Build an Embedded System

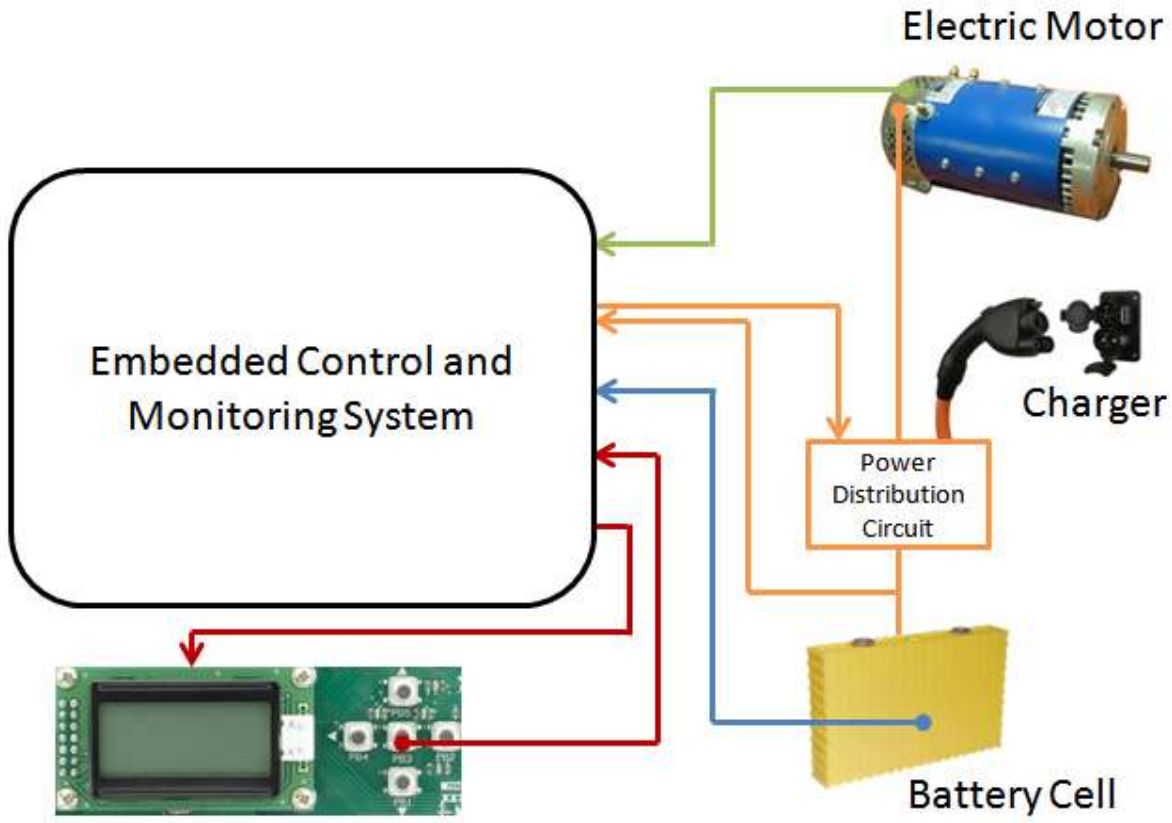
NIDAYS 2013

NATIONAL INSTRUMENTS™

Table of Contents

Tutorial Overview

In this tutorial, you will complete four exercises that introduce you 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:



Battery Management System Specifications

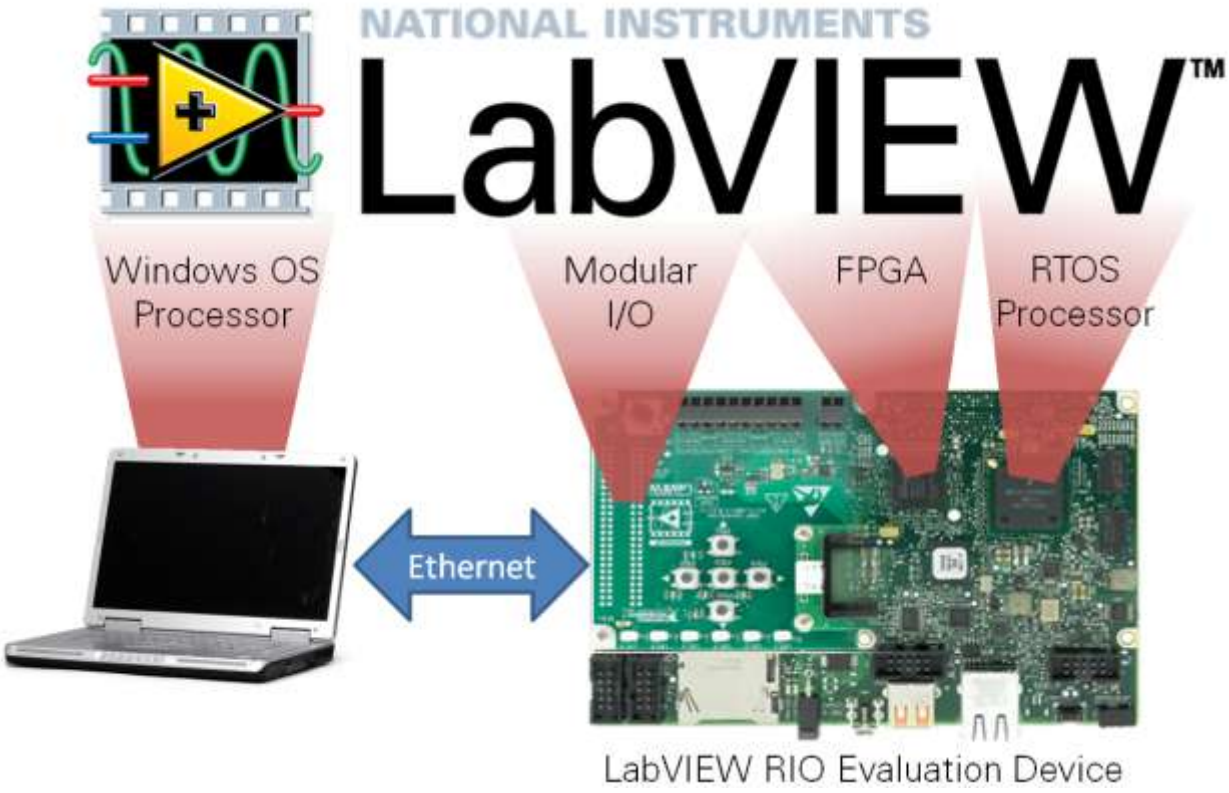
Monitoring Tasks

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

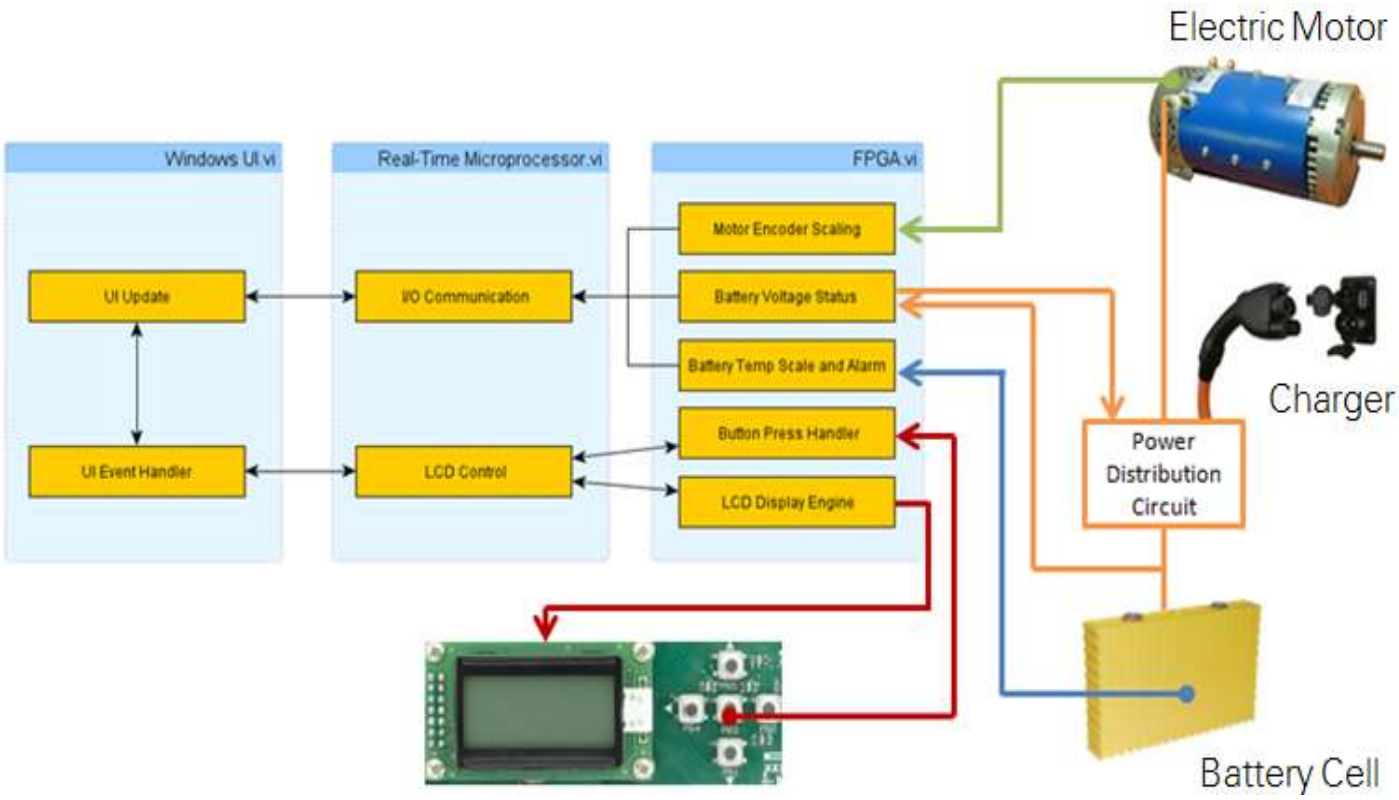
Control Tasks

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

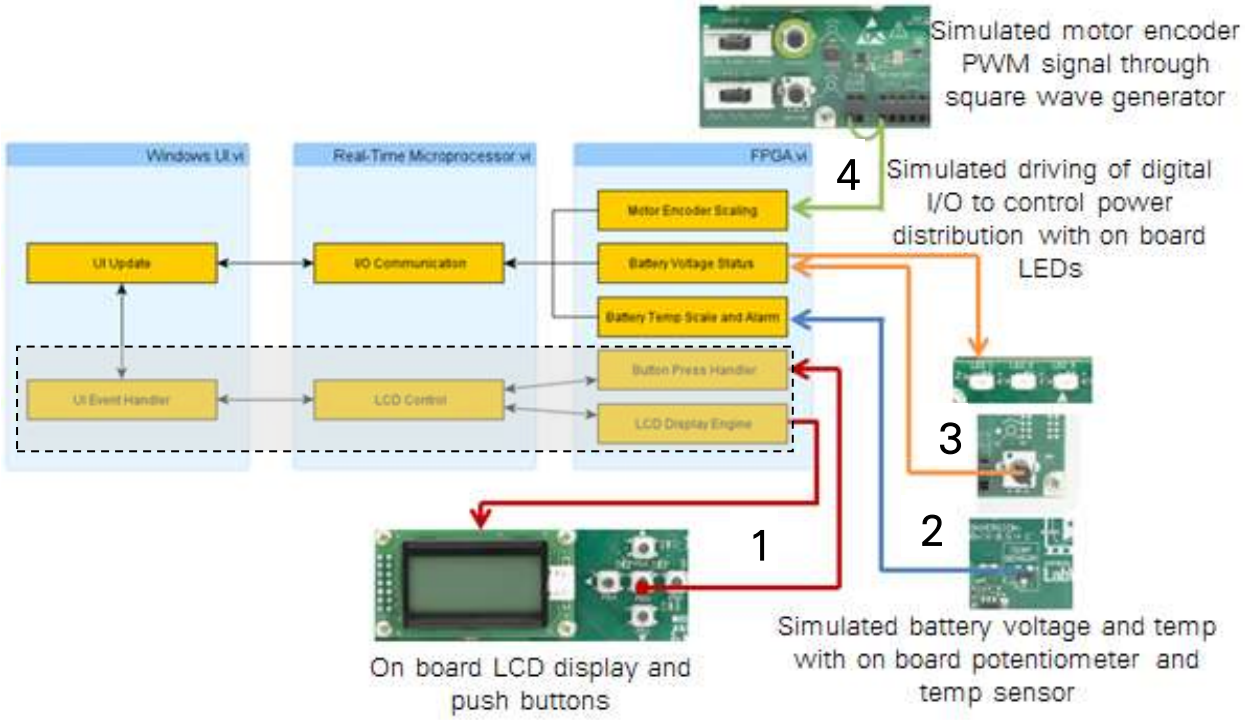
The system you are developing in this tutorial will include both a RIO board-level device and your Windows PC. Since LabVIEW includes a cross-compiler, it can be used to develop applications that will run on a floating point processor, an FPGA target, and a Windows PC.

