



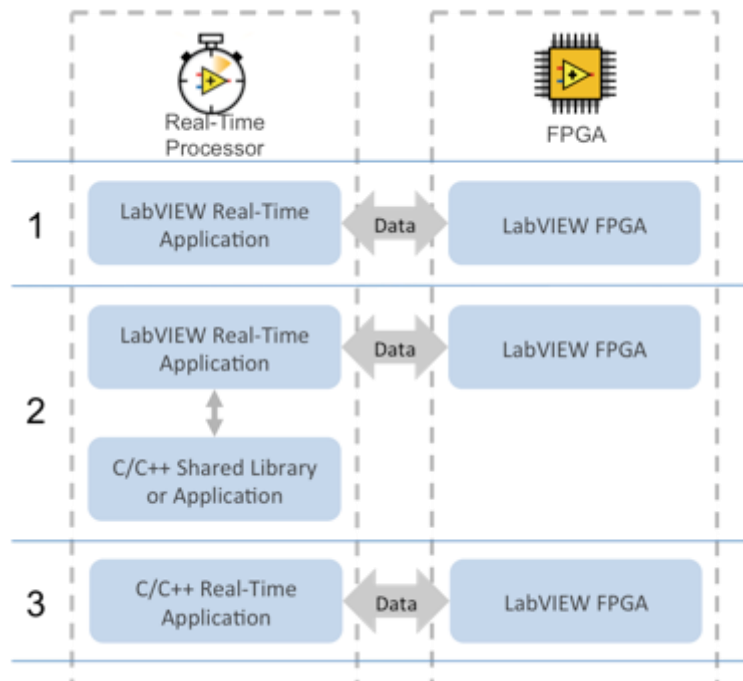
NI LabVIEW RIO Evaluation Kit

Tutorial

NI LabVIEW RIO Evaluation Kit Tutorial

This document contains step-by-step instructions for experiencing embedded system design using the NI LabVIEW system design software with NI reconfigurable I/O (RIO) hardware architecture to create an embedded control and monitoring system.

It is possible to program NI hardware using LabVIEW, C/C++, or a combination of both, as shown below. This tutorial will cover a full LabVIEW application and calling a C shared library from your LabVIEW Real-Time code (scenarios 1 and 2 below). There is a second tutorial, *NI LabVIEW RIO Evaluation Kit C++ Tutorial.pdf*, installed by the DVD included with this kit, that walks through programming the real-time application entirely in C++ (scenario 3), which can be found on disk in the same folder as this LabVIEW tutorial.



The ideal next step prototyping platform is [NI CompactRIO](http://ni.com/compactrio), built with the same architecture as this kit but with modular I/O. The programming experience and APIs learned in this evaluation experience are the same, in fact, you can even reuse the code you've built! This LabVIEW RIO architecture covers a wide range of hardware products; visit ni.com/embedded-systems 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

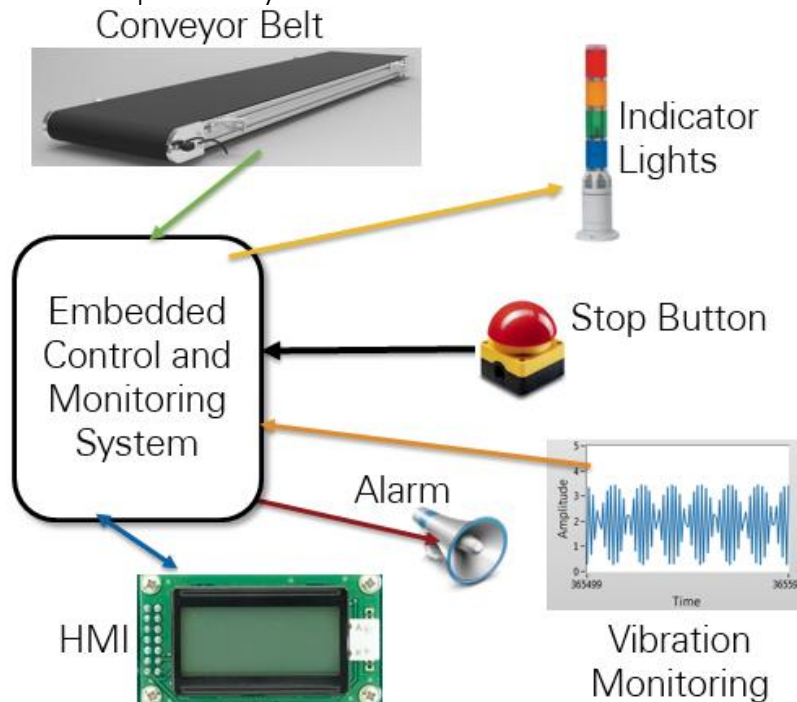
Tutorial Overview	4
Navigating the Exercises	10
Using the Solutions	10
Troubleshooting	10
Activating LabVIEW	11
Getting Started – LabVIEW Programming Basics	11
Initial System Configuration	13
Exercise 1 – Open and Run Application	16
Exercise 2 – Create a Monitoring and Control FPGA Application	21
Test the FPGA Application	32
Exercise 3 – Develop Real-Time Application	33
Test the Real-Time Application	47
Optional- Call existing C library from LabVIEW RT VI	49
Exercise 4 – Create a Windows User Interface	55
Run and Verify the Completed System	60
Exercise 5 – Startup Application	61
Appendix A – LabVIEW RIO Training Path	65
Appendix B - Changing the IP Address in the LabVIEW Project	68

