected. Finally, close out all of the Exercise 2 LabVIEW files and save the files if prompted.

Congratulations, you have completed your FPGA application testing! As you can see, the FPGA VI front panel is very helpful for debugging purposes.

(Optional) Real-Time Application Test

Initial Setup

1. Transition back to the Exercise 3-RT project to test the real-time application. Open up the **RT Microprocessor** VI if it is not already in memory.
2. Since the FPGA VI was compiled in a different project, the static FPGA compiled application, or bitfile, needs to be referenced in the current project. Toggle to the block diagram (**CTRL+E**), right-click on the **FPGA Open VI Reference** VI and select **Configure FPGA Open VI Reference...**
3. In the dialog that appears click the radio button next to **Bitfile** and browse to **.\\2- Create FPGA Application\FPGA Bitfiles** and select the file inside. Click **OK** on the dialog to exit.



Note: This is the bitfile that is a result of your Exercise 2 LabVIEW FPGA code compilation.

4. To finish the FPGA setup, right-click on the **resource name** input terminal on the **FPGA Open VI Reference** and select **Create»Constant**. From the constant drop down select **RIO0**. This is the name of your FPGA target resource from the LabVIEW Project.

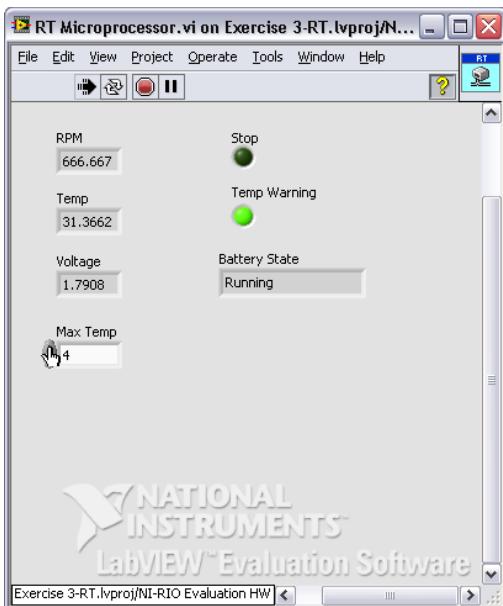


Application Testing

- Click the **Run** button to compile and deploy the real-time application to the NI-RIO Evaluation Device, which also includes the FPGA application since the VI references the FPGA bitfile you just selected. Save the VI if prompted.

Note: If a **Conflict Resolution** dialog box appears on deployment click **Apply** to overwrite the current real-time application that is still running on the device.

- For the real-time application testing, run the same tests as you did for the FPGA except now look to the Real-Time application front panel for verification of values.
- Also note that the LCD screen now displays the different menus that the real-time application generates and transfers to the FPGA.



- When you are finished testing the real-time VI, click the **Abort** button.
- Right-click on the *NI-RIO Evaluation HW* target in the LabVIEW project and select **Disconnect**.
- Save all of the Exercise 3 files by selecting **File»Save All** in the Project Explorer window.

You have now created a real-time application running on a microprocessor which acquires data from the FPGA, controls the LCD screen, and coordinates network communication with the Windows user interface!

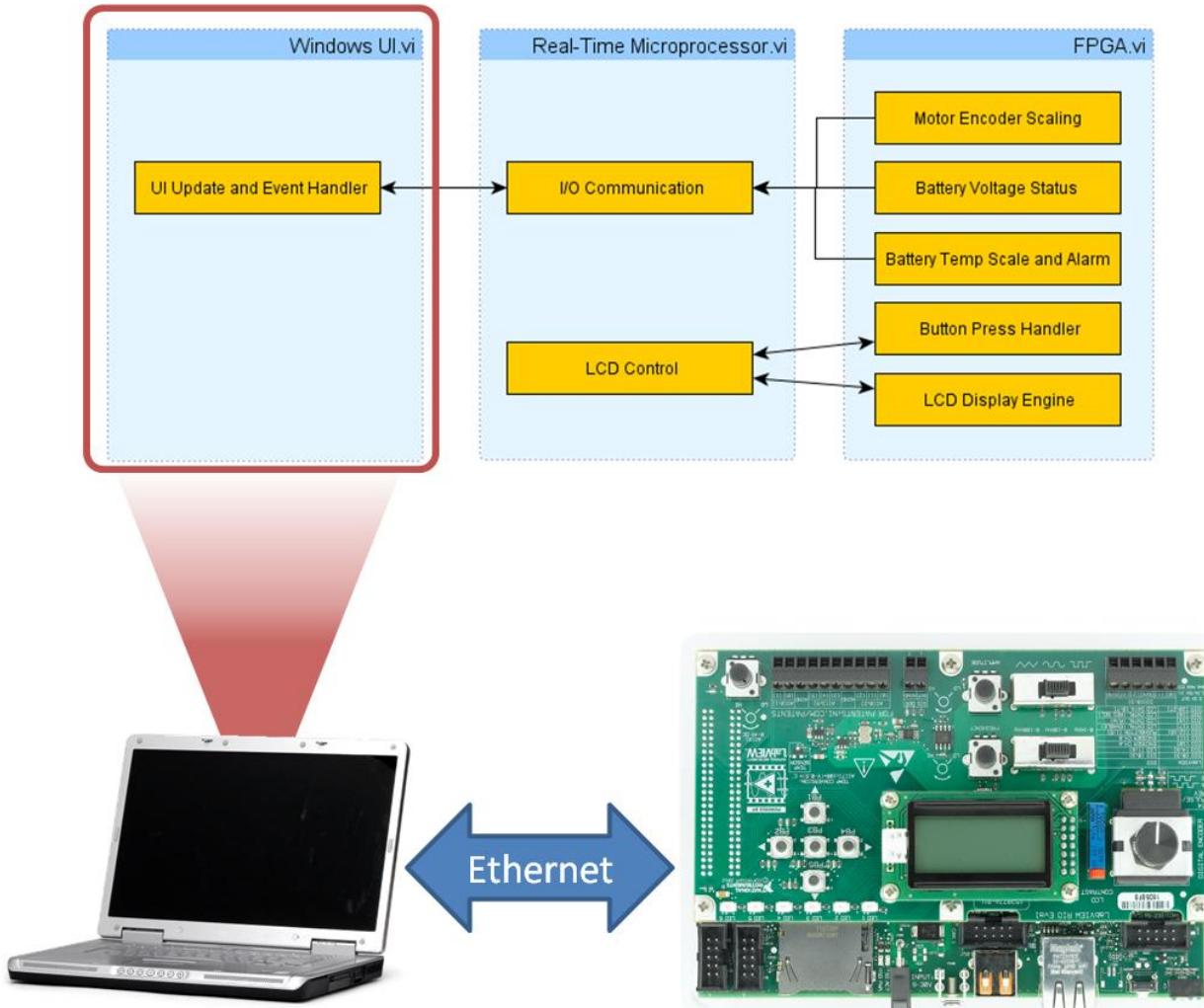
Exercise 4 | Create Windows User Interface

Summary

In this exercise you will complete your battery monitoring application using the FPGA VI that you created in Exercise 2 and the Real-Time VI you created in Exercise 3. You will modify the Windows User Interface VI to read data from the network published shared variables that you wrote data to in the Real-Time application in Exercise 3. Then you will finish by wiring this data to the User Interface components to display the current battery management system data. In this exercise, you will complete the following tasks:

1. Explore the Windows-Based Application User Interface
2. Finish Development of the Windows-Based Application User Interface
3. Run and Verify the Completed System

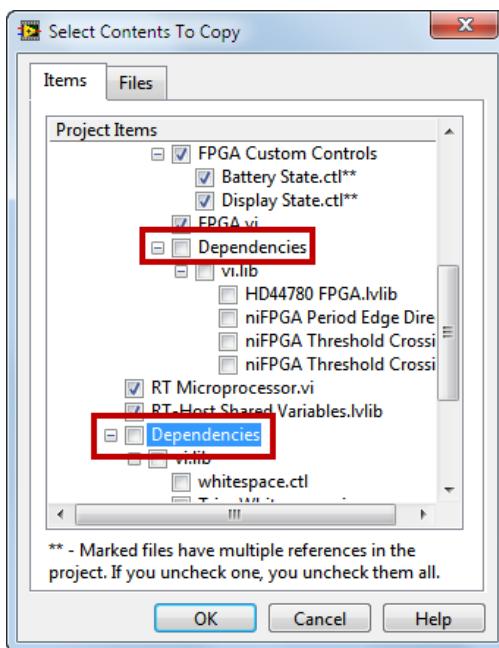
What am I going to accomplish in this exercise?



In this exercise you will finish the overall embedded system by completing a user interface running on your Windows development machine. The user interface components have already been placed for you, but the backend communication architecture needs to be completed so that you can update the user interface with current values and the state of the battery and motor.