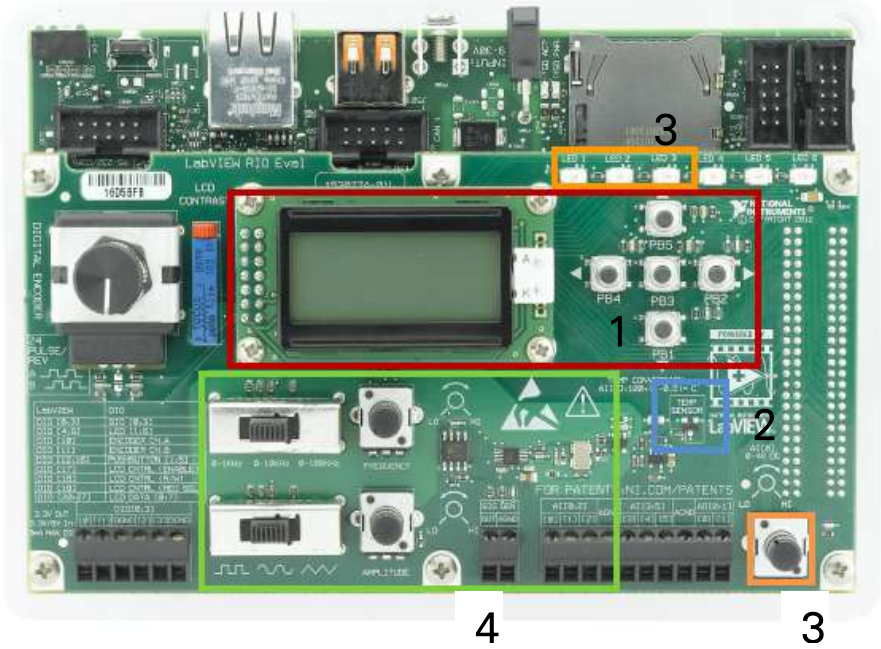


Using the flexibility of the three different target types, the requirements of the electric vehicle battery management system outlined in the previous pages have been mapped to tasks on each of the targets of the system (Windows UI, Real-Time microprocessor, and FPGA):



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: The UI Event Handler, LCD Control, LCD Display Engine and Buttorn Press Handler are not part of the NIDays Embedded exercise 3 and 4.

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.

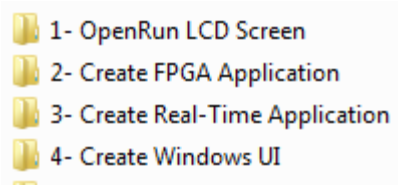
Exercise 2: Create a Monitoring and Control FPGA Application
Finish an FPGA application to acquire data from and control your I/O for the battery management system.

Exercise 3: Develop Real-Time Application
Finish a real-time application running on a processor which communicates with the FPGA 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.

Navigating the Exercises

The exercises are located on your computer's C: drive at **C:\Users\Public\Public Documents\National Instruments\NIDays\LabVIEW RIO** and contain the folders below:



Using the Solutions

Exercises 2 through 4 each have a solution in the **_Solutions** folder 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.

Exercise 1 | Open and Run LCD Application

Summary

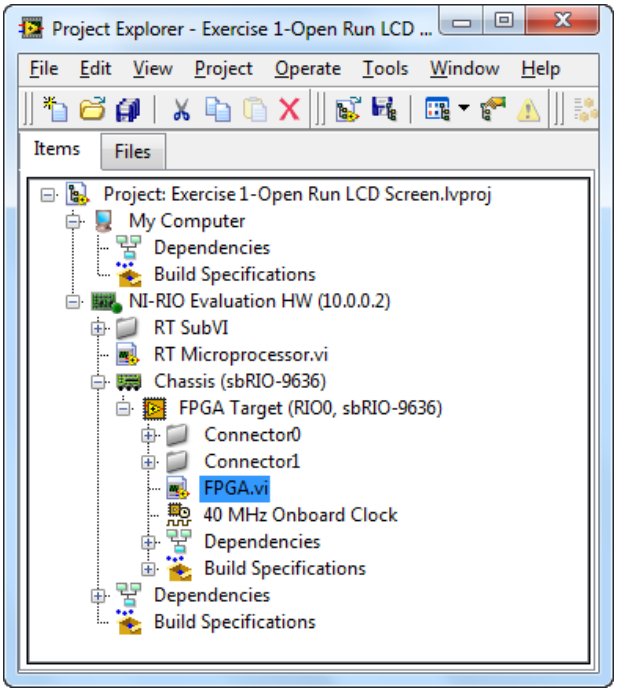
In this exercise you are going to open and run a preconfigured application that communicates between the Real-Time Microprocessor and FPGA to scroll text across the hardware's LCD screen. To deploy the application, you will complete the following tasks:

- Explore the Application
- Deploy and Run Your First LabVIEW RIO Application

Explore the Application

1. Open the Exercise 1 LabVIEW project file by navigating to C:\Users\Public\Public Documents\National Instruments\NIDays\LabVIEW RIO\1-OpenRun LCD Screen\Exercise 1-Open Run LCD Screen.lvproj.

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



3. 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.

4. Open the block diagram by pressing **CTRL+E** or navigating to **Window»Show Block Diagram** and observe that the FPGA VI uses low-level LCD driver VIs to interface with the LCD screen. Press **CTRL+H** to open the **Context Help** window, which gives

details about the code your cursor interacts with.

5. Close the FPGA VI. Do not save if prompted.

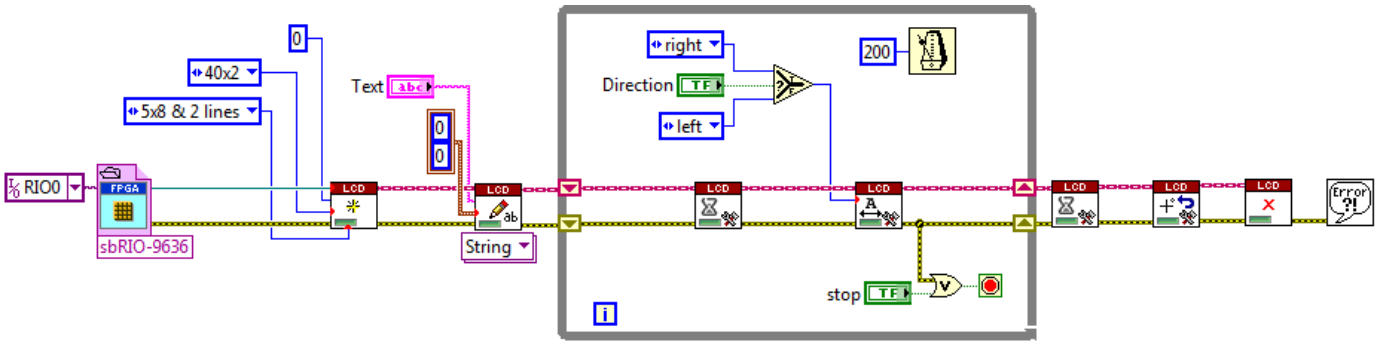
6. In the project, navigate to and double-click on the **RT Microprocessor.vi** that will execute on the Real-Time Operating System (RTOS) running on the processor.

7. 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.

8. Open the block diagram by pressing **CTRL+E.**

9. Observe that the real-time application opens a reference to the FPGA VI on the far left-hand side, and then sends commands to the FPGA through LCD driver VIs to drive the LCD screen through digital I/O lines. The application stops upon an error or pressing the Stop button and then the FPGA LCD Reference is closed.

   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 status 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.

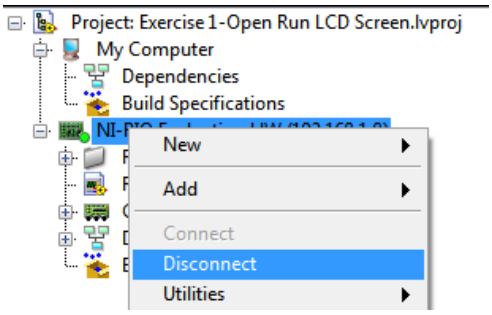
2. Keep the real-time application running. View the LCD screen on your target and view the scrolling text. Click on the **direction** horizontal toggle switch on the front panel to scroll the text in the opposite direction. Click the **STOP** button.

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 screen contrast is set appropriately by using your NI screwdriver to turn the LCD Contrast control, located to the left of the screen.

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. Close out all LabVIEW files, and do not save if prompted.



Note: 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 NI-RIO hardware!

Exercise 2 | Create a Monitoring and Control FPGA Application

Summary

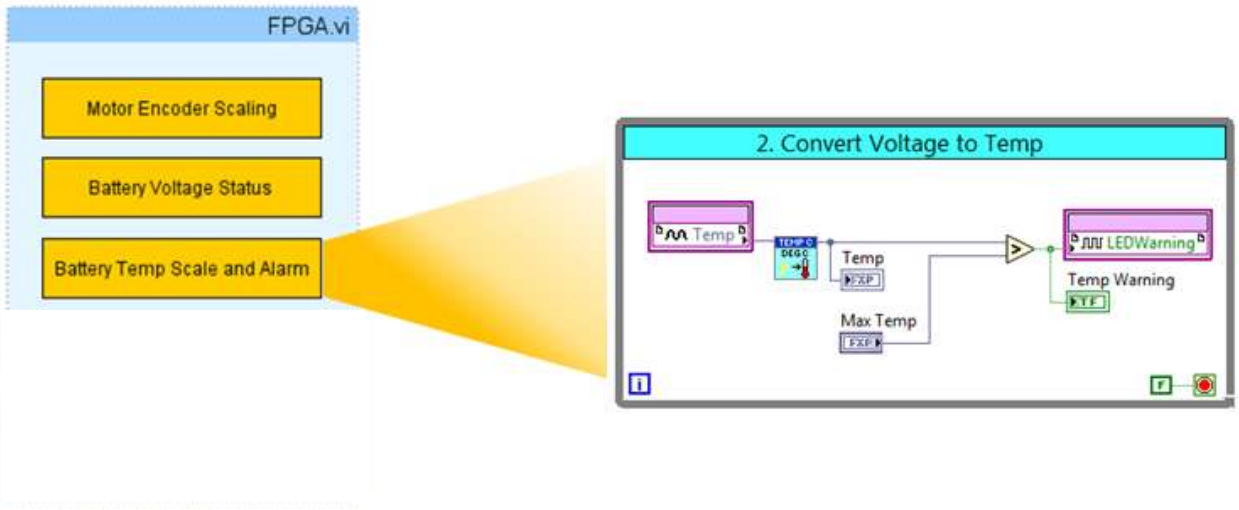
To start the development of your embedded system you are going to create the LabVIEW FPGA application which acquires data from and controls your I/O for the Battery Management System. In this exercise you will implement these tasks:

- Monitor the Battery Cell Temperature
- Monitor the Battery Cell Voltage and Control Power Distribution

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.

Monitor the Battery Cell Temperature

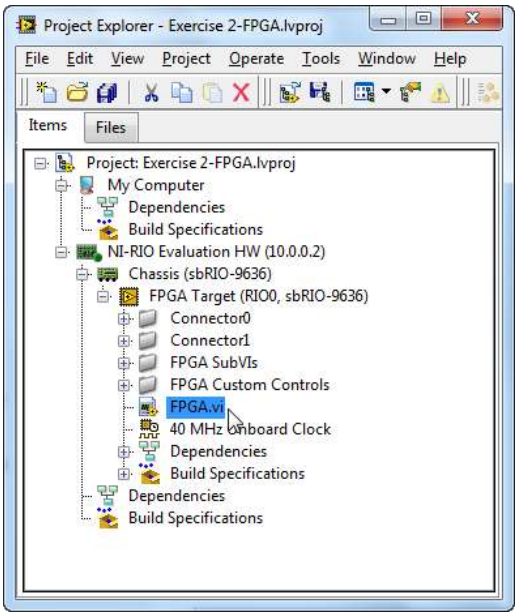
The next while loop will acquire the battery cell temperature voltage, scale it to degrees Celsius and compare it against a maximum temperature for alarming purposes. You will finish this task.



1. In LabVIEW select **File»Open Project**… and open the Exercise 2-FPGA project file located at C:\Users\Public\Documents\National Instruments\NIDays\LabVIEW RIO\2-Create FPGA Application\Exercise 2-FPGA.lvproj.

2. Expand out *NI-RIO Evaluation HW*, *Chassis,* and *FPGA Target* in the Project Explorer to expose the FPGA target.

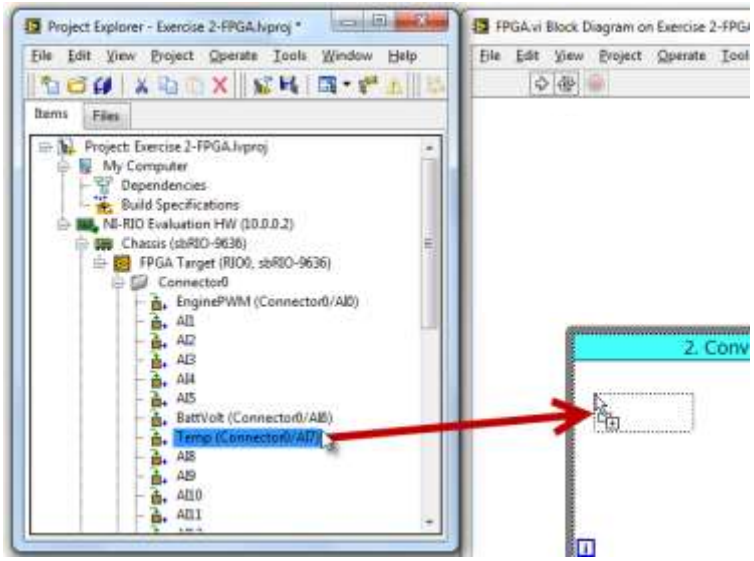
3. Double-click on **FPGA.vi** to open up the existing LabVIEW FPGA application.