Tutorial Overview

In this tutorial, you will complete five exercises that demonstrate how to develop an embedded system using LabVIEW system design software with 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 the embedded control and monitoring system for a part inspection machine scenario, similar to what is shown below. This machine carries the product through the inspection process and controls a variety of inspection aspects.



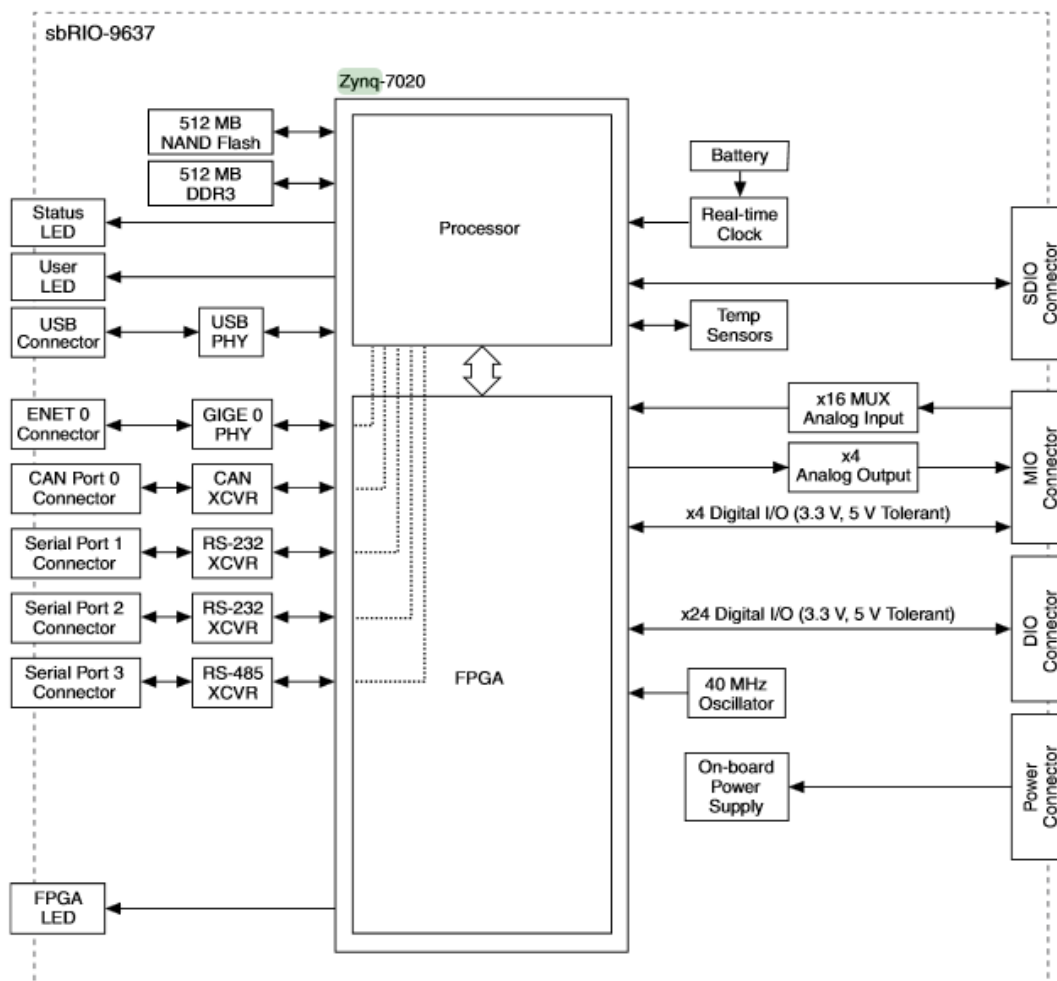
Here are the various components you will interact with in this scenario:



System Specifications

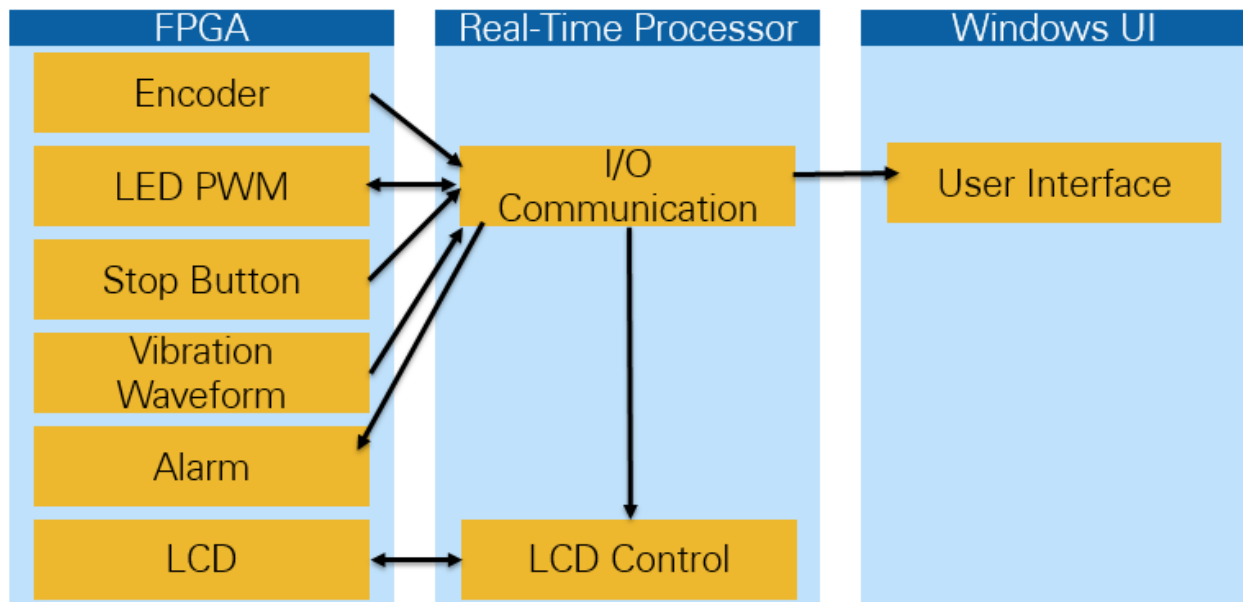
Your kit is based on a standard hardware architecture that includes a [Xilinx Zynq-7000 XC7Z020 All Programmable System on Chip](#), with an Artix-7 FPGA and a 667 MHz dual-core ARM Cortex-A9 processor running the [NI Linux Real-Time](#) (32 bit) distribution, and I/O available through the daughter card. The system you are developing in this tutorial will include both the embedded target and your Windows PC. It can be programmed using a single development tool chain, LabVIEW, or you can program the processor with C/C++; this kit includes the necessary cross-compiler and the Eclipse IDE. 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.

Block Diagram



System Diagram

Since there is some flexibility with three available targets to run code, the part inspection tasks have been mapped to processes on each of the targets (Windows User Interface, Real-Time processor, and FPGA). An FPGA (Field Programmable Gate Array) is basically software-defined hardware that can be reconfigured multiple times to create a custom circuit. All the system I/O goes through the FPGA and you can achieve very fast and reliable control and acquisition.