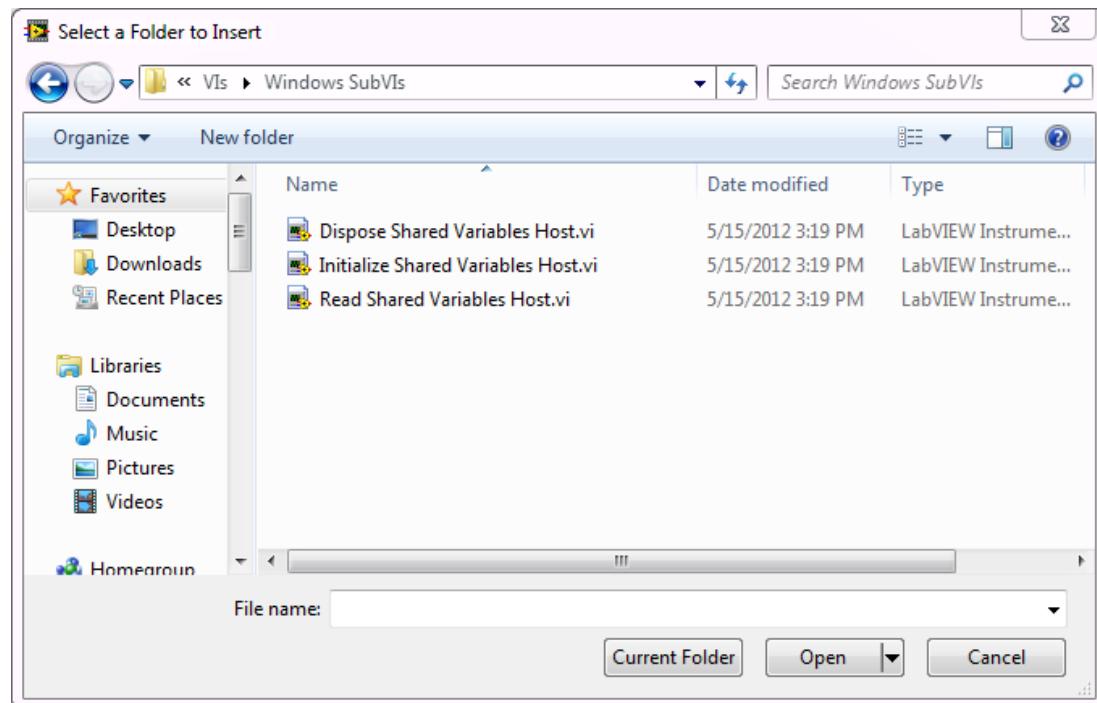
Project Setup

- With your Exercise 3 project still open in the Project Explorer Window, select **File»Save As...** to copy the project files into the Exercise 4 folder.
- In the Save As... dialog box that appears select **Duplicate .lvproj file and contents » Select contents to copy** and click **Continue...**
- Uncheck the *Dependencies* category for the Real-time and FPGA target hierarchies and select **OK**. These LabVIEW dependency files are not necessary since you will still be saving the files on the development machine.



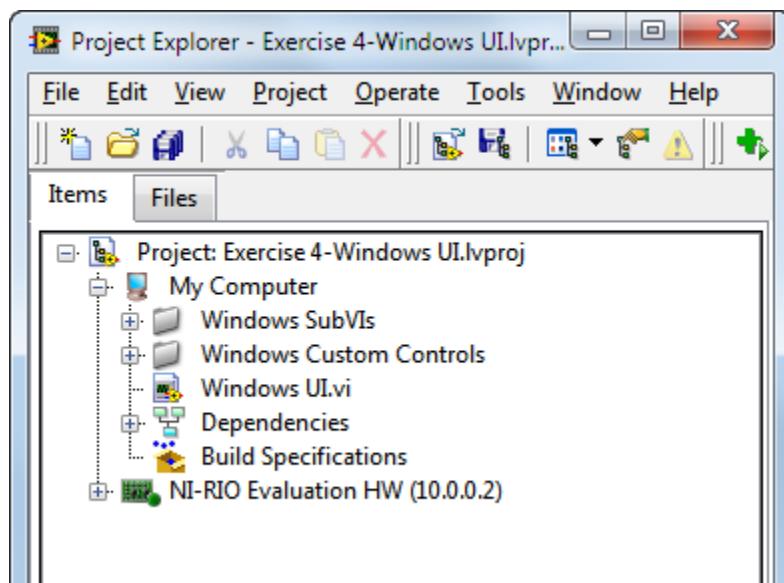
- Navigate to the ...\\4- Create Windows UI\\ folder and rename your project as *Exercise 4-Windows UI.lvproj*.
- Click **OK** to save the project and supporting files. If a Load Warning occurs click **Ignore** as it is confirming the new file locations.
- Close the Exercise 3 project and open the *Exercise 4-Windows UI.lvproj* from .\\4- Create Windows UI\\ that was just created. If prompted save the Exercise 3 project.
- To add the skeleton Windows UI application already created, in the Project Explorer window right-click on *My Computer* and select **Add»File...** Navigate to the .\\4- Create Windows UI\\Vis folder and select **Windows UI.vi**.

8. Right-click on *My Computer* and select Add»Folder (Snapshot)... Navigate to the .\4-Create Windows UI\VI\Windows SubVIs folder and select **Current Folder**.



9. Right-click on *My Computer* and select Add»Folder (Snapshot)... Navigate to the .\4-Create Windows UI\Custom Controls folder and select **Current Folder**. Select the **Custom Controls** folder that was just added to the project, press **F2**, and rename it **Windows Custom Controls**.

The *My Computer* section of your project should now look as follows:

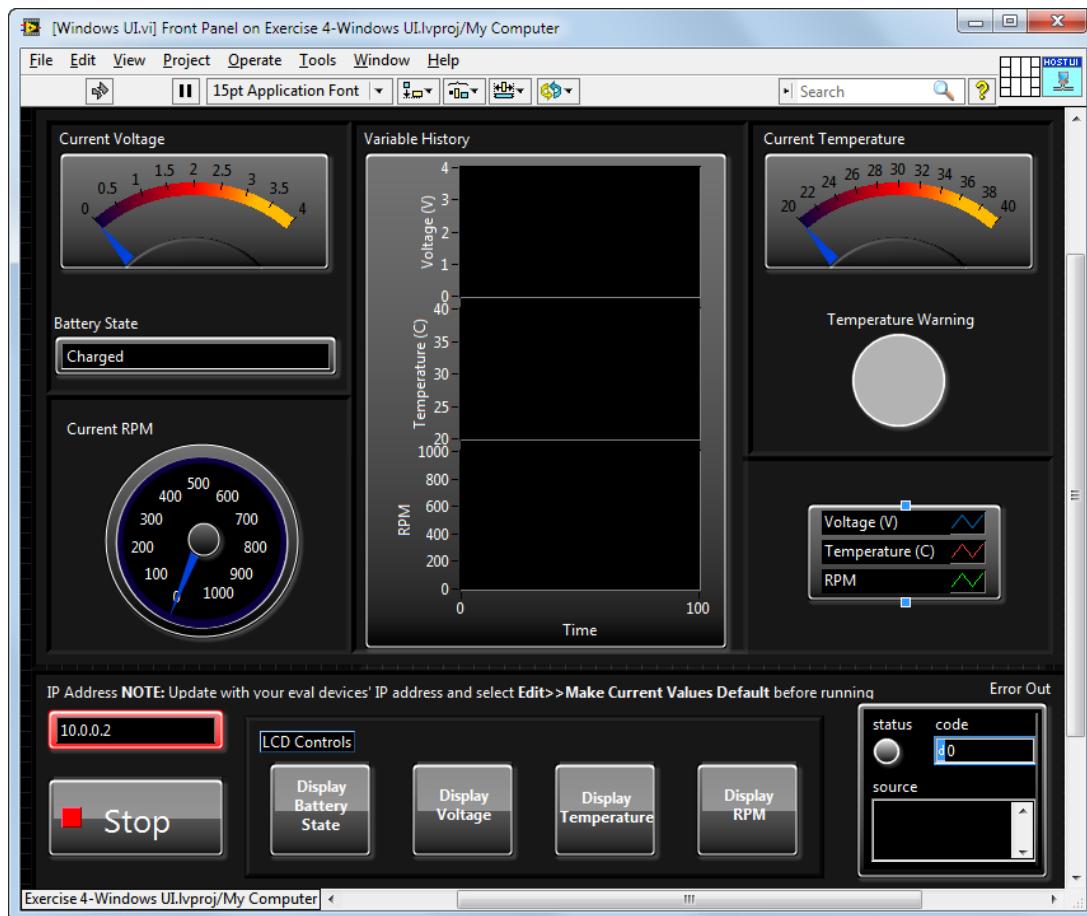


Explore the Windows-Based Application User Interface

In this section you will explore the Windows PC User Interface application and its interaction with real-time processor application.

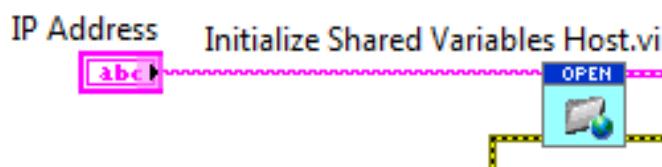
1. In the Exercise 4- Windows UI Project Explorer window, double click to open **Windows UI.vi** to view the User Interface. Note there are indicators for each of the values you've been monitoring on the Real-Time front panel thus far:

- ✓ Current Voltage, Temperature, and RPM, as well a variable history of these three values.
- ✓ Temperature Warning alarm Boolean indicator
- ✓ Along the bottom are LCD indicators to display the current LCD screen state