4. Switch to the block diagram by pressing **CTRL+E** or navigating to **Window»Show Block Diagram**.

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

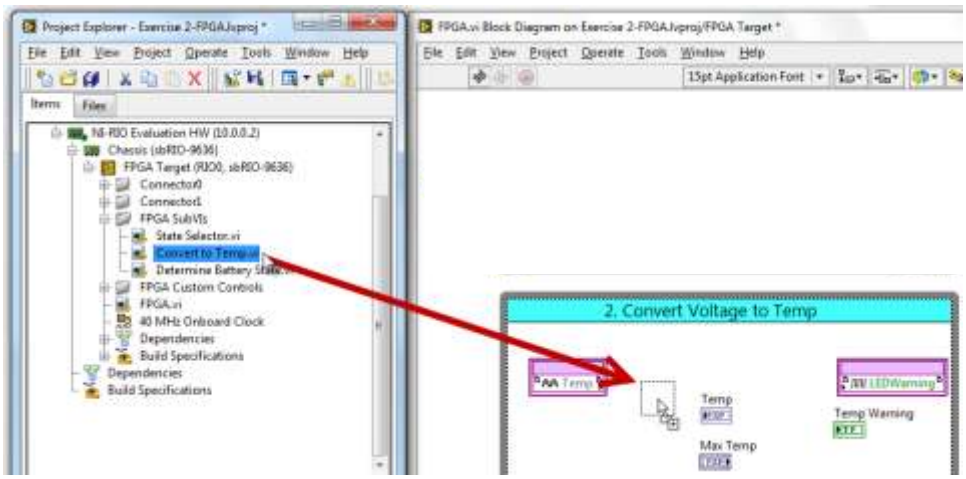
1. In the Project Explorer window under the FPGA Target expand out **Connector0** which exposes the I/O channels available from that connector. Locate the **Temp** (**Connector0/AI7**) channel and drag it into the left-hand side of the **Convert Voltage to Temp** while loop.



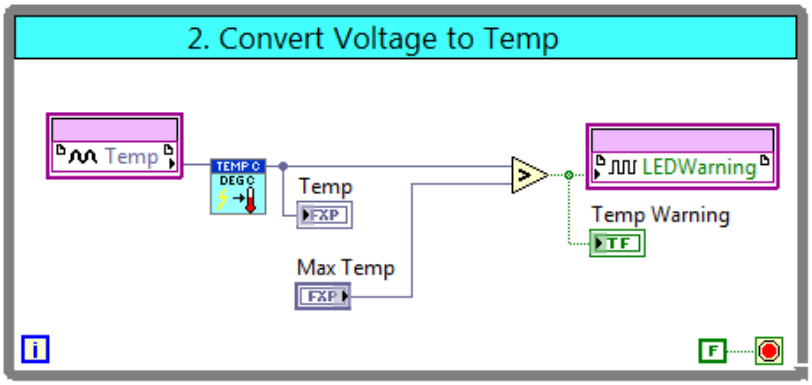
2. Still in the Project Explorer window, expand out **Connector1** and locate the **LEDWarning** (**Connector1/DIO9**) digital channel. Drag it into the right-hand side of the same while loop.

3. Right-click on the **LEDWarning** I/O node name 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.



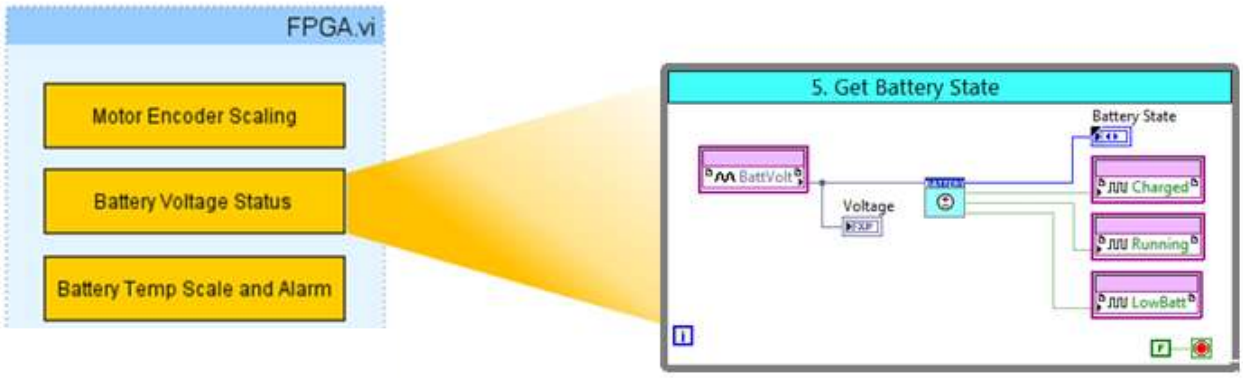
6. Insert a **Greater?** (**Functions»Programming»Comparison»Greater?**) logic comparison into the while loop and wire all the components in the loop as shown below. The **Convert Voltage to Temp** while loop is now complete.



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

Monitor the Battery Cell Voltage and Control Power Distribution

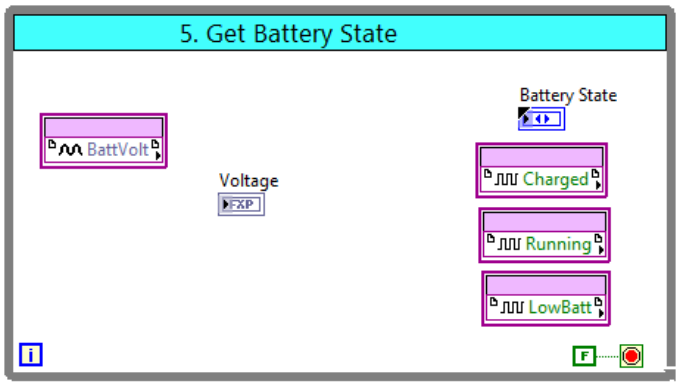
The last FPGA-based task is to acquire the current battery cell voltage, and based on its level, assert digital output lines in order to control a power distribution to charge the cell, draw off the cell, or throw an error if it has reached the voltage limit.



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.

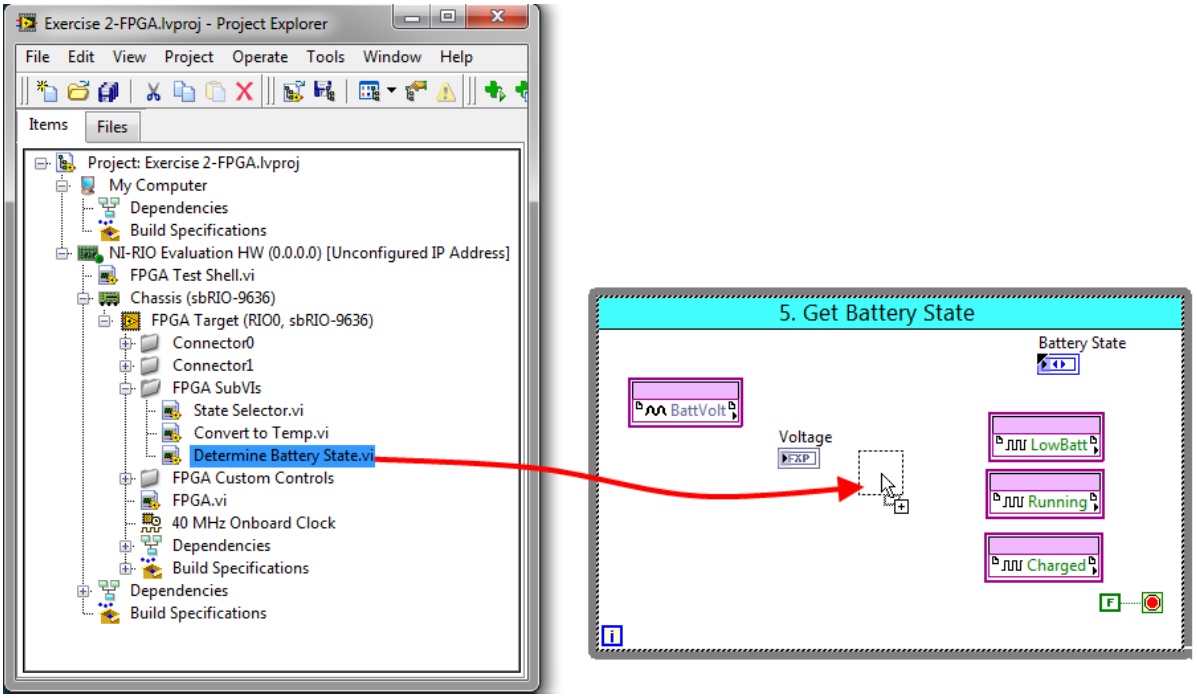
2. To assert the digital lines, expand out **Connector1** in the Project Explorer and drag over the following digital lines into the right-hand side of the same loop.

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

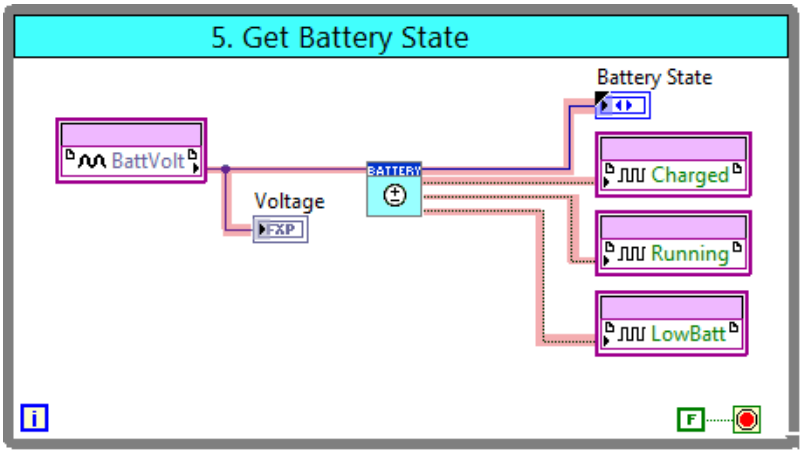


3. Hover over the **Charged** digital I/O node that was inserted in the last step until the cursor changes to a hand. Then right-click on the node and select **Change to Write**.
4. Repeat step 3 for the remaining two digital I/O nodes.

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



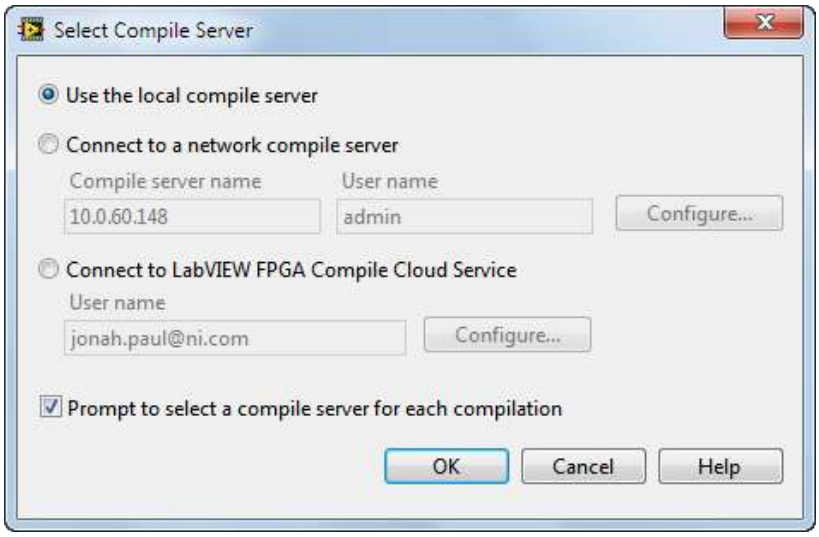
7. Save the FPGA VI by selecting **File»Save** or pressing **CTRL+S**.

Start LabVIEW FPGA Compilation Process

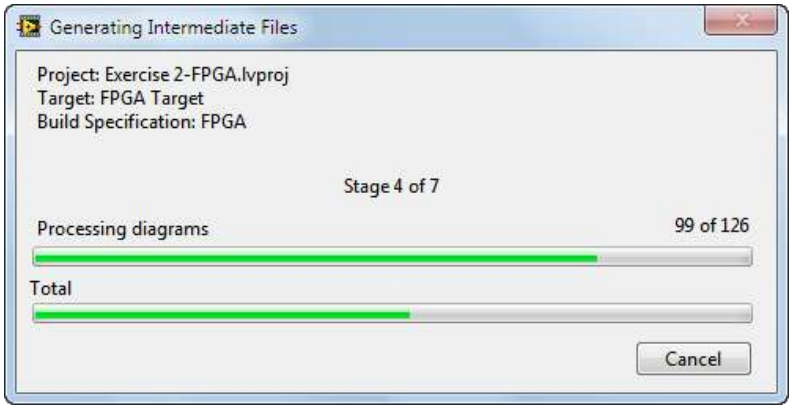
1. Navigate to **File»Save All** to save the FPGA VI and click the **Run** button to start the compilation process.