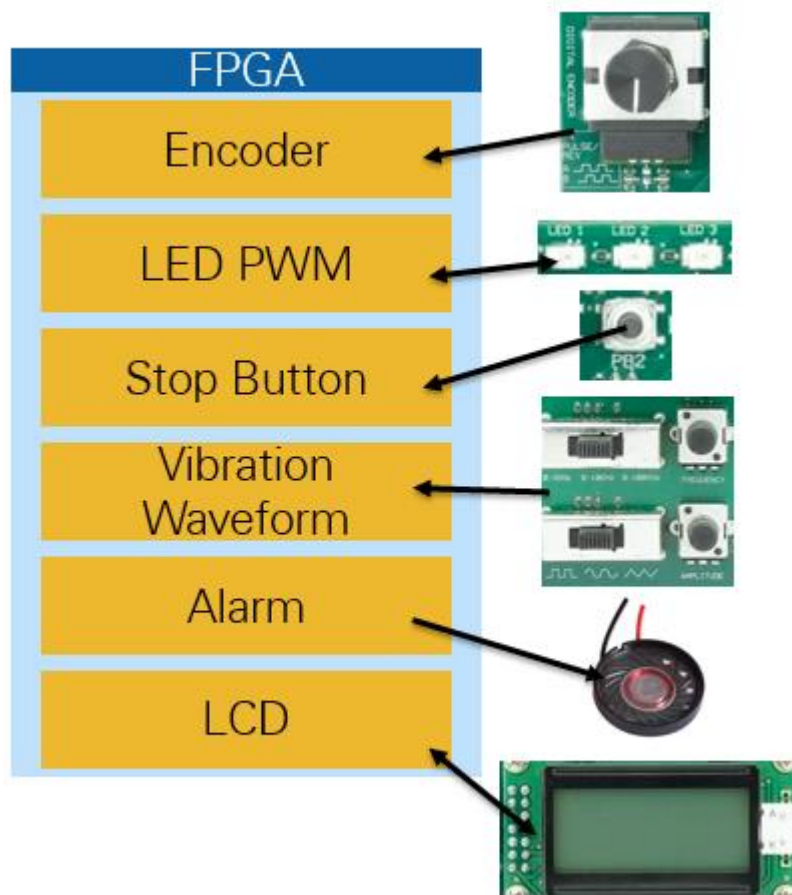
Monitoring Tasks

1. Vibration monitoring

Control Tasks

1. Encoder
2. LCD
3. Stop button
4. Alarm
5. Indicator LEDs

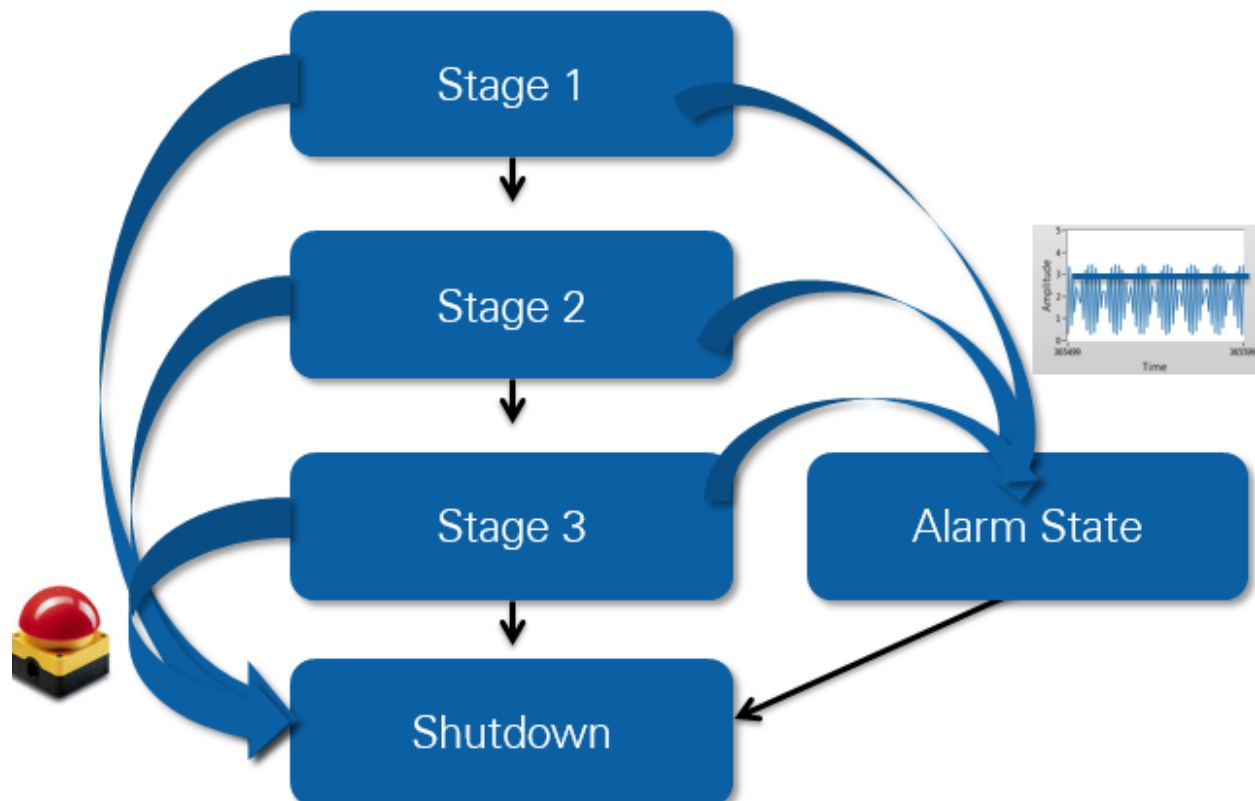
Finally, since you do not have an actual part inspection system to control and monitor here is how you will simulate various elements using the onboard I/O, connected through the FPGA of your evaluation device:



State Diagram

Normal machine operation will progress from stage 1 » 2 » 3 then proper shutdown. If an alarm condition is tripped, when the machine's vibration waveform amplitude crosses a threshold, you will enter the alarm state and automatically progress to proper shutdown. If you press the Stop button on your board (PB2) you will also automatically progress to proper shutdown.

For example, when the amplitude of the machine's vibration waveform exceeds a preset threshold, it could mean a damaged bearing that a technician should investigate.



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 Application

Explore and run a precompiled application that simulates precise intensity control of two lasers, pulsed at a high frequency. The FPGA implements a pair of high frequency and reliable pulse width modulation (PWM) channels. The RT controller hosts a simple user interface (UI) to control the lasers.