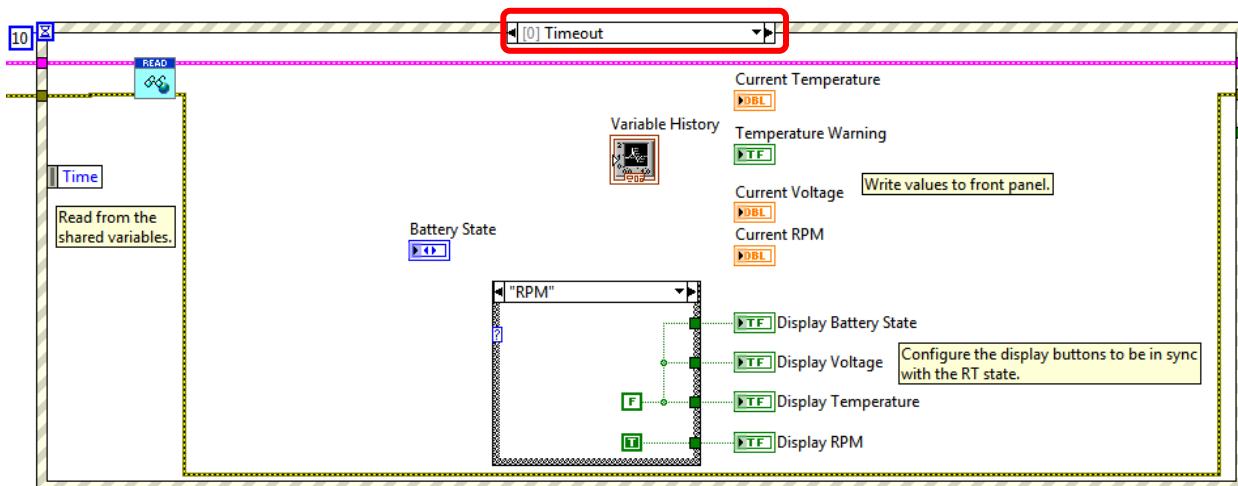


2. On the user interface, update the **IP Address** string control value to the IP address of your NI-RIO Evaluation Device.
3. To set this IP address as the default on start up, select **Edit»Make Current Values Default** after entering the IP address.

4. Save the VI by going to **File»Save** or pressing **CTRL+S**.
5. Press **CTRL+E** to view the block diagram logic. To the left of the while loop the IP address of the NI-RIO Evaluation Device is collected with the **IP Address** string control on the front panel and a connection to its shared variable engine, hosted on the microprocessor, is established with the **Initialize Shared Variables Host.vi**. This VI takes an IP address and builds references to all the shared variables used on the host, in this case, your NI-RIO Evaluation Device. Those references are bundled in a cluster to easily transmit the data in one wire.

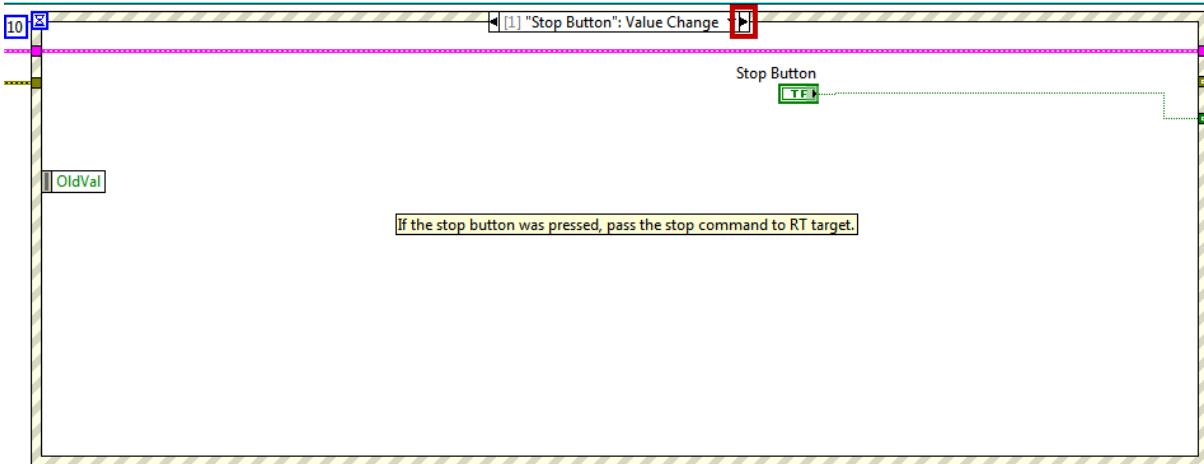


6. Inside the **UI Update and Event Handler** while loop, observe the Timeout case of the event structure. The LabVIEW Event Structure executes when a defined event is detected, in this case, if no event is detected within 10 ms, the timeout case is executed. This timeout case:
 - ✓ Reads the network published shared variables that were written to in the real-time application with the **Read Shared Variables Host.vi**.
 - ✓ Contains all of the mapping to connect the user interface components to data from the shared variables so that those components update with the most recent values. You will complete this wiring in the next section.



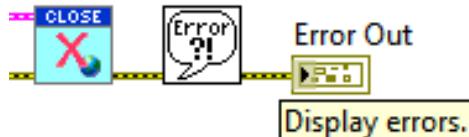
7. Now toggle the event structure to the “Stop Button”: Value Change case by clicking the arrow next to the Timeout case text.

8. Note that the user interface **Stop Button** Boolean control is located inside this event case, and the case will execute when the user interface detects a value change (push) of the Stop Button. In the next section you will add the **Stop** network published shared variable to communicate the stop condition to the real-time processor target.



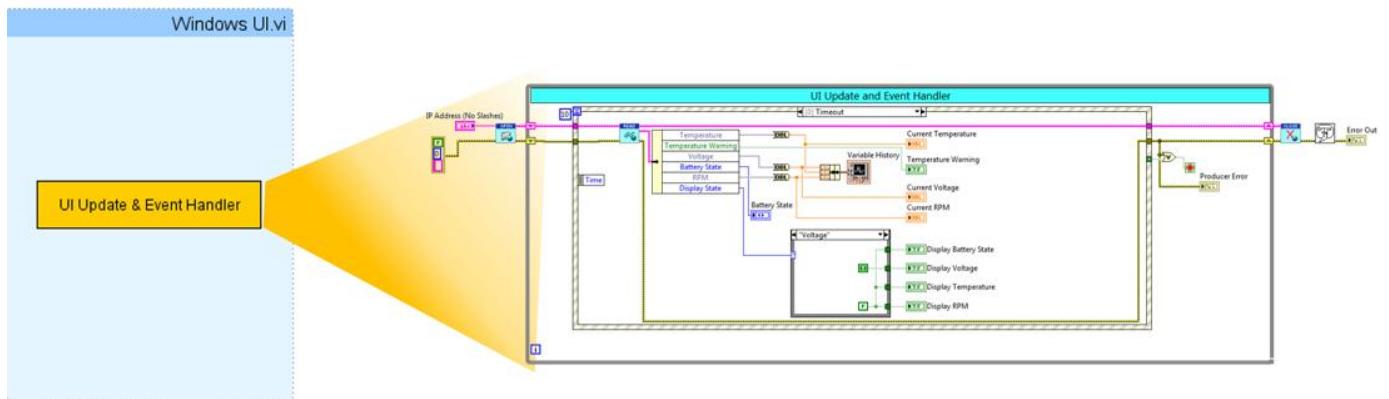
9. Finally, to the right of the while loop the shared variable connections are closed before errors are checked the application exits.

Close connection to shared variables.

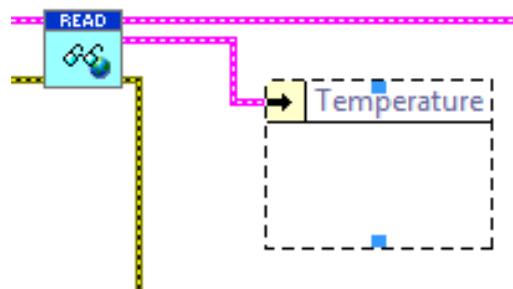


Finish Development of the Windows-Based Application User Interface

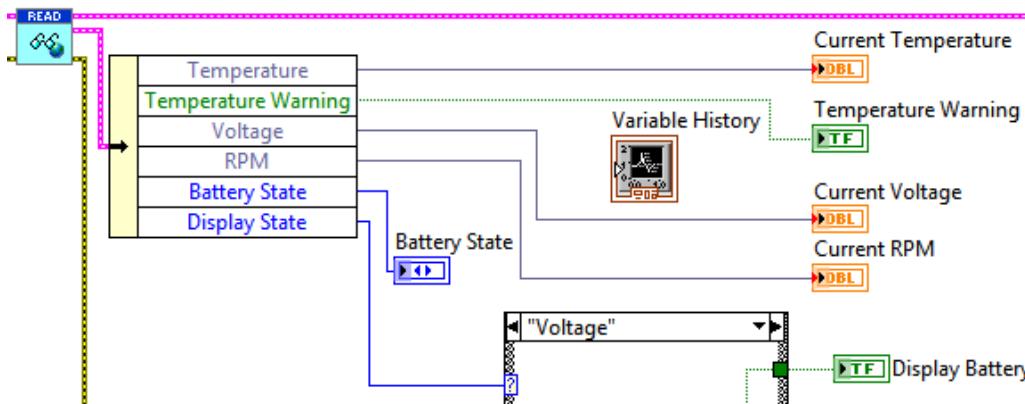
In the last section you will complete the Windows User Interface application by connecting the block diagram logic to front panel controls/indicators and insert the Stop shared variable for communication of the stop condition to the NI-RIO Evaluation Device.



1. In the **UI Update and Event Handler** loop the shared variables that are read need to be unbundled for connection to their respective front panel indicators.
 - ✓ In the Timeout event case, insert an **Unbundle By Name** function (Functions»Programming»Cluster, Class, & Variant) to the right of the **Read Shared Variables Host VI**.
 - ✓ Wire the **Current Values** output of the **Read Shared Variables Host VI** to the input of the **Unbundle By Name** function. This unbundles each of the six shared variable values that were contained in the cluster.
 - ✓ Expand the values to unbundle by left-clicking to select the **Unbundle By Name** function and drag out the blue lower boundary until six terminals are exposed.