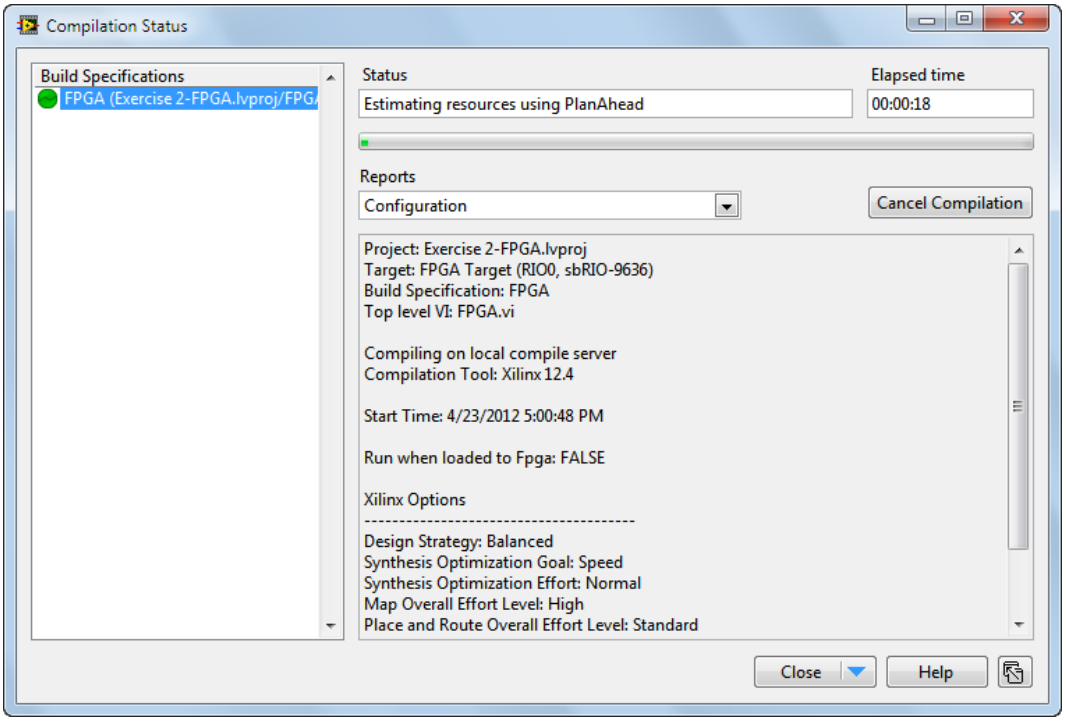
Note: In contrast to LabVIEW applications running on a Windows OS, an FPGA-based VI is compiled down to a bitfile, which is loaded onto the FPGA chip configuring its logic cells and I/O blocks to implement the requested logic. This is a two-step process. First LabVIEW generates intermediate files and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

2. In the dialog box that appears, 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 15 minutes to finish compilation. When complete, the FPGA VI will automatically start running in interactive mode if you leave it open.





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.

Test the FPGA Application

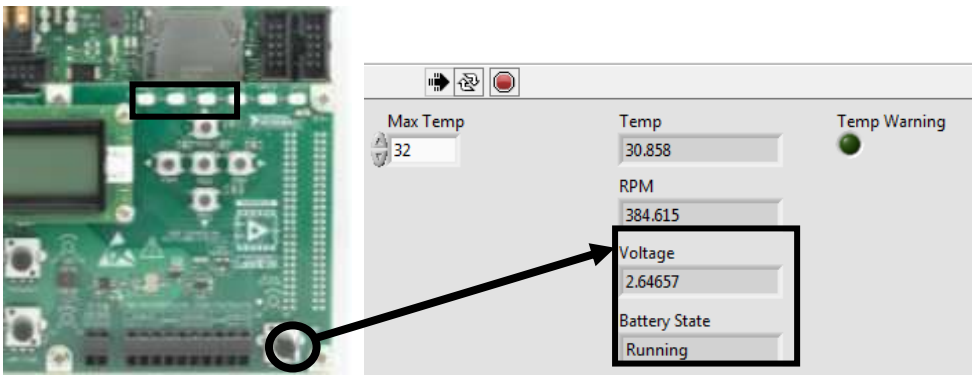
When the FPGA VI completes its compilation process, you can run the following tests on the FPGA application.

1. Double click on the **FPGA.vi** if it is not already open.

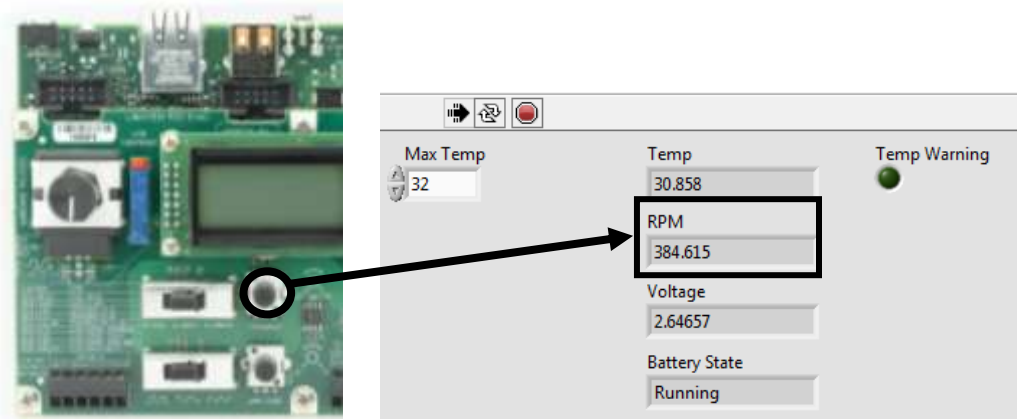
Note: If the FPGA VI did not complete compilation, reference the solution at …\BYOES\Solutions\ 2- Create FPGA Application and open the Exercise 2-FPGA Solution.lvproj.

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 called **Voltage** increases and the **Battery State** changes. Also note the changes on 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).

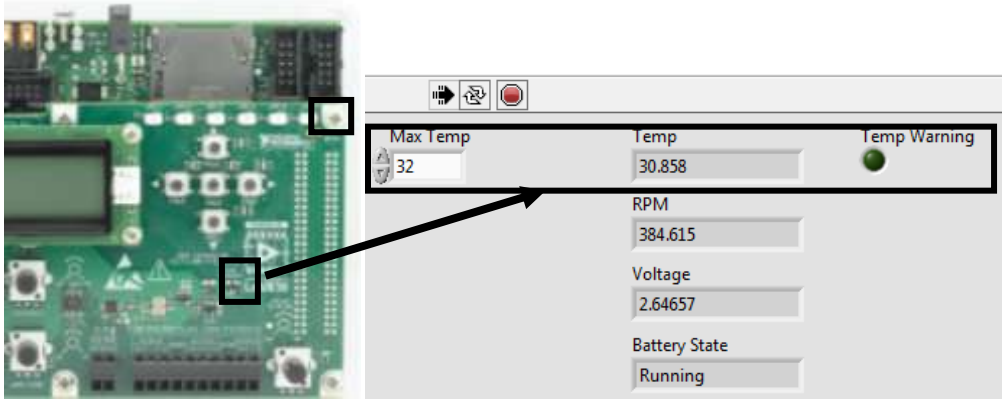


   ✓ Turn the potentiometer pictured below to vary the Motor PWM signal frequency and verify that the **RPM** signal responds accordingly on the FPGA front panel.



   ✓ 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** control so that you can test if the temperature is above the maximum that

the **Temp Warning** front panel alarm LED indicator 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 push buttons PB1, PB2, PB4, and PB5 and verify that the **FPGA Display Command** indicator updates with a new menu item.

3. When you are finished testing the FPGA VI, click the **Abort** button  .

4. 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.

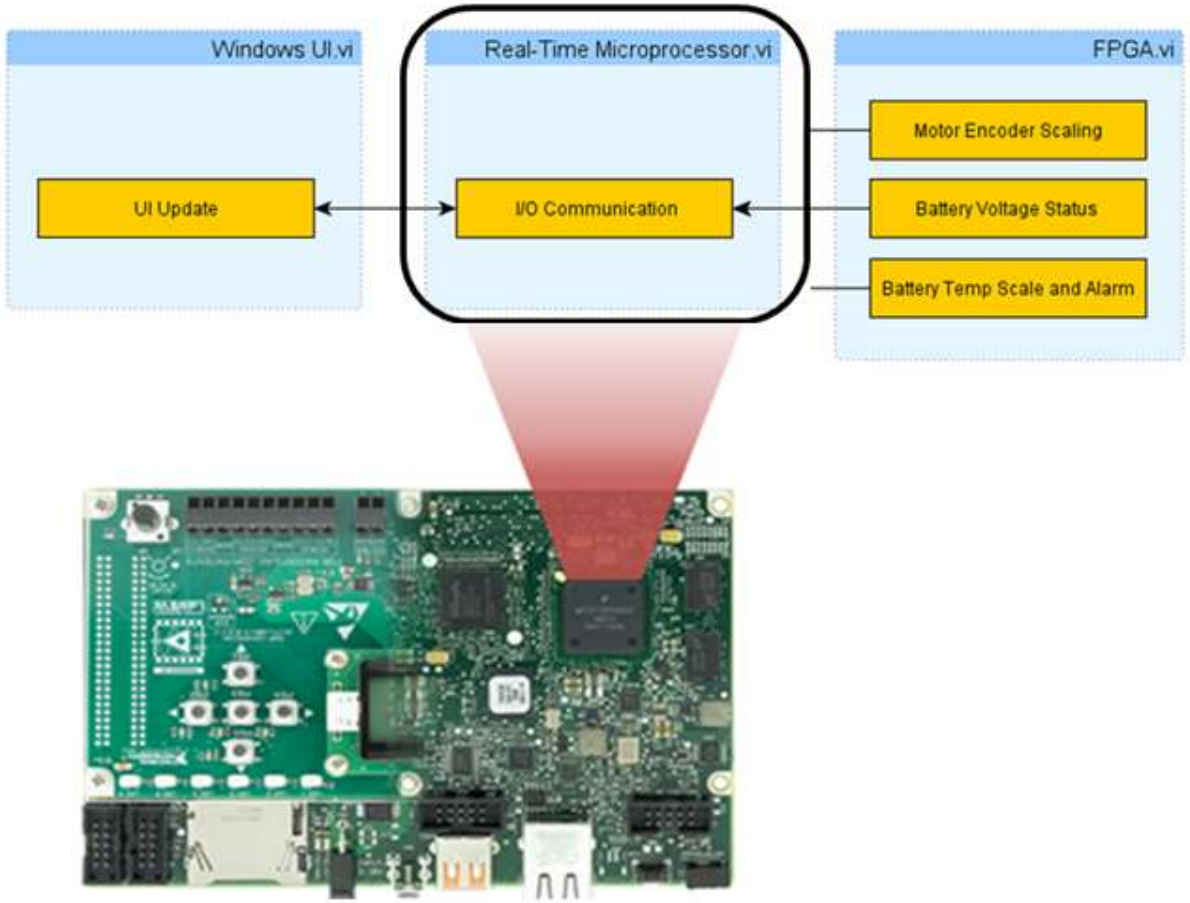
Congratulations, you have completed your FPGA application testing! As you can see, the FPGA VI front panel is very helpful to test your I/O and is efficient for debugging purposes.

Exercise 3 | Develop a Real-Time Application

Summary

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

What am I going to accomplish in this exercise?

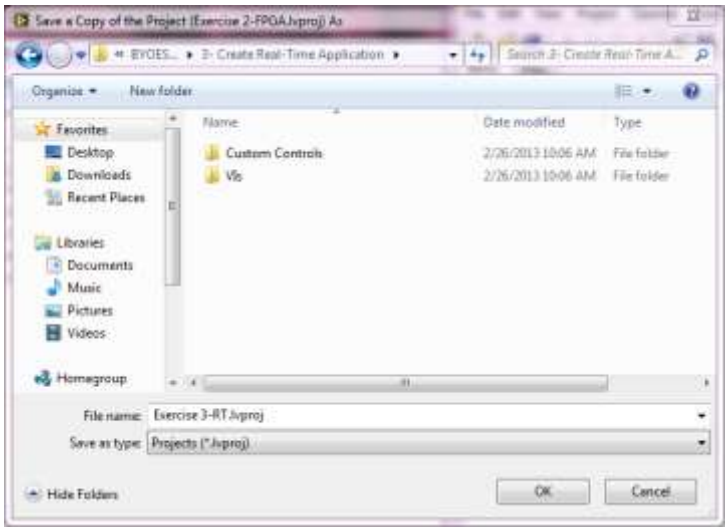


Project Setup

1. After your Exercise 2 FPGA VI is done compiling, return to the Project Explorer Window and select **File»Save As…** to save the project in the Exercise 3 folder.

2. In the Save As… dialog box that appears, select **Rename.**

3. Navigate to the .\LabVIEW RIO\3- Create Real-Time Application\ folder and rename your project as *Exercise 3- RT.lvproj*. Click **OK** to save the project in this new location.



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

4. To add the real-time VI's starting template, in the Project Explorer window right-click on *NI-RIO Evaluation HW* and select **Add»File…** Navigate to the .\3- Create Real-Time Application\VIs folder and select the **RT Microprocessor.vi**

5. To include the network-published shared variables that will be used to communicate with the Windows user interface, right-click on *NI-RIO Evaluation HW* and select **Add»File…** Navigate to the .\BYOES\3- Create Real-Time Application\ folder and select **RT-Host Shared Variables.lvlib**.