Exercise 2: Create a Monitoring and Control FPGA Application

Create an FPGA application on your own to control and interact with your I/O for the part inspection machine. While this application is compiling, start on Exercise 3.

Note: For this exercise you will either need internet access to compile your LabVIEW FPGA code with the [NI LabVIEW FPGA Compile Cloud Service](#) (required if using Windows 8) or you will need to install the LabVIEW 2015 FPGA Module Xilinx Vivado 2014.4 Compile Tools to your computer from the second DVD in your kit accessories box or from [ni.com](#).

Exercise 3: Develop Real-Time Application

Design a real-time application running on the processor which communicates with the FPGA to exchange data and control the LCD. This code is based on a State Machine Producer/Consumer architecture.

Exercise 4: Create a Windows User Interface

Extend the embedded system to include a user interface running on a Windows computer communicating over the network to display the current status of your part inspection system.

Exercise 5: Create a Standalone Startup Application

Now that your system is complete, learn how to deploy the system to run standalone and startup automatically at system reboot.

After completing these exercises, ask questions and explore a variety of LabVIEW Real-Time and LabVIEW FPGA examples and getting started resources built for the kit on an online community for LabVIEW RIO Evaluation Kit users at [ni.com/rioeval/nextstep](#).

For additional details on the NI Linux Real-Time distribution and tutorials for interacting with third party packages and C/C++ code, visit [ni.com/linuxrtforum](#).

Navigating the Exercises

The LabVIEW RIO Evaluation Kit DVD installs the exercises on your development machine. By default, these exercises install on your computer's hard drive at:

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

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

This is the directory structure for each of the exercises, and you will find additional resources linking to ni.com.



Using the Solutions

If for any reason you are not able to complete an exercise successfully feel free to open the solution to review and test and/or use it to continue on to the next exercise. You will need to modify the solution LabVIEW project with your device's IP address if using Ethernet. Reference **Appendix B** for more details on this process.

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 document installed at **All Programs»National Instruments»LabVIEW RIO Evaluation Kit** or 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 installation questions. If that document doesn't cover your question, please post it to the LabVIEW RIO Evaluation Kit user community (ni.com/rioeval/nextstep) or contact NI Support (ni.com/ask).

Activating LabVIEW

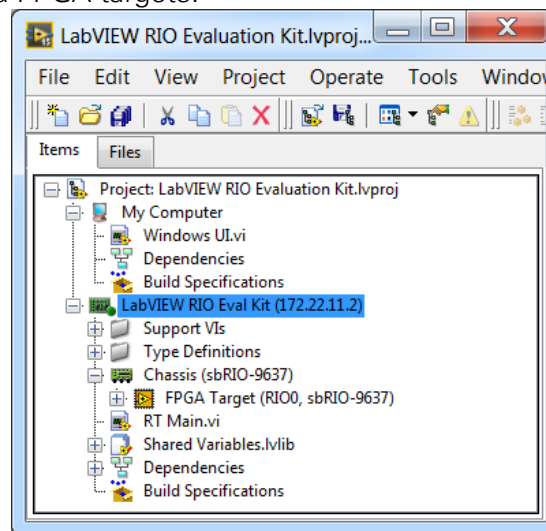
The evaluation kit installs a 90-day full evaluation version of LabVIEW; you will be prompted to activate LabVIEW when you open the environment. To continue in evaluation mode, click the “Launch LabVIEW” button.

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. 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»LabVIEW RIO Evaluation Kit Setup Utility**.

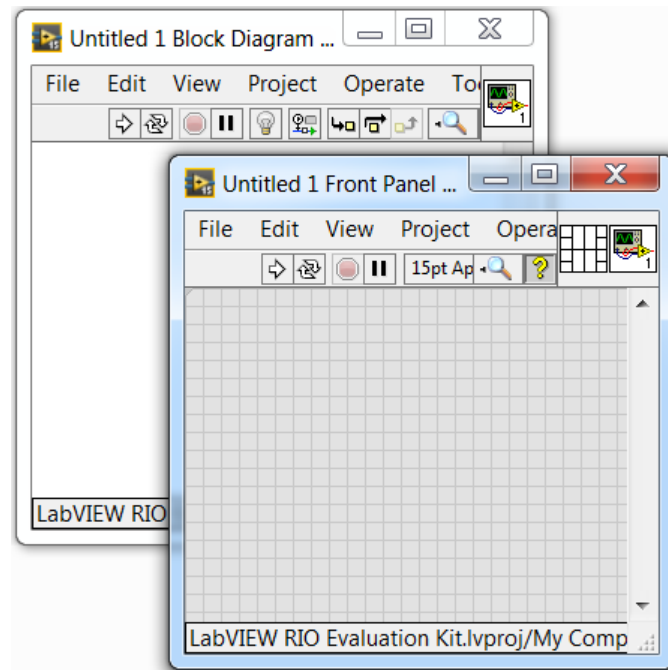
A LabVIEW application is called a “VI”, or virtual instrument, and is composed of two primary elements: a front panel and a block diagram, which you can program using the LabVIEW Functions Palette.