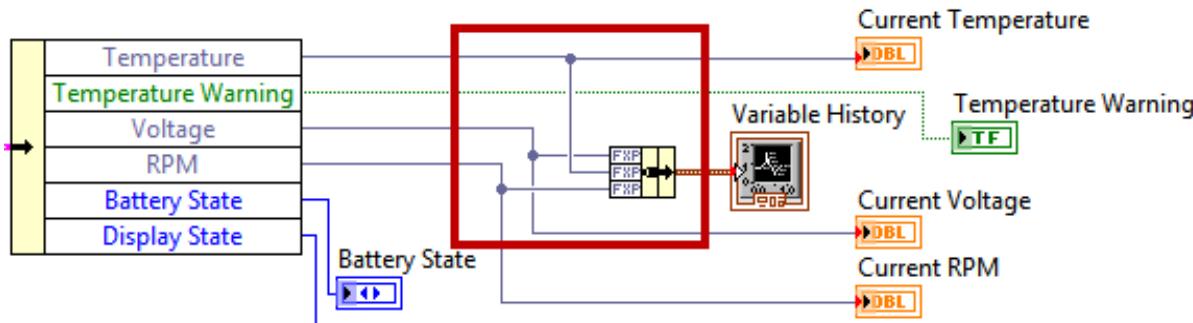


- ✓ Left-click on each terminal and select the shared variable name that matches the order in the screenshot below.
- ✓ Wire the output terminals to the matching front panel indicators.



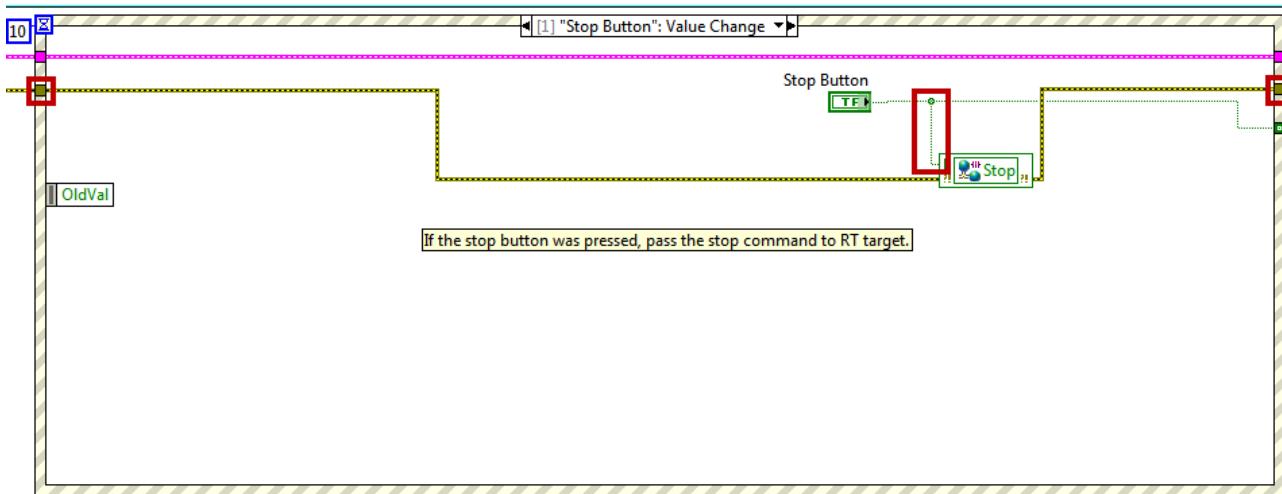
2. Add a **Bundle** function and wire it up as shown to group Voltage, RPM, and Temperature values for input into the Variable History chart.

- ✓ Insert the **Bundle** function (Functions»Programming»Cluster, Class, & Variant)
- ✓ Expand the terminals to three by selecting the Bundle and extending the lower border.
- ✓ Create a branch from the existing wires and connect **Voltage** to the top terminal, **Temperature** to the middle terminal, and **RPM** to the lower terminal.
- ✓ Wire the output of the **Bundle** function into the input of the **Variable History** Waveform Chart indicator.



3. Switch the event structure to the “Stop Button”: Value Change case by clicking the arrow next to the Timeout case text.
4. Inside the project under the *NI-RIO Evaluation HW* target, locate the **Stop** shared variable inside the **RT-Host Shared Variables.lvlib**. Drag this variable into the block diagram and insert it into the current event case to communicate the stop condition to the NI-RIO Evaluation Device.

5. Change the **Stop** shared variable to write by selecting it, right-clicking, and going to **Access Mode»Write**.
6. Branch the wire from the **Stop Button** terminal and connect it to the **Stop** input of the **Stop** shared variable. Wire the error line from the left tunnel to the shared variable **error** in terminal and wire the **error out** to the right-hand tunnel.



7. Save the Windows UI VI by going to **File»Save** or pressing **CTRL+S**.

Run and Verify the Completed System

Now that you have completed the development of your system, run through these steps to verify its behavior.

1. Open and run the **RT Microprocessor VI** from the *Exercise 4- Windows UI* project. If you have a broken Run arrow refer to the note below.

Note: If you did not complete the optional real-time application testing at the end of Exercise 3, go to **page 60** and complete the **Initial Setup** section of the real-time application to link to the FPGA application bitfile.

Note: If a **Conflict Resolution** dialog box appears on deployment click **Apply** to overwrite the current real-time application that is still running on the device.

2. Open and run the **Windows UI VI** from the *Exercise 4- Windows UI* project.

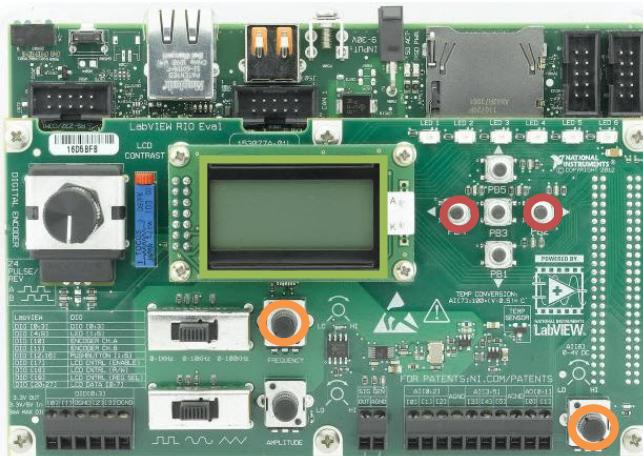
Note: If the Windows application throws an error or does not respond, verify that the correct IP address for your NI-RIO Evaluation Device is entered on the user interface. If it is not, re-enter it, select **Edit»Make Current Values Default**, and click the Run button to restart the Windows application.

3. On the daughter card push **PB4** to display the voltage on the LCD screen and turn the potentiometer.

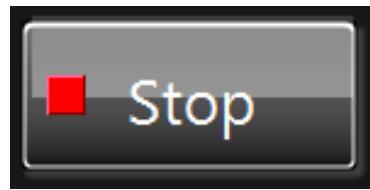
- ✓ Verify that the battery voltage changes on the Windows User Interface and on the LCD screen accordingly.
- ✓ Note the battery state as you change it from Low Batt, to Running, to Charged and the corresponding LEDs that turn on like a gauge.

4. On the daughter card push **PB2** to display the motor RPM on the LCD screen.

- ✓ Turn the frequency potentiometer for the function generator and verify that the RPM value changes on the Windows User Interface and on the LCD screen accordingly.



5. Once you are done testing, click Stop on the Windows UI to stop both applications.



6. When you are finished testing the system, right-click on the *NI-RIO Evaluation HW* target in the LabVIEW project and select **Disconnect**. Note the bright green Boolean turns off, indicating the target is disconnected.

Congratulations, you have finished the development of your LabVIEW RIO-based embedded system!

Next Step

Deploy and Replicate Your Embedded System

Exercise 5 | Application Deployment and Replication

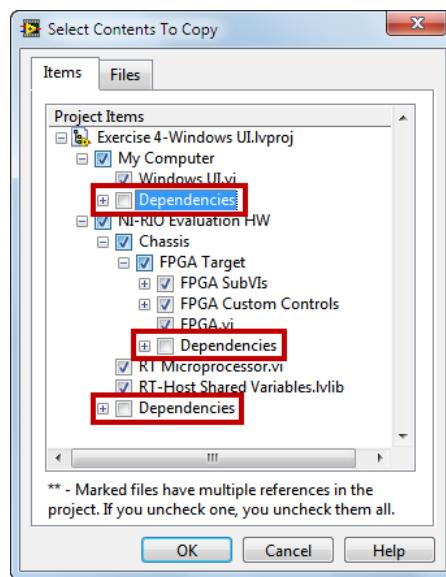
Summary

Now that you have completed the development of your embedded system, you need to deploy and replicate the application. In this exercise you will complete these two tasks:

1. Create and Deploy a Startup Real-Time Executable
2. Save a System Software Image and Deploy to Formatted Hardware

Project Setup

1. With your Exercise 4 project still open in the Project Explorer Window, select **File»Save As...** to copy the project files into the Exercise 5 folder.
2. In the *Save As...* dialog box that appears select **Duplicate .lvproj file and contents » Select contents to copy** and click **Continue...**
3. Uncheck the *Dependencies* category for each of the My Computer, Real-Time Processor, and FPGA target hierarchies and select **OK**. These LabVIEW dependency files are not necessary since you will still be saving the files on the development machine.