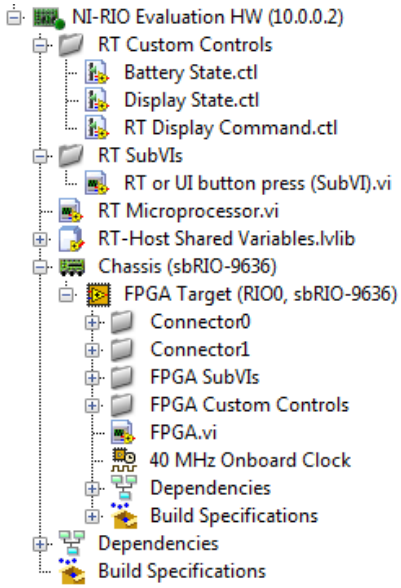
6. Right-click again on *NI-RIO Evaluation HW* and select **Add»Folder (Snapshot)…** Navigate into the .\BYOES\3- Create Real-Time Application\Custom Controls folder as shown below and select **Current Folder**.

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

8. Right-click a final time on *NI-RIO Evaluation HW* and select **New»Virtual Folder**. Name it **RT SubVIs** and then right-click on that virtual folder and select **Add»File…** Navigate into the .\BYOES\3- Create Real-Time Application\VIs folder and select **RT or UI button press (SubVI).vi**.

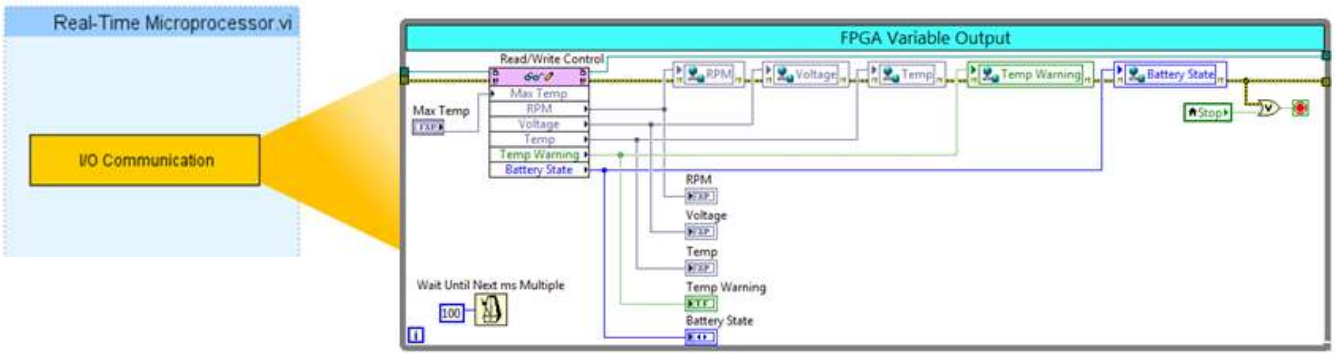
9. Expand out the RT Custom Controls and FPGA Custom Controls folder and verify that there are no file conflicts due to the project copy. If you observe that one of the controls has "conflict" noted next to its name in the project, call the instructor over to help you resolve it before moving forward.

Your project hierarchy underneath *NI-RIO Evaluation HW* should now look as shown below.

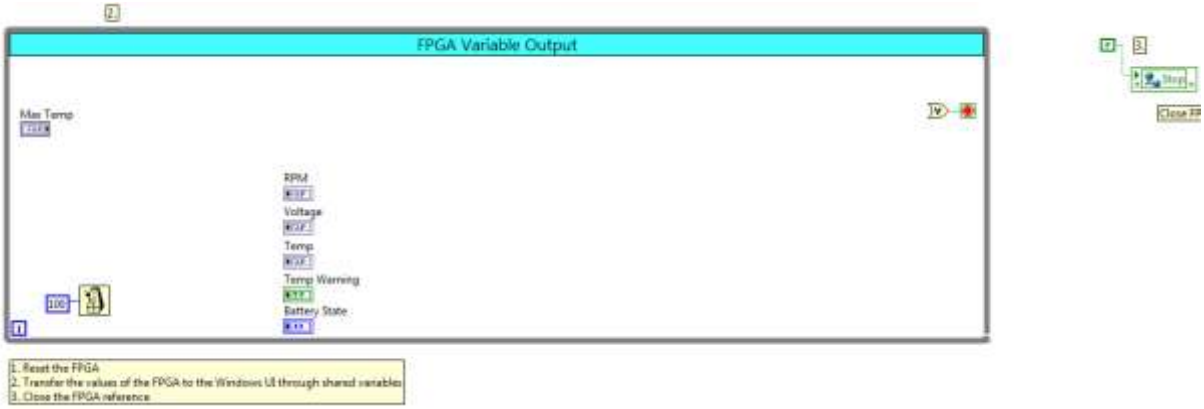


Communicate with the FPGA

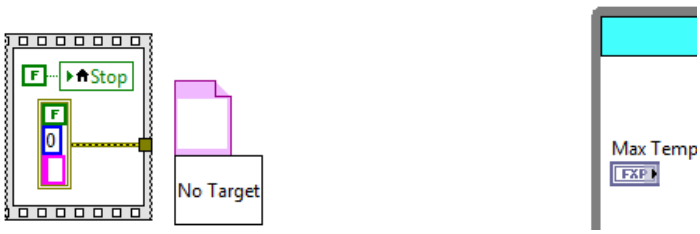
In this section you will open a reference to the FPGA target, implement communication with the target across the backplane bus, and close the reference.



1. Open up the **RT Microprocessor** VI from the project and toggle to the block diagram (Press **CTRL+E**). The run arrow is broken due to unwired terminals

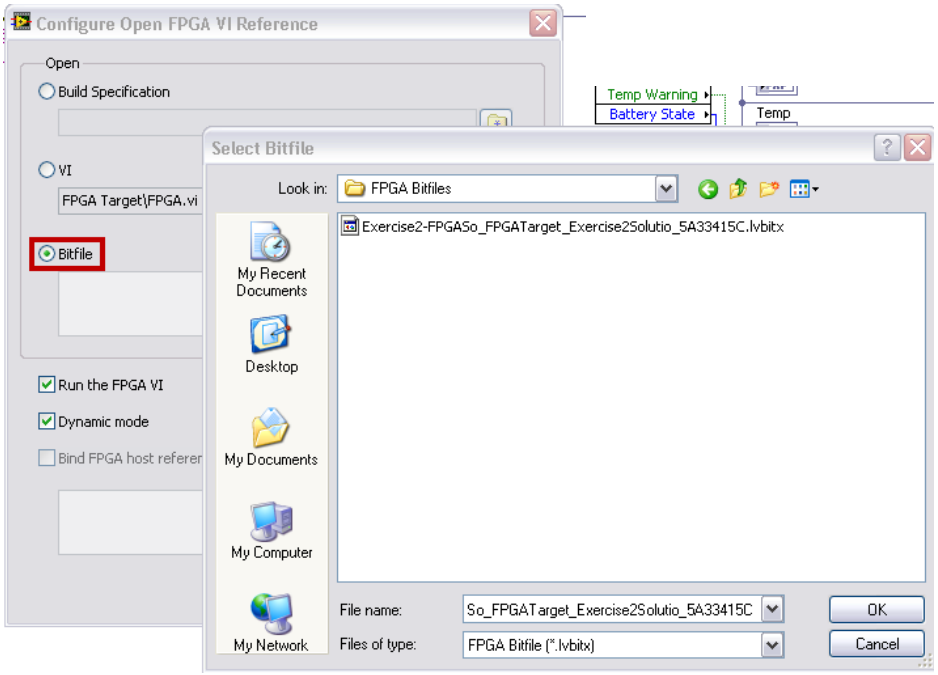


2. Navigate to the FPGA Interface palette and pin it to the block diagram. From this palette insert an **Open FPGA VI Reference VI** (**Functions»FPGA Interface»Open FPGA VI Reference**) on the block diagram as shown below.

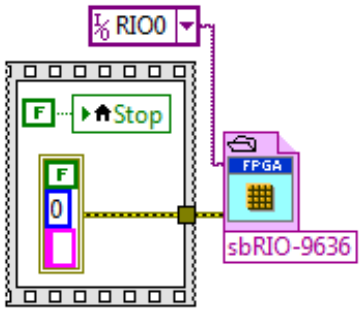


3. Since the FPGA VI was compiled in the Exercise 2 project, that compiled FPGA application, or bitfile, needs to be referenced in our current project. Right-click on the **Open FPGA VI Reference VI** and select **Configure Open FPGA VI Reference.**

4. In the dialog that appears click the radio button next to **Bitfile** and browse to .\BYOES\2- Create FPGA Application\FPGA Bitfiles\ and select the file inside. Click **OK**. This is the bitfile resulting from your Exercise 2 LabVIEW FPGA code compilation. If your Exercise 2 compilation did not complete successfully, use the Exercise 2 solution bitfile located at .BYOES \Solutions\2- Create FPGA Application\FPGA Bitfiles.

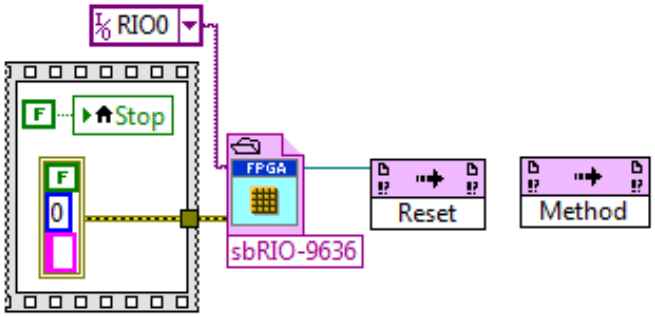


5. To finish the FPGA setup, right-click on the **resource name** input of the **Open FPGA VI Reference** and select **Create»Constant**. From the constant drop down arrow, select **RIO0**. This is the resource name of your FPGA target from the LabVIEW project.



6. Insert two **Invoke Method** nodes (**Functions»FPGA Interface»Invoke Method**) on the block diagram between the Open FPGA VI Reference and the while loop. These will reset and load the FPGA application respectively.

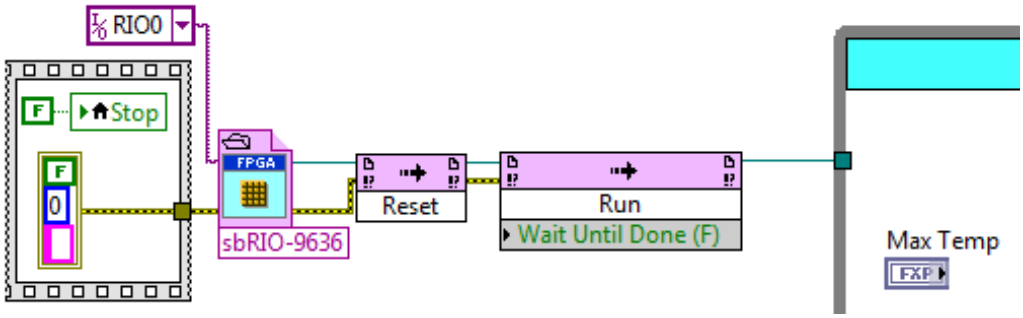
7. Wire the **FPGAVI Reference Out** of the Open FPGA VI Reference VI to the first **Invoke Method**, and then click on the white section titled **Method** and select **Reset**.



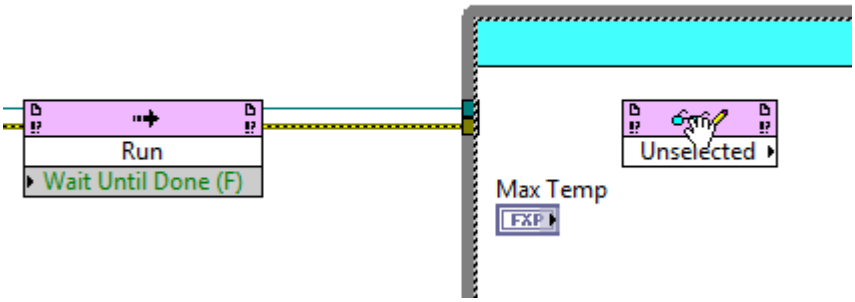
8. Wire the **FPGA VI reference** output of the first **Invoke Method** into the input of the second **Invoke Method** and select **Run** as the method for the second **Invoke Method**.

9. Next wire the **FPGA Reference VI** output of the second Run **Invoke Method** to the border of the **FPGA Variable Output** while loop.

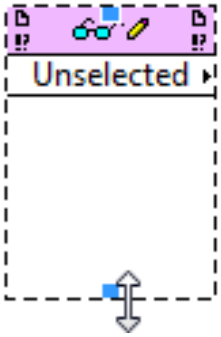
10. Complete the error wires as shown below. Also draw an error wire into the while loop from the Run Invoke Method.



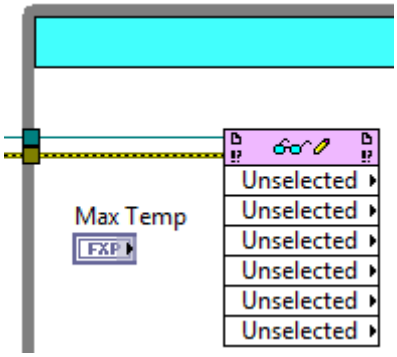
11. Now insert a **Read/Write Control** (**Functions»FPGA Interface»Read/Write Control**) inside the **FPGA Variable Output** while loop. This will define the data communicated between the FPGA and real-time application.