- **Project Explorer** – The Project Explorer window provides a system-level view of the files and VIs in your application and allows you to manage code and build specifications for desktop, real-time, and FPGA targets.



- **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.



- **Controls Palette** – The Controls palette contains user interface components for creating front panels. To create your user interface, drag the components onto the front panel and wire them on the block diagram to the data you want to display or control.
- **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 primitive blocks and functions for creating FPGA, RT, and Windows applications. Drag the components onto 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 to create a dataflow program.

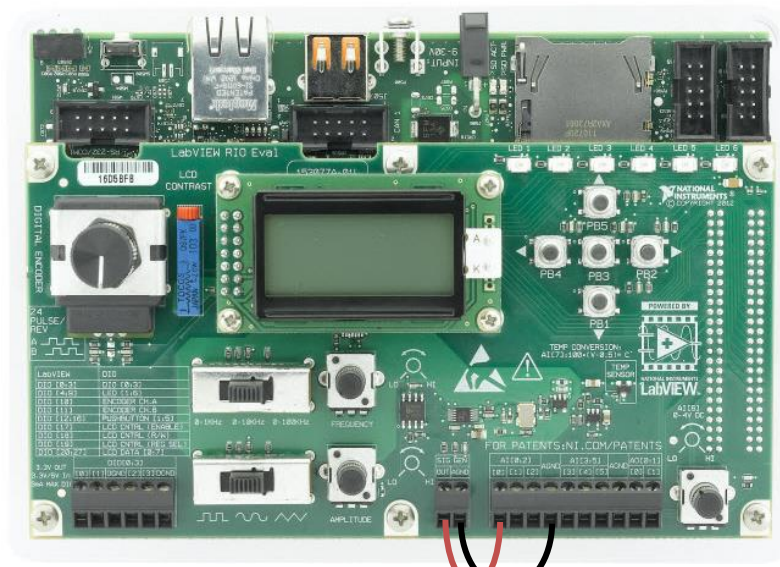
Using the functions palette – In this tutorial, **bold** text denotes an item to select from the Functions palette. To access the Function palette, right-click anywhere on the LabVIEW block diagram. You can also “pin” the functions palette (in the upper left corner of its window) so that it is always present on the block diagram.

More on Using LabVIEW - To learn more about the LabVIEW graphical programming environment including syntax, deployment, debugging, and more, we *strongly recommend* you reference <http://www.ni.com/gettingstarted/labviewbasics/>.

Initial System Configuration

LabVIEW RIO Evaluation Hardware

1. If you haven't already, complete the **Setup Wizard** located at **Start»All Programs»National Instruments»LabVIEW RIO Evaluation Kit»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 (ground) terminals as shown below.



3. In preparation for application development verify that the function generator is set to **0-10KHz** (center) and **Sine Wave** generation (center) settings.

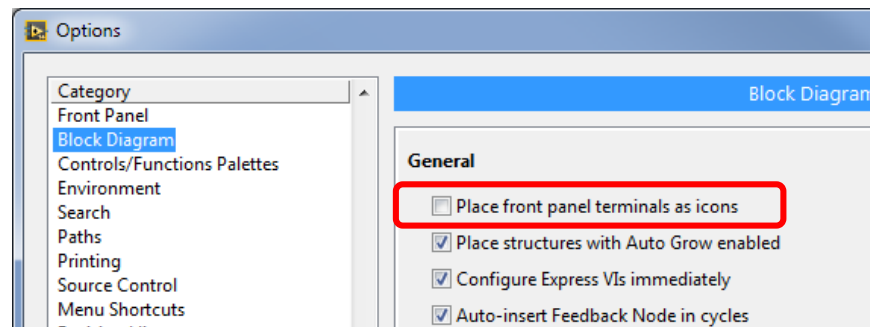


4. Connect the Speaker, red wire to AO0 and black wire to AGND as shown below.

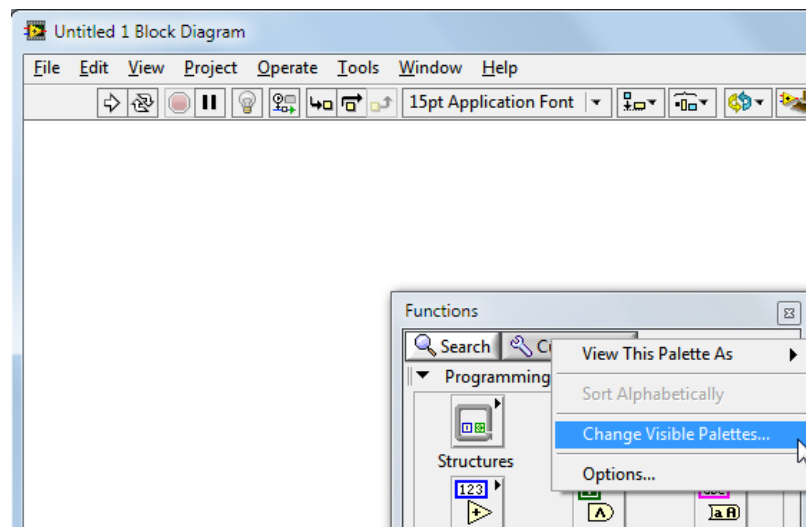


LabVIEW Environment

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 for new UI controls and indicators you create.