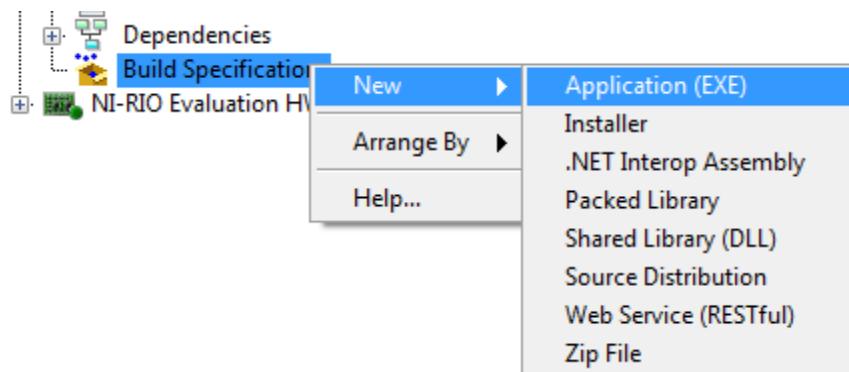


4. Navigate to the ...\\5- DeployReplicate System folder and rename your project as *Exercise 5- Deploy Replicate.lvproj*. Click **OK** to save the project and supporting files.
5. Close the Exercise 4 project and open the *Exercise 5- Deploy Replicate.lvproj* from ...\\5- DeployReplicate System\\ that was just created. Save Exercise 4 files if prompted.

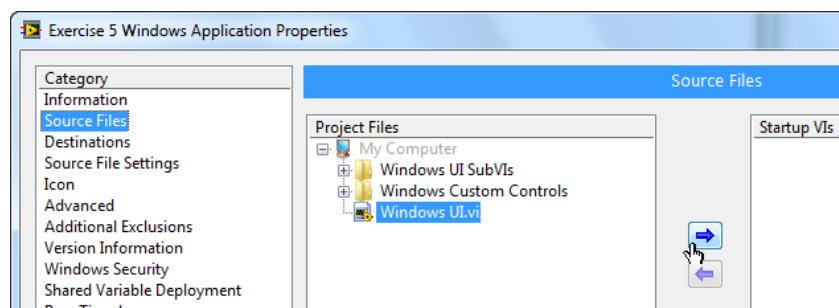
Create and Deploy a Startup Real-Time Executable

Create an executable for your Windows User Interface application

1. In the LabVIEW Project Explorer window expand out the My Computer target, right-click on the Build Specification, and select **New»Application (EXE)**.



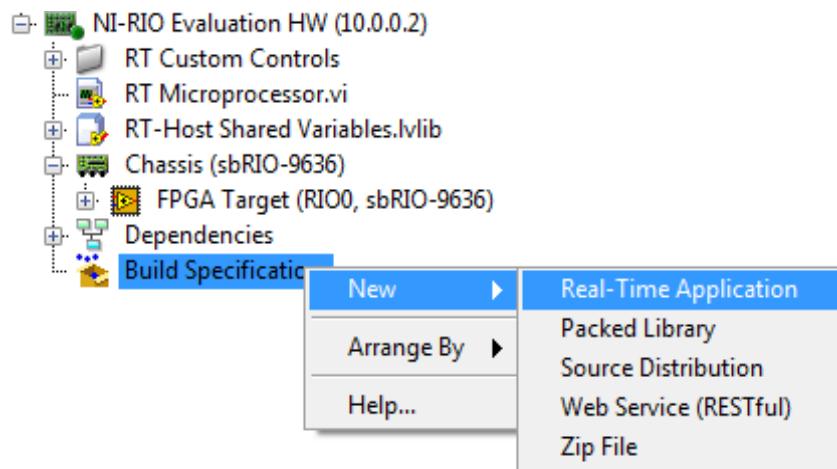
2. An *Application Builder Information* dialog window may open upon building an application for the first time. Check the **Do not prompt again for this operation** box and click **OK**.
3. Once the *My Application Properties* dialog window appears, enter the **Build Specification name** and **Target filename** as *Exercise 5 Windows Application* and the Destination directory as *.\5- DeployReplicate System\builds\ Windows Application*.
4. In the *Category* list on the left-hand side of the window, click on **Source Files** and select **Windows UI.vi** from the Project Files section. Add the VI as the Startup VI by clicking the right arrow. This designates the VI to build into an executable and its User Interface that will appear when running the executable.



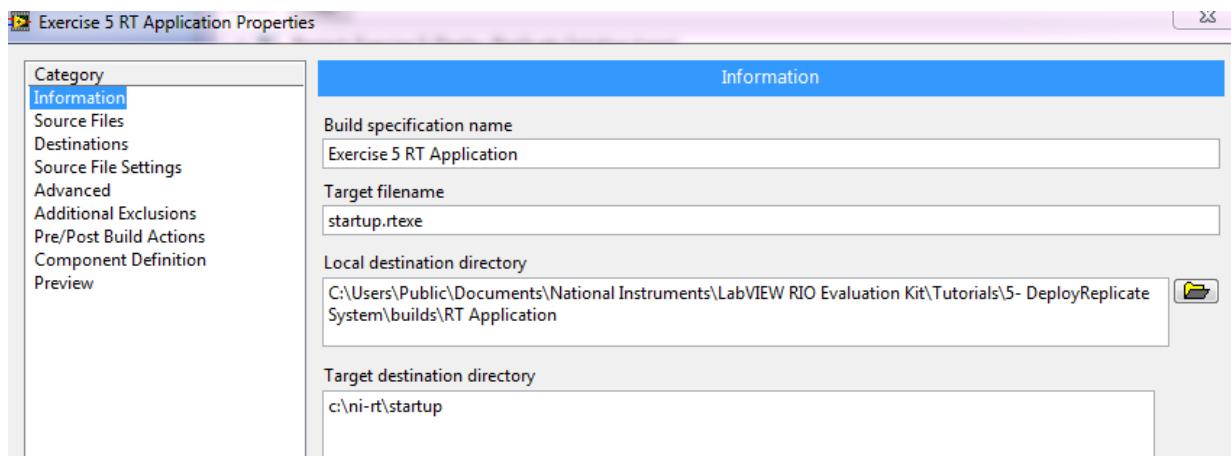
5. Leave the remainder of the categories as defaults and click **Build**. Once the build is successful click **Done**.

Create an executable for your real-time application

- In the LabVIEW Project Explorer window expand out the *NI-RIO Evaluation HW* target, right-click on the bottom Build Specification, and select **New»Real-Time Application**.



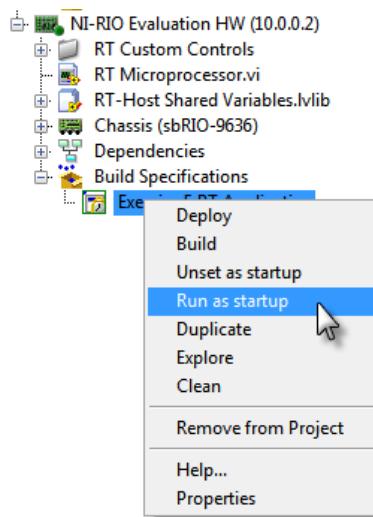
- In the dialog window, enter the Build specification name as *Exercise 5 RT Application*.
- Set the Local destination directory as *.\5- DeployReplicate System\builds\ RT Application*.



- In the *Category* list on the left-hand side of the window, click on **Source Files** and select **RT Microprocessor.vi** from the Project Files section and add the VI as the Startup VI by clicking the right arrow.
- Leave the remainder of the categories as defaults and click **Build**. Once the build is successful click **Done**.

Set the real-time executable as a startup application so you do not need to manually deploy it to the target every time you reboot the system.

11. In the LabVIEW Project Explorer window, right-click on the *Exercise 5 RT Application* build specification that you created and select **Run as startup**.



Note: If a **Conflict Resolution** dialog box appears on deployment click **Apply** to overwrite the current real-time application that is still running on the device.

12. LabVIEW will deploy the application to the target hard drive and then will prompt you to reboot the target to start executing the startup application. Click **Yes** to continue with the reboot.

Note: Allow for a few minutes for the Real-Time Operating System (RTOS) to boot up after reboot. Once it does successfully reboot the LCD screen will start updating.

13. Navigate to and run on the Windows executable that you built, located at .\5-DeployReplicate System\builds\Windows Application\Exercise 5 Windows Application.exe

Verify that the initial deployment of the system works by running through the system tests from the *Run and Verify the Completed System* section of Exercise 4 on [page 72](#). It should now execute, communicate with the headless real-time application, and update the LCD screen.

14. Close all LabVIEW files and save if prompted.

Save a System Software Image and Deploy to Formatted Hardware

Warning

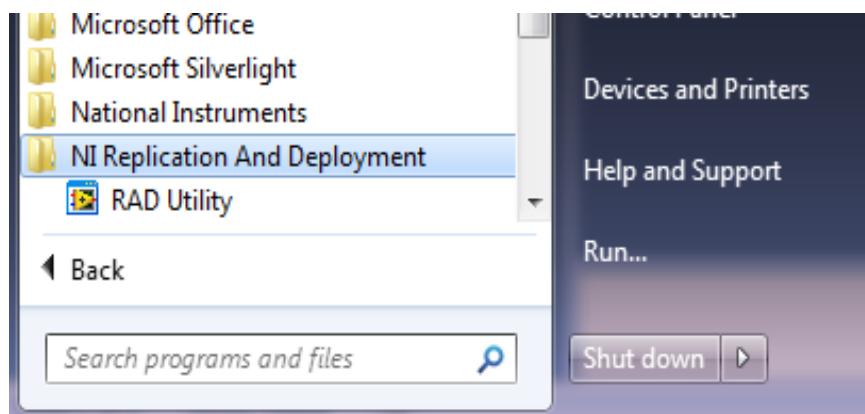
This section re-formats the hard drive on your NI-RIO Evaluation Device. If for some unexpected reason after 5 minutes or more the device does not respond, you can reset the device to a known state by re-running the LabVIEW RIO Evaluation Setup Wizard (Windows Start Menu»All Programs»National Instruments»LabVIEW RIO Evaluation Kit»Setup Utility).