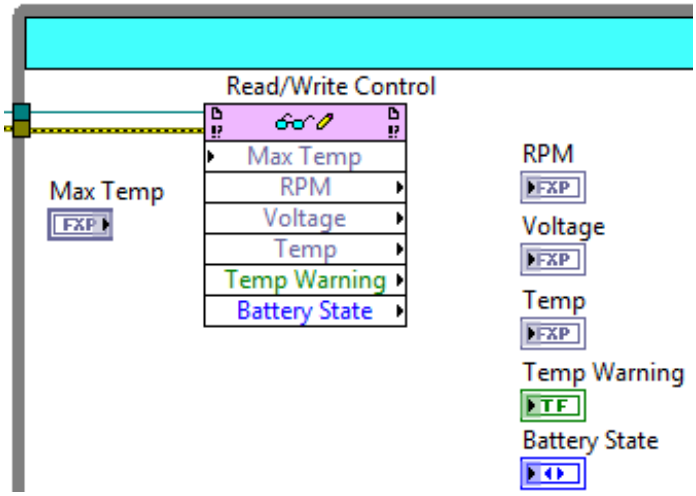
12. Expand the **Read/Write Control** to include six terminals by clicking on the lower border and dragging it down as shown below.



13. Wire the **error out** and **FPGA VI reference** from the loop's tunnel to the corresponding **Read/Write Control** input terminals.

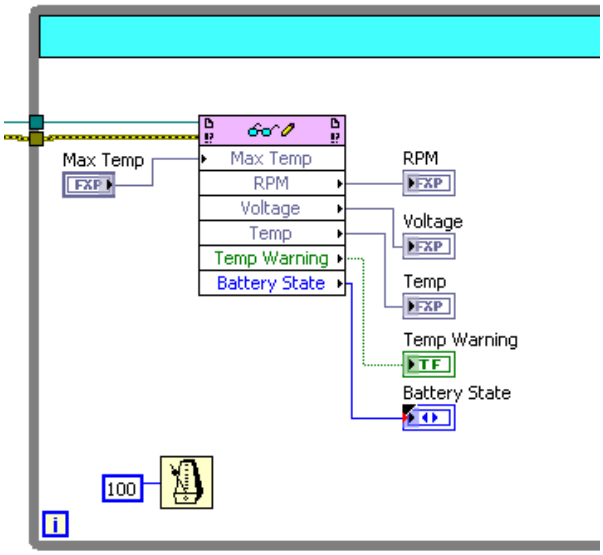


14. To configure the unselected terminals, click on each of them and select the control or indicator from the FPGA front panel that should be polled. Configure the terminals in the order shown below to include **Max Temp, RPM, Voltage, Temp, Temp Warning,** and **Battery State**.

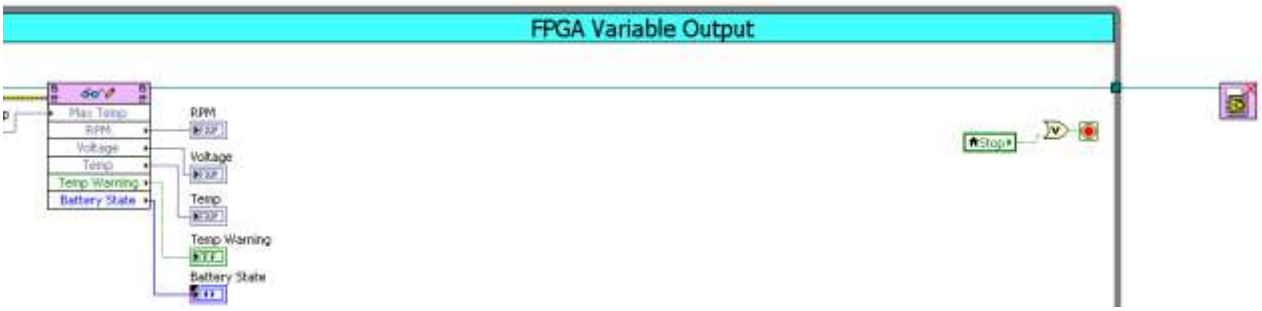


15. By default the terminals may all be inputs. If so, right-click on terminals 2-6 and select **Change to Read** to change them to outputs. **Max Temp** will be the only input.

16. Wire the **Read/Write Control** terminals to the matching front panel control and indicators as shown below.



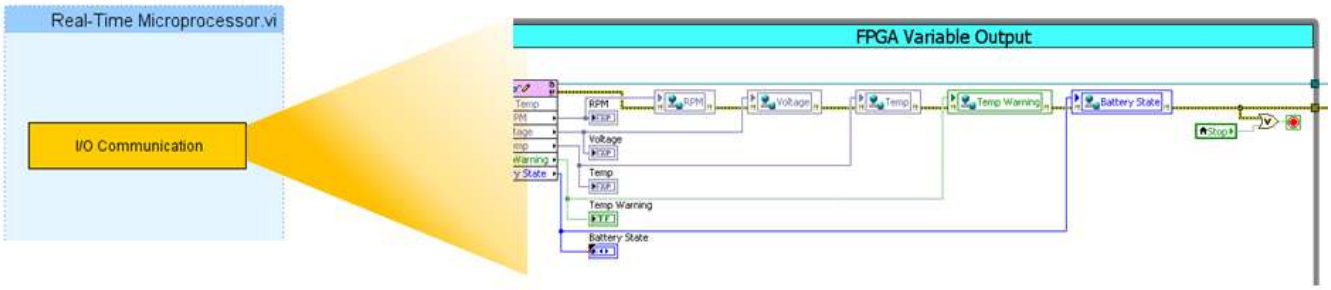
17. Finally, to properly close the reference to the FPGA target, insert a **Close FPGA VI Reference** function (**Functions»FPGA Interface»Close FPGA VI Reference**) to the right of the **FPGA Variable Output** while loop and wire the **FPGA VI reference** as shown.



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

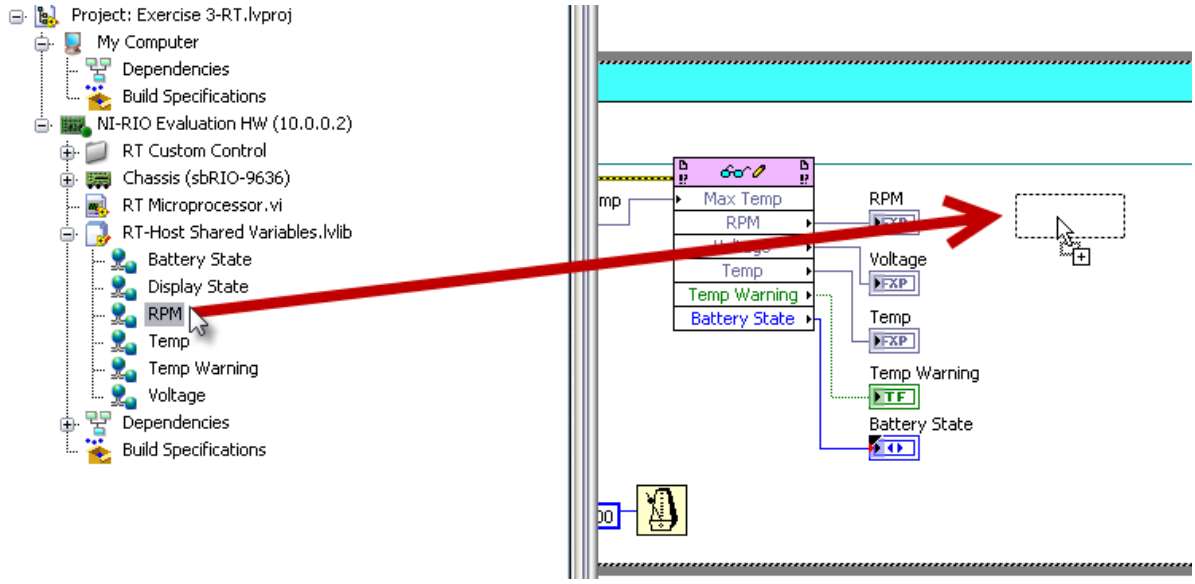
Forward Data onto the Windows Target

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

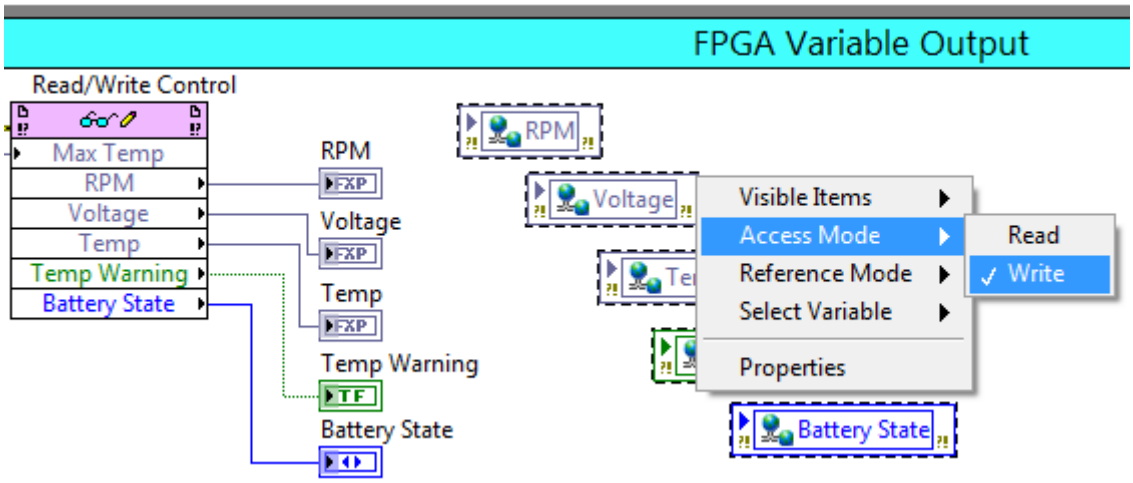


1. Inside the project under the *NI-RIO Evaluation HW* target, locate the **RT-Host Shared Variables.lvlib**. This library contains the **Network-Published Shared Variables** that have been created for communication to the Windows PC.

2. Expand the library (.lvlib) to show the **Network-Published Shared Variables**. Drag an instance of each variable, excluding **Display State**, into the block diagram.

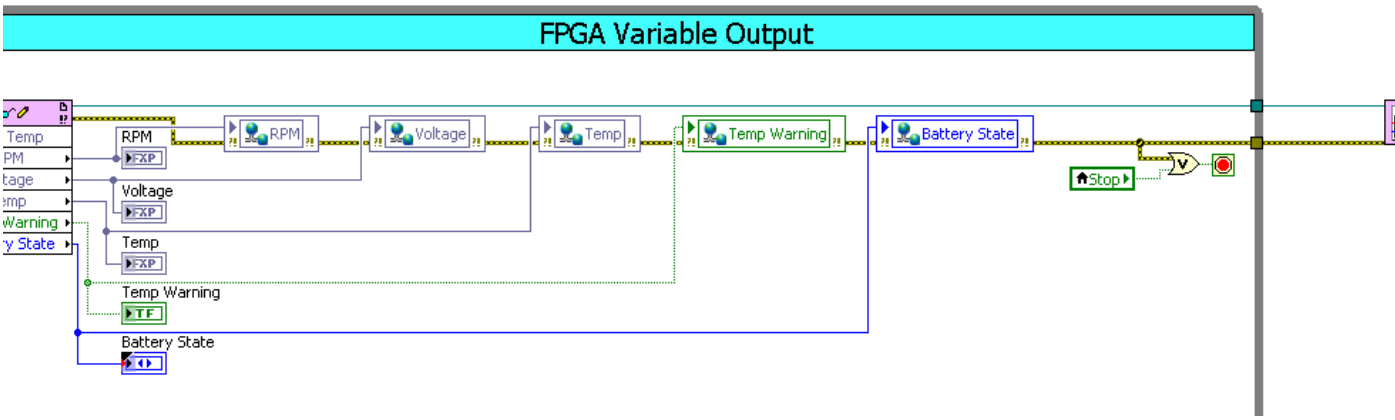


3. By default the shared variable static nodes are in read mode. Change the variables to write access mode by dragging a box around all of them to select them, right-clicking on one, and selecting **Access Mode»Write**.



4. Branch a wire from each of the FPGA **Read/Write Control** outputs to the respective variables as shown in the next screenshot.

5. In order to ensure correct error propagation, wire the **error out** from the **FPGA Read/Write Control** through each of the **Shared Variable** static nodes and then through while loop boundary into the **Close FPGA VI Reference**. Also branch the error wire into the **OR** logic to stop the while loop if there is an error.



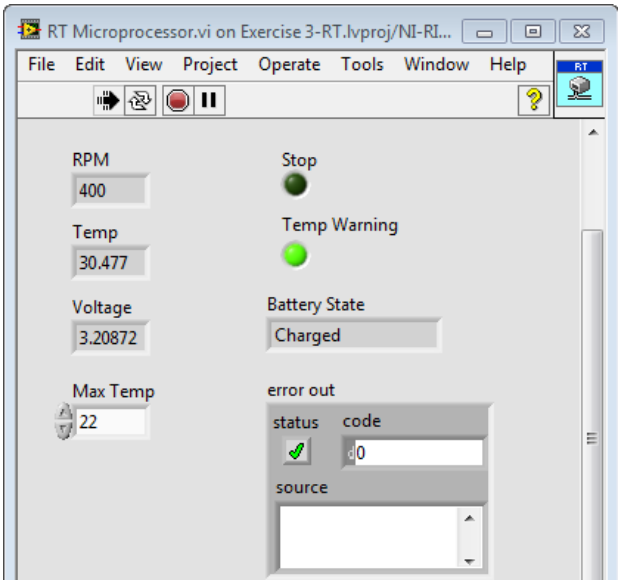
6. Right-click on the **error out** terminal of the **Close FPGA VI Reference** and select **Create»Indicator** in order to display any errors on the front panel.
7. Save the Real-Time VI by selecting **File»Save** or pressing **CTRL+S**.