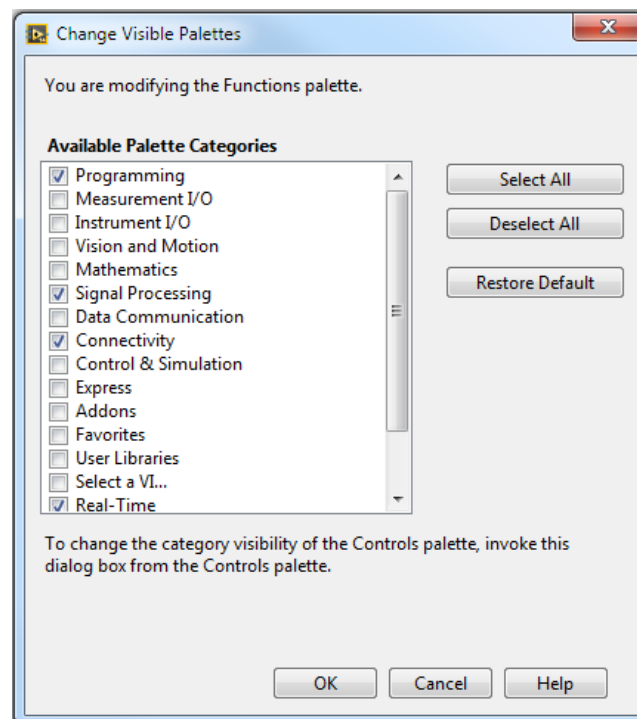


2. Create a new application (called a VI, or Virtual Instrument) by selecting **File»New VI**. Click on the white Block Diagram window to bring it to the front and right-click within it to make the *Functions* palette appear. This is where you can find the programming nodes and functions to write your code. Pin down the *Functions* palette (upper left corner of its window) and click on **Customize»Change Visible Palettes...**

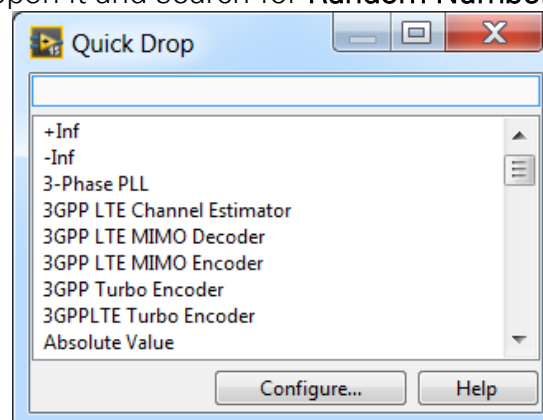


- Click **Deselect All** and then check the following palettes that you will use in this tutorial. This will simplify what you see when you open the *Functions* palette. Once you are done click **OK**.

Palettes Needed
 Programming
 Signal Processing
 Connectivity
 Real-Time
 FPGA Interface



- Another way to search for functions by name is using the **Quick Drop** menu. Press <Ctrl-Spacebar> to open it and search for **Random Number**, which it will find.

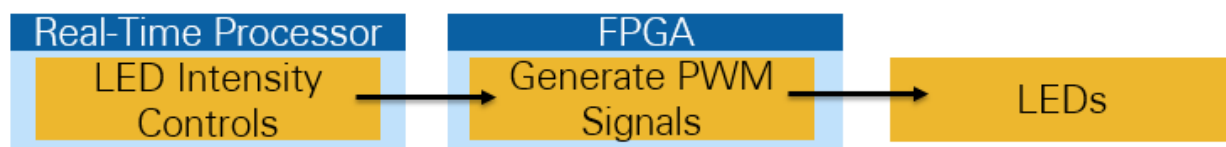


- Close the **Untitled 1.vi** without saving it.