1. National Instruments provides a utility to rapidly create and deploy hardware images of your application. The utility is called the **Replication and Deployment (RAD) Utility** and can be downloaded and installed from the following location on ni.com:

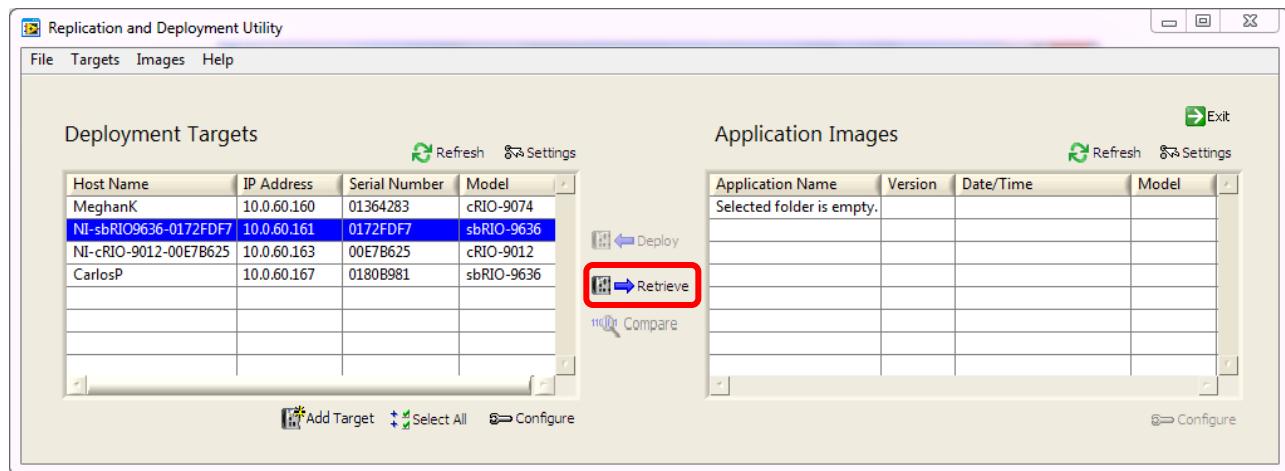
[Replication and Deployment \(RAD\) Utility](#)

and download **rad_3002_installer.zip**

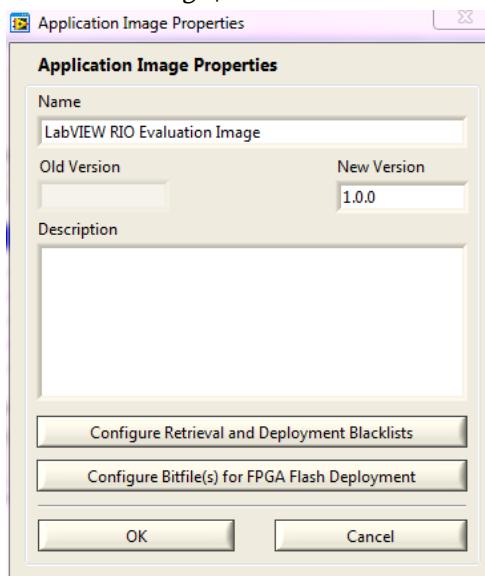
2. Once you have downloaded the RAD Utility from the above location run the installer.
3. Navigate to and open the **NI Replication and Deployment (RAD) Utility** from Windows Start Menu»All Programs»NI Replication And Deployment»RAD Utility.



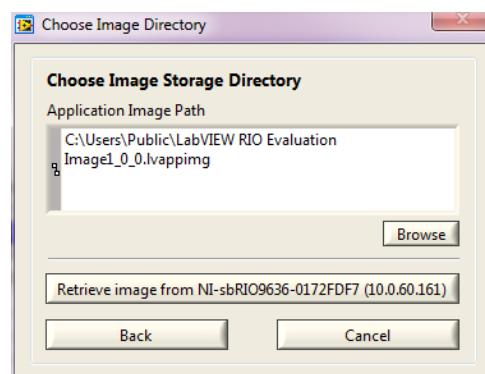
4. When the RAD utility opens, it will scan for systems on the network and will populate your target in the left-hand table. If you are connected to a router and there are other targets on the network, they will appear as well. Select the *NI-sbRIO9636-<Your Device serial number and IP address>* target and click **Retrieve** to retrieve an image.



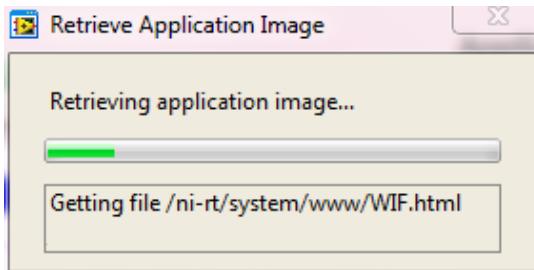
5. A *Select Image Type* dialog box will appear. Select **New Application Image**, enter the name *LabVIEW RIO Evaluation Image*, and click **OK**.



6. Choose the image storage directory by browsing to the folder **\5- DeployReplicate System\images**. Click **Retrieve Image from NI-sbRIO-<Your Device serial number and IP address>** to start saving the target's image.



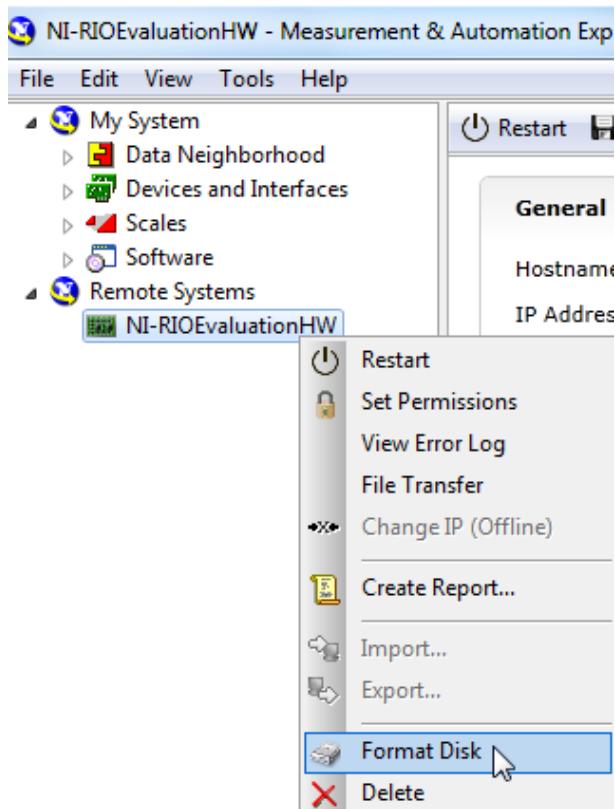
7. The retrieval and image processing will take about 5 minutes. When you receive the dialog box that the application image retrieval was successful, click **OK**.



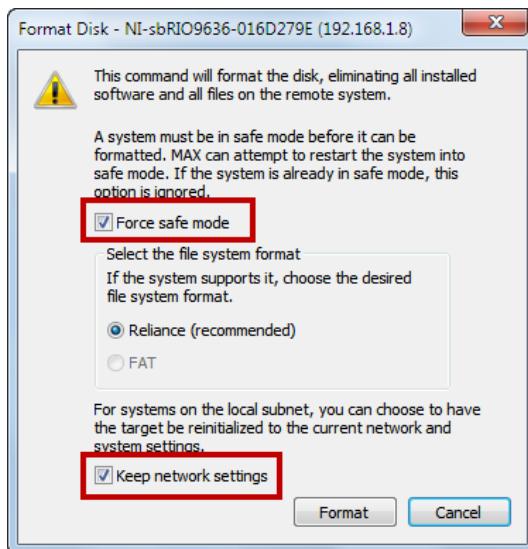
8. Once the image has been successfully copied to your development computer, you will now format the NI-RIO Evaluation Device to simulate how you would go about deploying this image to a new system.

To format the target, launch the **Measurement & Automation Explorer** (NI MAX), located on your Windows Desktop or at (Start»All Programs»National Instruments»NI MAX).

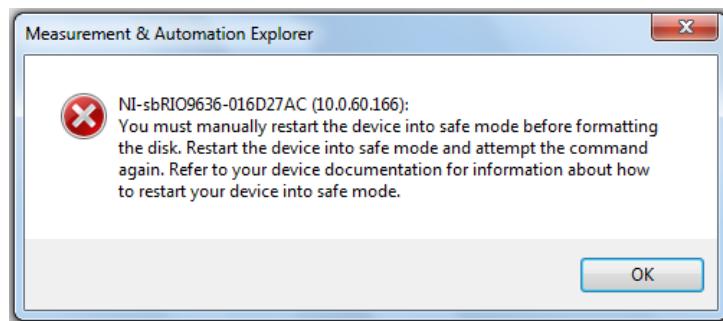
9. Expand **Remote Systems** in the left-hand navigation window to expose your target. Right-click on your target (it may be named *NI-sbRIO-<Your Device serial number and IP address>*) and select **Format Disk**.