Test the Real-Time Application

1. Click the **Run** button to deploy the real-time application down to the NI-RIO evaluation device, which also includes the FPGA application since the VI references the compiled FPGA bitfile. Save the VI if prompted.

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

2. 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. This front panel also is simple as it is not intended to be used as the final user interface.



3. When you are finished testing the real-time VI, click the **Abort** button  . We will properly implement a user interface Stop Button in the next exercise.

4. Right-click on the *NI-RIO Evaluation HW* target in the LabVIEW project and select **Disconnect.**

5. 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, and coordinates network communication with the Windows user interface.

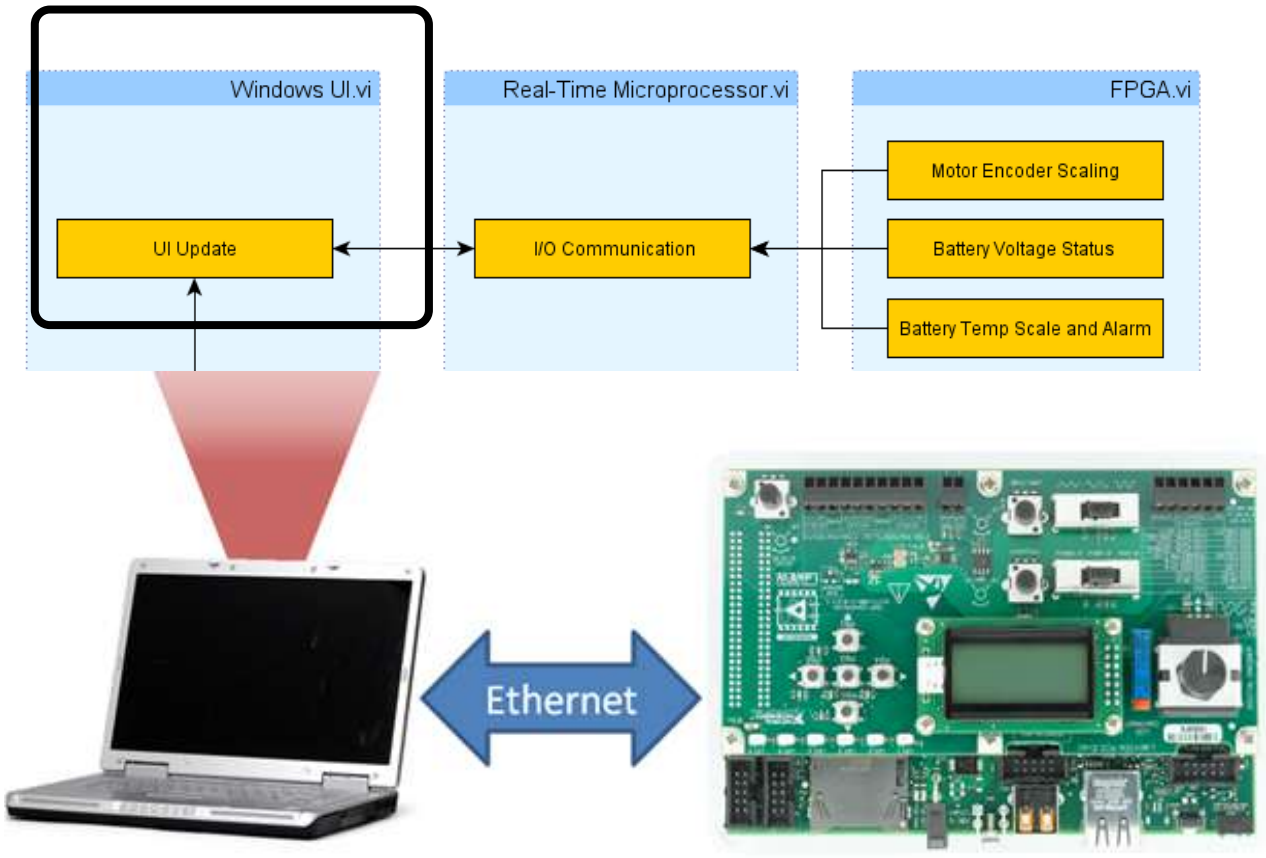
Exercise 4 | Complete System by Interconnecting Targets

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 complete 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. In this exercise, you will complete the following tasks:

- Explore the Windows-Based Application User Interface
- Finish development of the Windows-Based Application
- Run and Verify the Completed System

What am I going to accomplish in this exercise?



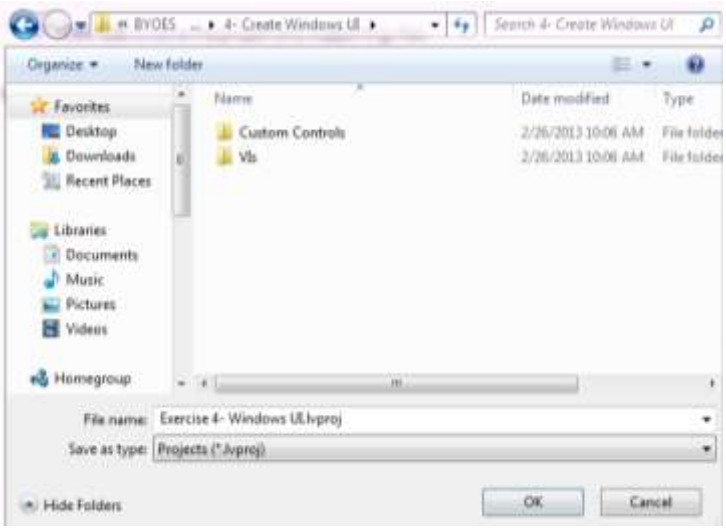
In this exercise you will finish the overall embedded system by completing a user interface running on your Windows development machine. The user interface components have already been placed for you, but the backend communication architecture needs to be completed so that you can update the user interface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

Project Setup

1. With your Exercise 3 project still open in the Project Explorer Window, select **File»Save As…** to save the project in the Exercise 4 folder.

2. In the Save As… dialog box that appears, select **Rename**

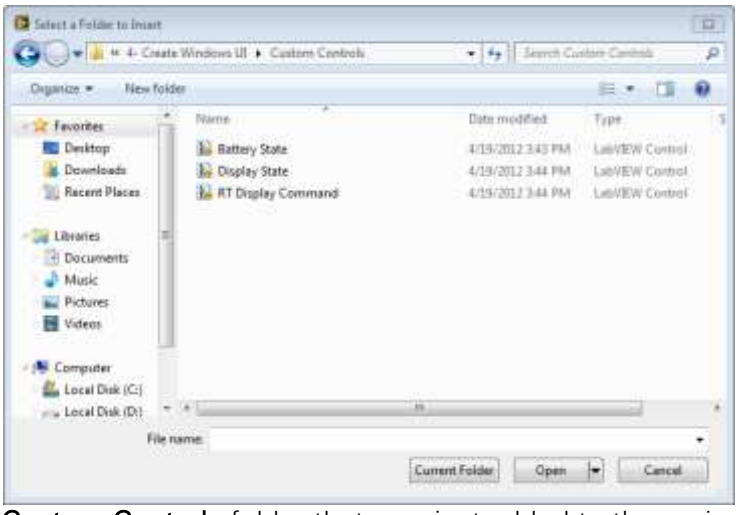
3. Navigate to the …\LabVIEW RIO\4-Create Windows UI\ folder and rename your project as *Exercise 4- Windows UI.lvproj*.



4. Click **OK** to save the project in this new location. Click **OK** if asked to save over existing files.

5. 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 .\BYOES\4-Create Windows UI\VIs folder and select **Windows UI.vi**.

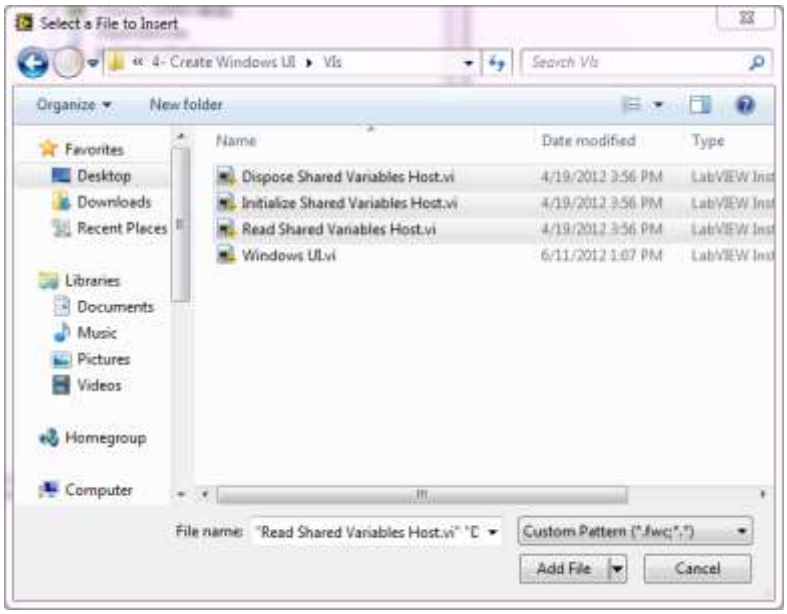
6. Right-click again on *My Computer* and select **Add»Folder (Snapshot)…** Navigate into the .\LabVIEW RIO\4-Create Windows UI\ Custom Controls folder and select **Current Folder**.



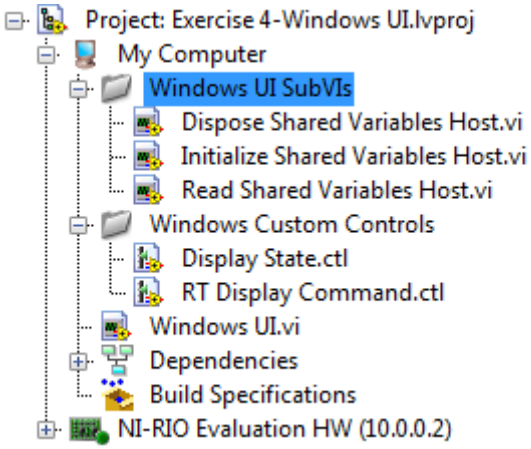
7. Select the **Custom Controls** folder that was just added to the project, press **F2**, and rename it **Windows Custom Controls**.

8. Right-click a final time on *My Computer* and select **New»Virtual Folder**. Name it **Windows UI SubVIs** and then right-click on that virtual folder and select **Add»File…**

9. Navigate to the .\LabVIEW RIO\4-Create Windows UI\VIs folder and hold down the **CTRL** key to select the **Dispose Shared Variables Host.vi**, **Initialize Shared Variables Host.vi**, and **Read Shared Variables Host.vi**. Click on **Add File**.



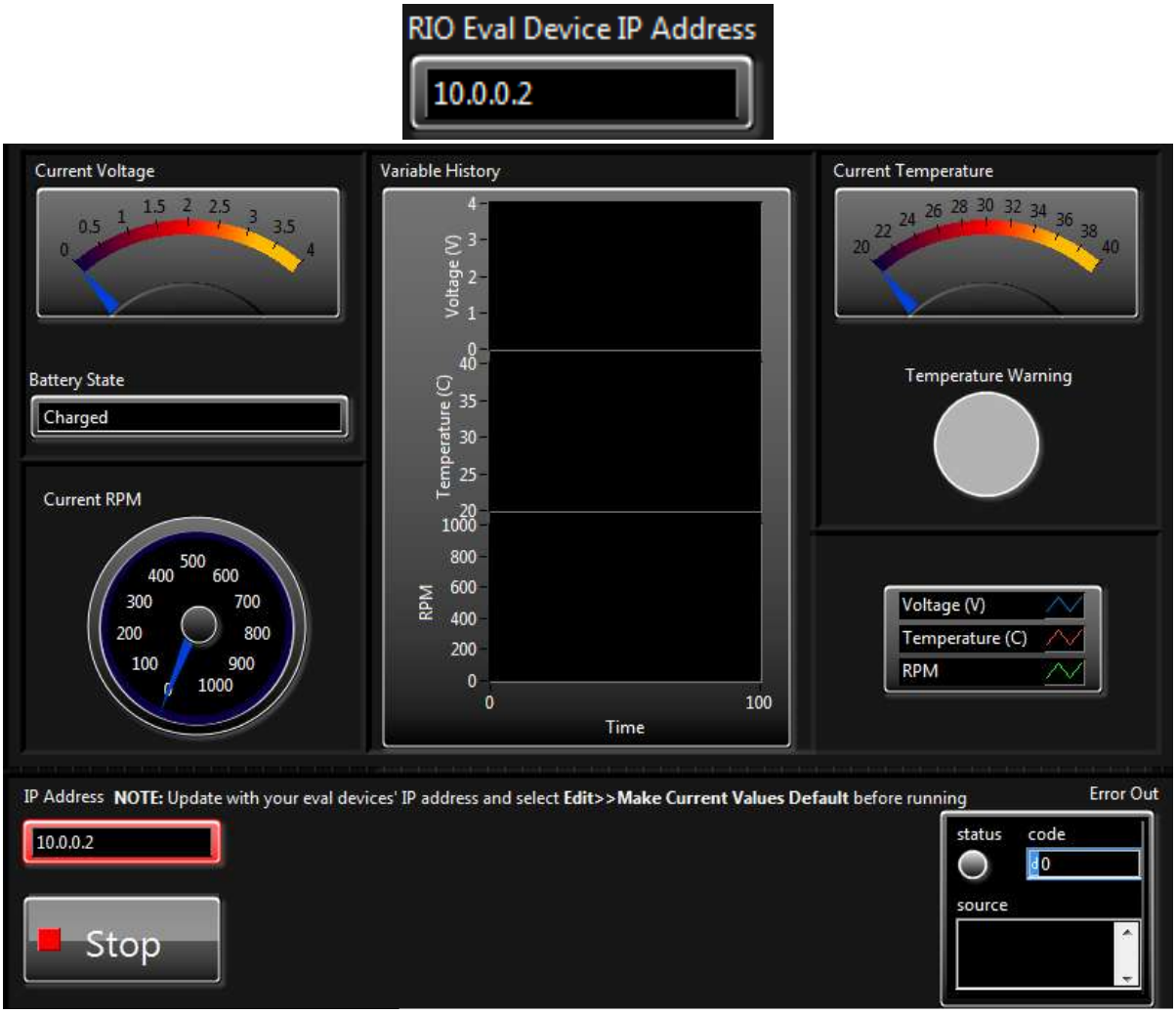
The My Computer section of your project should now look as shown below:



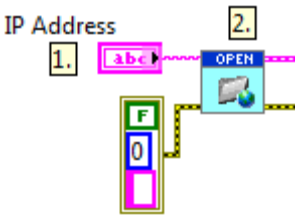
Explore the Windows-Based Application User Interface

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

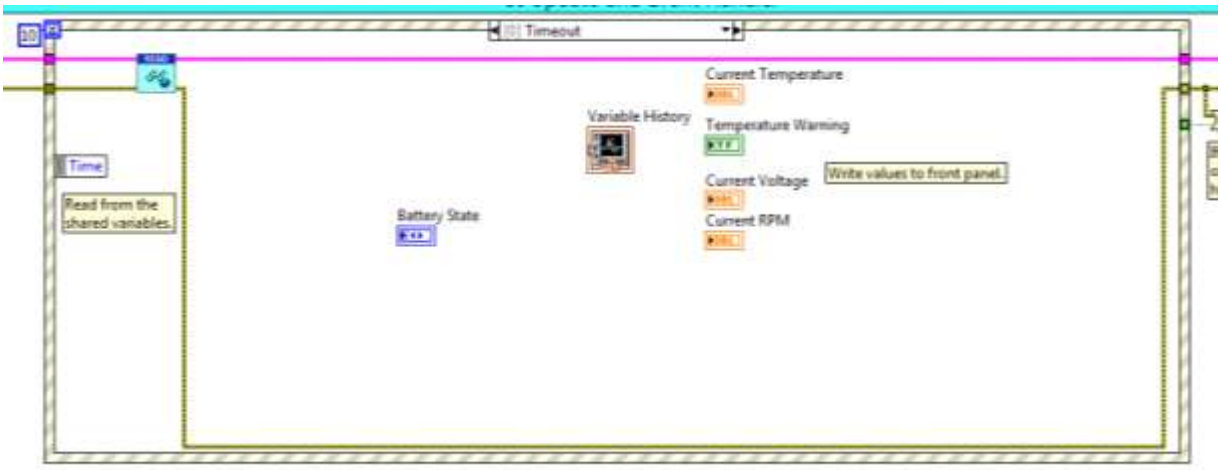
1. In the Project Explorer window, double click to open **Windows UI.vi** to view the User Interface. Note there are indicators for each of the values you've been monitoring on the Real-Time front panel thus far:
   - ✓ Current Voltage, Temperature, RPM, as well as a Variable History of these three values
   - ✓ Temperature Warning alarm Boolean indicator
2. Confirm the **RIO Eval Device IP Address** string value matches the IP address of your target.



3. Press **CTRL+E** to view the block diagram. The IP address of the target is gathered to the left of the loops with the **IP Address** string control and a connection to its shared variable engine 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.

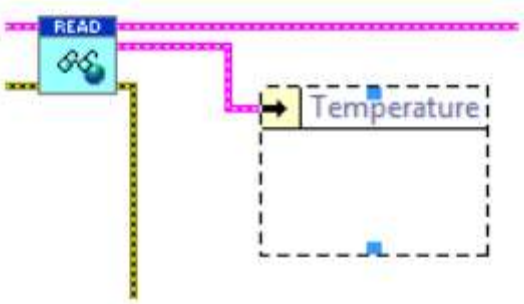


4. Inside the **UI Update and Event Handler** while loop, observe the Timeout case of the event structure. The LabVIEW Event Structure executes when a defined event is detected, in this case, if no event is detected in 10 ms, the timeout case is executed. This timeout case:
   - ✓ Reads the network published 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 update with the most recent values. You will complete this wiring in the next section.

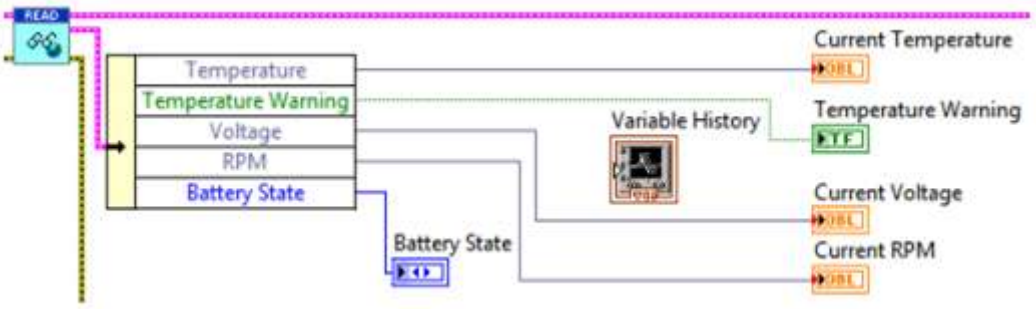


5. Now toggle the event structure to the "**Stop Button**":**Value Change** case by clicking the arrow next to the Timeout case text.
6. Note that the user interface **Stop Button** control is located inside this even tcase and the case will execute when the user interface detects a value change of the 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.

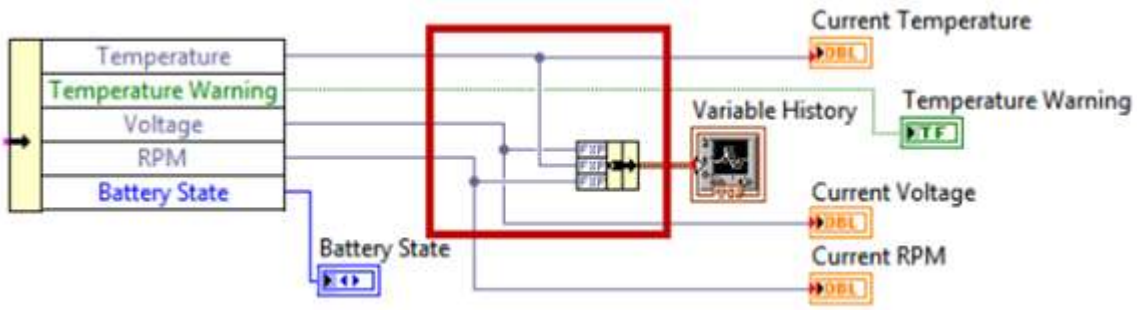
7. In the **UI Update and Event Handler** loop the shared variables that are read need to be unbundled for connection to their respective front panel indicators.
   - ✓ In the Timeout event case, insert an **Unbundle By Name** function (Functions»Programming»Cluster, Class, & Variant) to the right of 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.



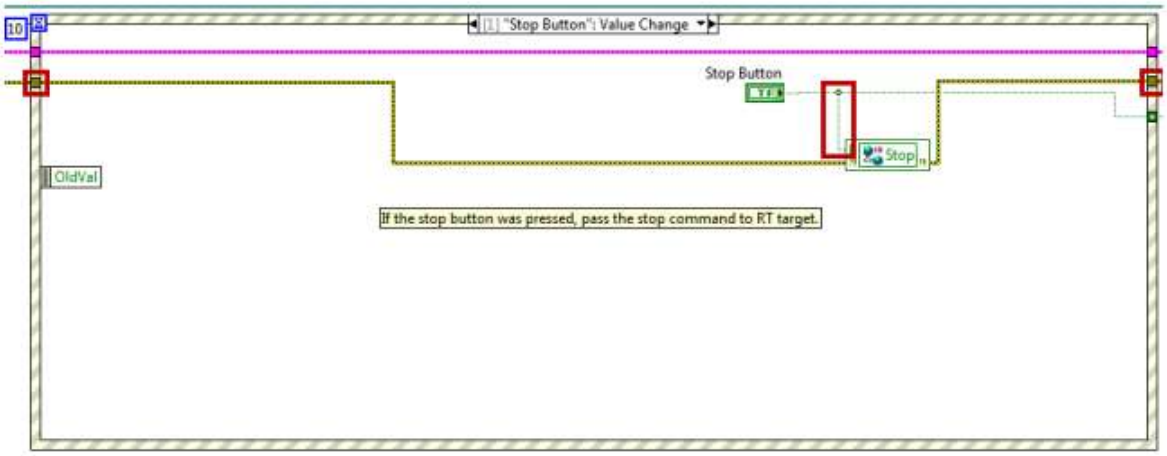
   - ✓ Left-click on each terminal and select the shared variable name that matches the order in the screenshot below.
   - ✓ Wire the output terminals to the matching front panel indicators.



8. 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.



9. Switch the event structure to the "**Stop Button**": **Value Change** case by clicking the arrow next to the Timeout case text.
10. 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.
11. Change the **Stop** shared variable to write by selecting it, right-clicking, and going to **Access Mode»Write**.
12. 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.



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

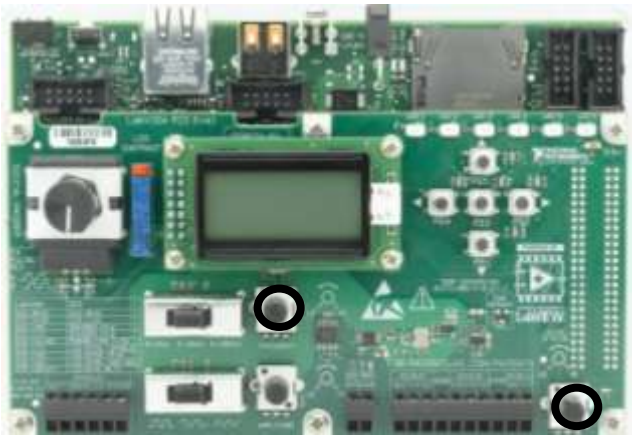
Run and Verify the Completed System

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

1. Open and run **RT Microprocessor.vi**. Values will not update until Step 2.

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

2. Run the **Windows UI.vi** to connect the network streams and see the values update.
3. On the daughter card, turn the potentiometer.
   - ✓ Verify that the battery voltage changes on the Windows User Interface.
   - ✓ 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 turn the frequency potentiometer for the function generator and verify that the theoretical RPM changes on the Windows User Interface



5. Once you are done testing, click the **Stop Button** 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 Explorer window and select **Disconnect**. Note the bright green Boolean turns off, indicating the target is disconnected.

Congratulations, you have finished the development of your RIO-based embedded system that includes a Windows user interface, a real-time application running on a microprocessor, and an FPGA interfacing with the system I/O!

Order your own NI LabVIEW RIO Evaluation Kit

Experience Embedded System Development With LabVIEW

- Extended evaluation version of LabVIEW, LabVIEW Real-Time Module, and LabVIEW FPGA Module
- Included embedded hardware target with RTOS processor, reconfigurable FPGA, analog and digital I/O
- Onboard LCD, function generator, potentiometer, LED, and temperature sensor I/O
- Step-by-step tutorials for building embedded FPGA and processor-based applications
- Setup wizard and fully documented, ready-to-run examples of common embedded tasks

Overview

The NI LabVIEW RIO Evaluation Kit includes all you need to experience the NI graphical system design approach for embedded systems. This approach combines LabVIEW system design software and a standard FPGA-based NI reconfigurable I/O (RIO) hardware platform to reduce time to market for embedded control and monitoring applications.

The kit includes a 90-day extended evaluation of the LabVIEW FPGA and LabVIEW Real-Time modules; an NI RIO hardware evaluation device; a daughter board for easy I/O interfacing; a step-by-step tutorial; and numerous fully documented, ready-to-run examples of common embedded tasks implemented in LabVIEW.

Use the evaluation kit to learn more about the NI LabVIEW RIO architecture, which features LabVIEW system design software and NI RIO hardware platforms including NI CompactRIO, NI Single-Board RIO, and NI R Series devices.

The LabVIEW RIO Evaluation Kit is intended to assist in evaluating the platform. You need to purchase a nonevaluation NI hardware system to start developing your application. Because the evaluation kits are heavily subsidized so you can evaluate LabVIEW programming on an NI RIO hardware device, orders are limited to one per customer.

Do not purchase this kit for the Build Your Own Embedded System Workshop. You will be provided with purchasing information after registering for the event.