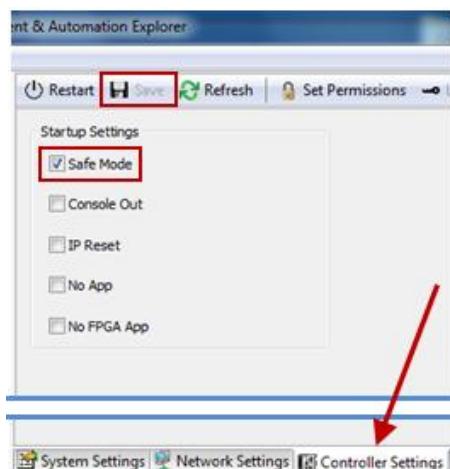
10. Check the boxes to **Force safe mode** and to **Keep network settings**, then click **Format**.



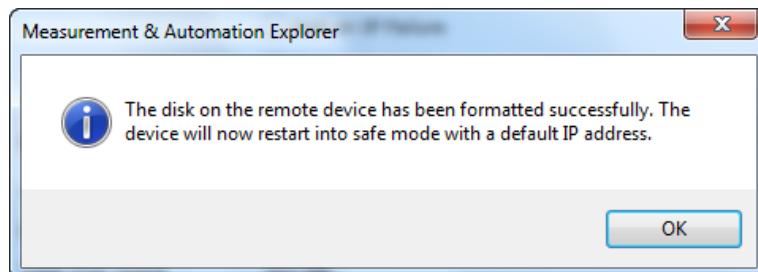
Note: If after clicking **Format** Measurement & Automation Explorer displays the following warning dialog:



Then navigate to the **Controller Settings** tab for the NI-RIO Evaluation Device (shown below), check the box next to **Safe Mode** to force the device into safe mode and click **Save**. When prompted to restart, click **Yes** and wait for the device to reboot before retrying steps 8-10.



11. Once you have successfully formatted the device the following dialog window will appear.



Note: The reboot will take about a minute to complete.

12. Click **OK** and then go back into the RAD Utility.

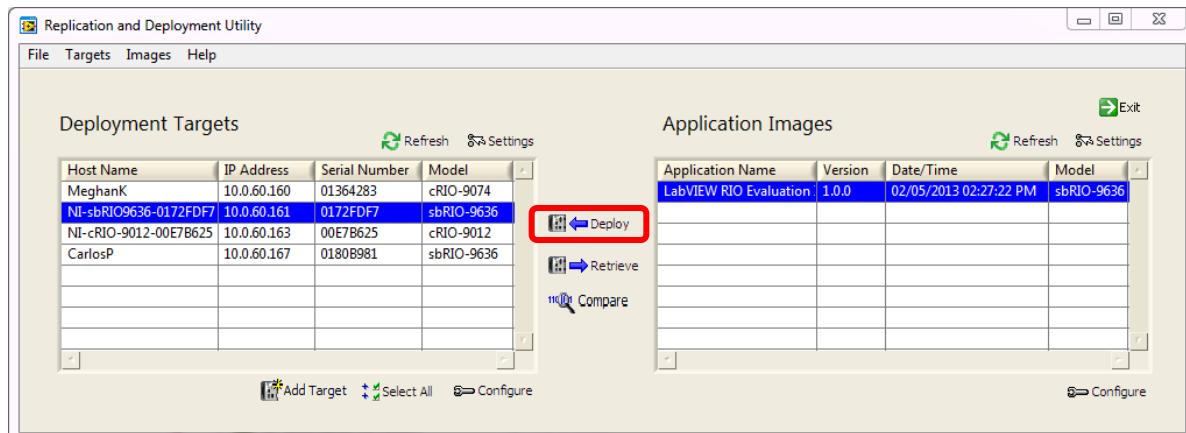
13. In the utility click **Refresh** to force the application to rescan the network and update the target information.



14. Since you have not specified your exercise folder as the default place to scan for images there are no image files showing up in the Application Images table. To change the location where the RAD scans:

- ✓ Click on **Settings** in the upper left corner above the Application Images table
- ✓ Browse to .\5- DeployReplicate System\Images and select **Use Current Folder**
- ✓ Click **OK**

15. Select your target on the left and the image on the right and then click **Deploy**. Verify in the dialog window that appears that the target is listed and then click **Deploy Application Image to Listed Targets** to start the replication process.

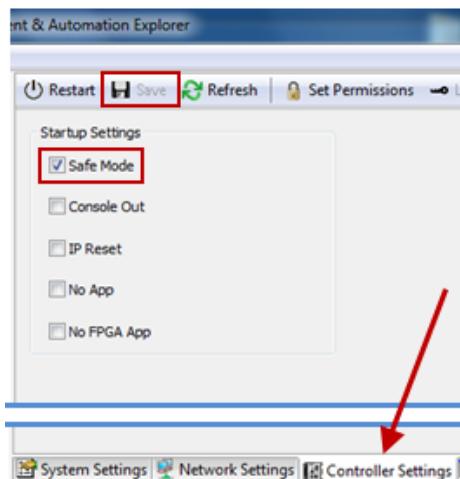


Note: The imaging process will take about 5 minutes.

16. After the replication is complete, the LCD screen will start updating again with your I/O, signaling the startup application is running. At this point close the dialog window.

Note: If you put the device into Safe Mode before, follow these steps to disable it now:

- ✓ Navigate to the **Controller Settings** tab for the NI-RIO Evaluation Device (shown below) in Measurement & Automation Explorer (NI MAX).
- ✓ Uncheck the box next to **Safe Mode** to reboot the target out of safe mode and click **Save**.
- ✓ When prompted to restart click **Yes**.
- ✓ Wait for the device to reboot and then move on to the next step.



17. To verify that the target has successfully been replicated, re-run your Windows application (\5- DeployReplicate System\builds\Windows Application\Exercise 5 Windows Application.exe) and you should receive updates from the embedded system.

Congratulations! Using NI's graphical system design approach, you have now completed the development, deployment, and replication of a LabVIEW RIO-based embedded system with three targets (Desktop PC running Windows OS, Processor running a Real-Time OS, and an FPGA).

Next Steps

This evaluation tutorial was an introduction to the LabVIEW RIO Architecture. Before you purchase NI LabVIEW and a RIO hardware device to start programming for your application, please review the following:

1. LabVIEW RIO Architecture Training Path

Since this was a brief introduction to the LabVIEW Real-Time and LabVIEW FPGA modules it is highly recommended that you better understand what further knowledge you need to gain before you start creating your own system. Appendix A has a guide to help you identify a training path to gain the appropriate skill level for the task you are trying to complete using LabVIEW.

2. RIO Hardware Form Factors

In this evaluation kit you used a board-level form factor of the RIO hardware platform. This however is just one form factor of many different families of RIO hardware products that can all be similarly programmed with the LabVIEW FPGA and LabVIEW Real-Time modules.



Learn more about the other families by visiting ni.com/embeddedsystems.

3. Online Community with Further Exercises and Resources

To find getting started resources, more advanced tutorials specifically for the LabVIEW RIO Evaluation Kit, user applications, discussion forums, and to learn more about LabVIEW RIO Architecture products, visit ni.com/rioeval/nextstep.

Appendix A | LabVIEW RIO Training Path

Maximize Your RIO Investment Develop Faster and Reduce Maintenance Costs

For developing embedded control and monitoring systems, the combination of NI LabVIEW software and NI CompactRIO or NI Single-Board RIO hardware offers powerful benefits including the following:

- Precision and accuracy - Precise, high-speed timing and control combined with accurate measurements
- Flexibility – Hundreds of I/O modules for sensors, actuators and networks that with LabVIEW can connect quickly to control and processing algorithms and system models
- Productivity – LabVIEW system design software for programming processors, FPGAs, I/O and communications
- Quality & ruggedness – High-quality hardware and software for deploying reliable embedded systems that last

However, there is still a learning curve to effectively take advantage of these benefits, and your application or job in part determines the size of that curve. Every project is different. To be successful, you should determine up front what you need to learn to deliver a system that meets or exceeds requirements while also minimizing development time. If the requirements for your next project differ significantly from your current one, assess what additional concepts you should learn to successfully complete it. For example, you may be currently developing a functional prototype and just want a system that works, but if the design is approved you will likely want something that is built to last and minimizes long-term maintenance costs. Consider the different capabilities needed for each stage of developing an application based on CompactRIO or NI Single-Board RIO, and take advantage of resources that can help you efficiently learn those necessary skills.

Core Capabilities Required for all CompactRIO and NI Single-Board RIO Users

To begin with, everyone who uses LabVIEW and CompactRIO or NI Single-Board RIO should have the ability to

- Install and configure CompactRIO hardware and LabVIEW software
- Create a diagram or architecture for your system
- Navigate the LabVIEW environment
- Apply key LabVIEW structures (While Loops, clusters, arrays, and so on)
- Develop basic, functional applications in LabVIEW
- Apply common design patterns (state machine, producer/consumer, and so on)
- Understand the difference between Windows and real-time operating systems
- Implement communication between processes
- Deploy an application

To help you learn these abilities, National Instruments recommends the following resources:

- Getting Started With NI Products (ni.com/gettingstarted)
- LabVIEW Core 1, LabVIEW Core 2, and LabVIEW Real-Time 1 training courses (ni.com/training)
- LabVIEW for CompactRIO Developer's Guide (ni.com/compactriodevguide)

From there, attributes of your application or job determine whether you need additional capabilities.

Identifying Additional Capabilities Required by Your Application

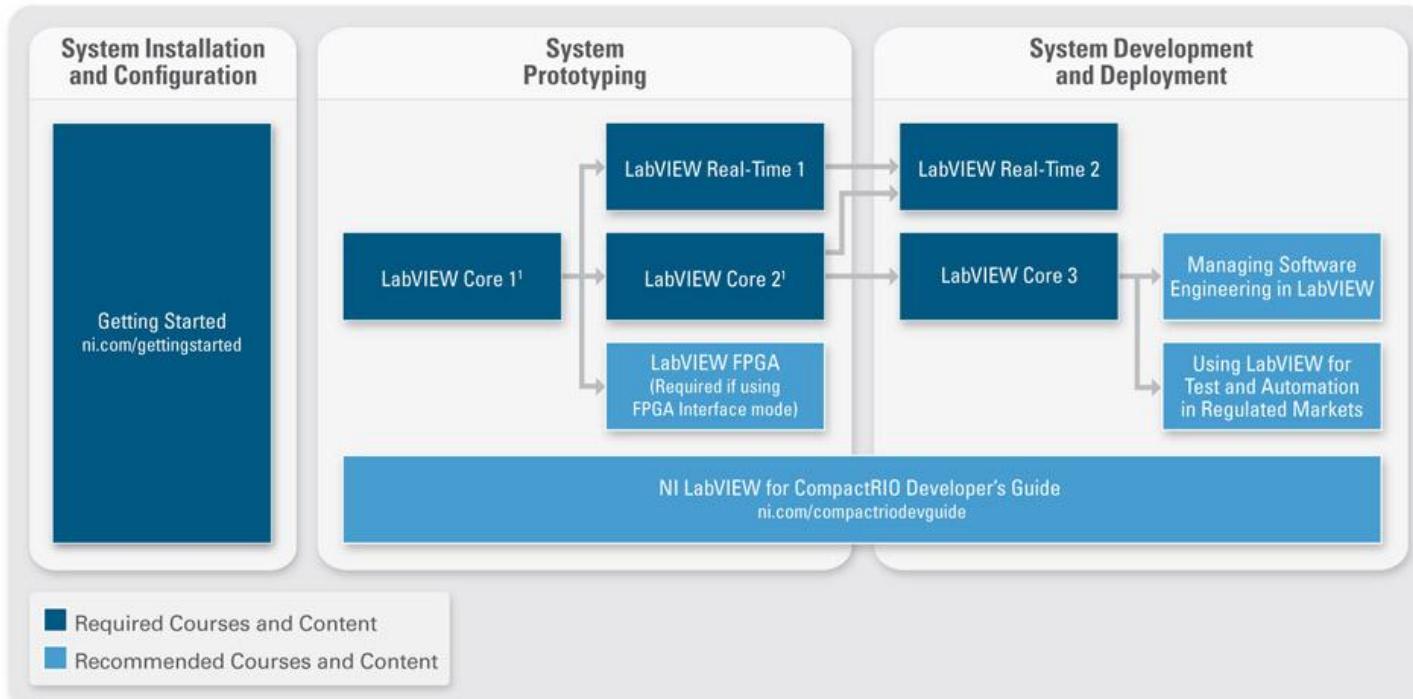
To determine the level of skills you need, ask the following four questions (circle one answer for each). For each of your answers, look at the following table to identify the capabilities you need and learning resources you can use.

1. How will the system I am developing be used?
 - a. Ongoing use or deployment over multiple months or years
 - b. Functional prototype or short-term use system
2. What I/O rates does my application require?
 - a. One or more I/O channels sampled or updated at >500 Hz
 - b. All I/O channels sampled or updated at rates <500 Hz
3. Who is developing the LabVIEW code for this system?
 - a. Multiple developers with each responsible for a portion of the codebase
 - b. A single person is developing the entire codebase
4. Will this system be used in the medical device industry to automate a manufacturing process or test products?
 - a. Yes
 - b. No

Questions and Answers		You Need to Be Able To...	Recommended Resources
1	a	No additional capabilities required	
	b	<ul style="list-style-type: none"> • Follow software engineering best practices to create scalable, maintainable applications in LabVIEW • Identify performance, reliability, and communication requirements for your system • Optimize your code to meet those requirements • Design for reliability: Build in system health monitoring and comprehensive error handling 	NI training courses: <ul style="list-style-type: none"> • LabVIEW Real-Time 2 • LabVIEW Core 3 ni.com content: <ul style="list-style-type: none"> • LabVIEW for CompactRIO Developers Guide
2	a	<ul style="list-style-type: none"> • Compile and deploy your VIs to hardware targets based on reconfigurable I/O (RIO) • Use an FPGA to acquire and output analog and digital signals • Understand and control timing of operations on the FPGA target 	NI training courses: <ul style="list-style-type: none"> • LabVIEW FPGA ni.com content: <ul style="list-style-type: none"> • LabVIEW for CompactRIO Developers Guide
	b	Acquire I/O using the NI Scan Engine	ni.com content: <ul style="list-style-type: none"> • LabVIEW for CompactRIO Developers Guide
3	a	No additional capabilities required	
	b	<ul style="list-style-type: none"> • Adapt the software engineering process to your project • Select and use appropriate tools to help you manage application development • Conduct an effective LabVIEW code review • Develop a test and validation strategy 	NI training courses: <ul style="list-style-type: none"> • Managing Software Engineering in LabVIEW
4	b	No additional capabilities required	
	a	<ul style="list-style-type: none"> • Understand regulatory requirements in the industry • Follow best practices for using standards and application life cycle processes • Use the GAMP 5 risk-based approach for developing test applications • Take advantage of NI tools and techniques to simplify testing and documentation requirements 	NI training courses: <ul style="list-style-type: none"> • Using LabVIEW for Test and Automation in Regulated Markets

Table 1. *Identify the capabilities you need for your project and the learning resources you can use.*

CompactRIO/Single-Board RIO Recommended Resources Summary



¹ Since LabVIEW Core 1 and 2 fit into one week, you may choose to take both LabVIEW Core classes prior to LabVIEW FPGA and LabVIEW Real-Time 1.

Need More Help?

Contact a National Instruments Training & Certification Specialist at ni.com/contact for additional guidance on the level of skill you need for your application.

No Time to Learn?

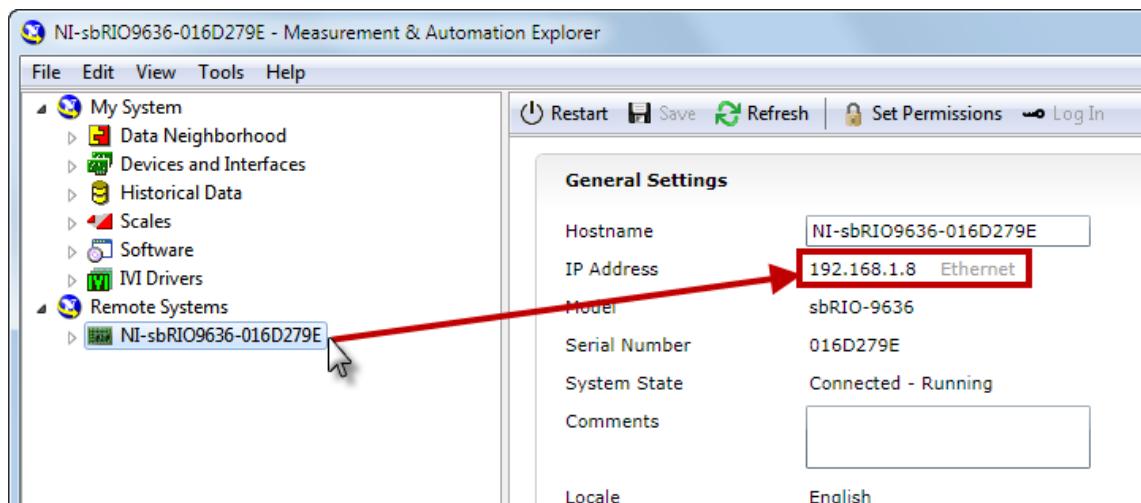
Many National Instruments Alliance Partners have already invested in the level of proficiency required for your application. If you have a CompactRIO or NI Single-Board RIO project that requires a greater skill level than you currently have and you are unable to gain the required level in the time allotted for your project, NI can temporarily augment your expertise by connecting you with an Alliance Partner that can provide consulting services while you get up to speed. Find an Alliance Partner in your area at ni.com/alliance.

Appendix B | Changing the IP Address in the LabVIEW Project

Your RIO device is identified by its IP address. For each exercise, confirm that the IP address in the project matches the IP address of your RIO device. The National Instruments LabVIEW RIO Evaluation Setup utility should have prompted you to write down the NI-RIO Evaluation Device IP address, but you also can locate the device through the following steps.

If you already know your IP address, skip to Step 4.

1. Determine the IP address of your NI-RIO Evaluation Device by opening the Measurement & Automation Explorer with the icon on your desktop or by selecting **Start»All Programs»National Instruments»NI MAX**.
2. Click the triangle next to Remote Systems.
3. Click on your NI-RIO Evaluation Device in the Remote Systems tree and on the right hand side note the IP address that appears in the System Settings tab.



4. Change the IP address of the *NI-RIO Evaluation HWtarget* in the LabVIEW Project Explorer window to match the IP address of your evaluation board.
 - a. Right-click the *NI-RIO Evaluation HWtarget* in the LabVIEW Project Explorer window and select **Properties** from the menu to display the General properties page.
 - b. In the **IP Address / DNS Name** box, enter the IP address you wrote down from the National Instruments LabVIEW RIO Evaluation Kit Setup utility or just now from Measurement & Automation Explorer and click **OK**.
 - c. Right-click on the *NI-RIO Evaluation HWtarget* in the Project Explorer window and select **Connect** to verify connection to the evaluation device.