Exercise 1 | Open and Run Application

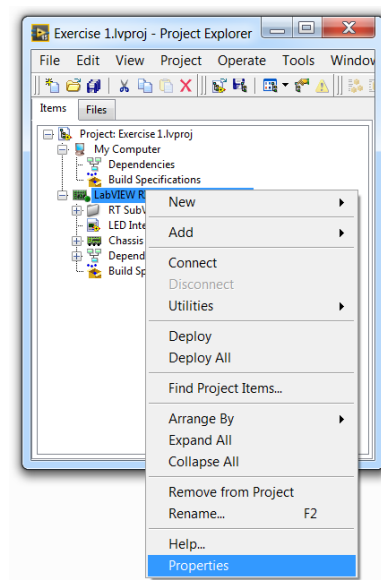
Summary

In this exercise you will learn how to navigate the LabVIEW Project with your evaluation kit, then open and run a pre-compiled application with code running on both the FPGA and RT Processor. This application simulates precise intensity control of two lasers (LEDs), which need to be pulsed at a high frequency to prevent damage of the part under inspection. The FPGA is used to implement a pair of high frequency and reliable pulse width modulation (PWM) channels. The real-time controller hosts a simple user interface (UI) for control of the lasers. This UI allows the user to set the precise intensity of each laser independently, or to initiate automatic and coordinated laser pulses at a user defined interval.



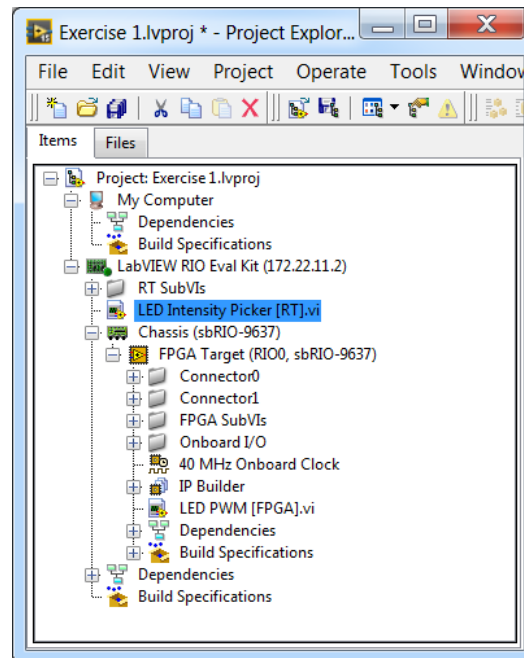
Explore the Application

1. Open up the Exercise 1 LabVIEW project file by navigating to `.\1- Open and Run \Exercise 1.lvproj`.
2. Ensure the IP address of the *NI-RIO Evaluation HW* target in the Project Explorer window to match the IP address of your evaluation board.
 - a. Right-click the *LabVIEW RIO Eval Kit* target 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 device IP address and click **OK**.

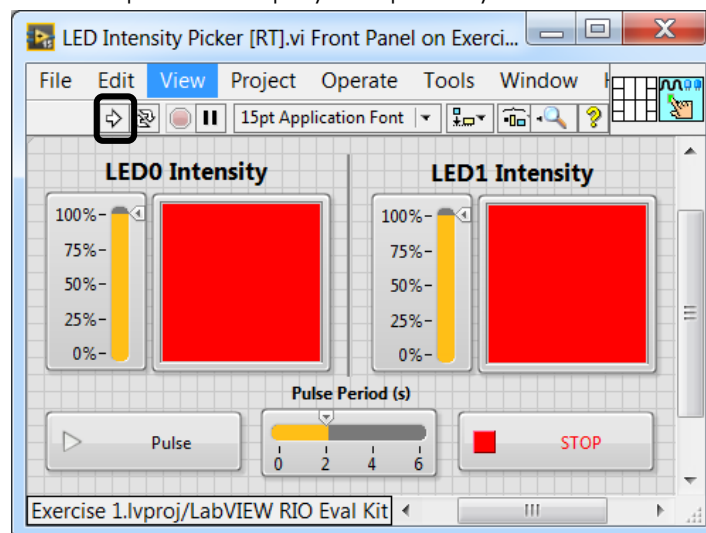


Note: If you forgot to write down the IP address from the setup utility and need to determine the IP address of your device, see *Appendix B - Changing the IP Address* in the LabVIEW Project at the end of this tutorial for further instructions. The default IP address when connected over USB is 172.22.11.2.

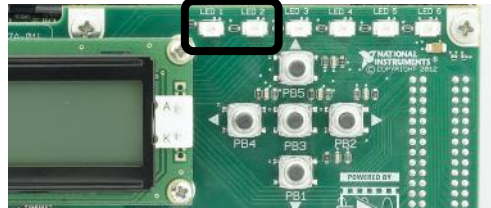
3. In the project, expand out the *LabVIEW RIO Eval Kit* target, Chassis, and then the FPGA Target. Note there are separate VIs for the Real-Time processor and the FPGA. Double-click on **LED Intensity Picker [RT].vi** to open it.



4. Press the **run arrow** to start **LED Intensity Picker [RT].vi**. Select **Save** through any dialogue prompts. Once you kick off the deployment, a dialog box will appear showing the current process and then will deploy the application down to the processor on your target via Ethernet or USB. In this case, a pre-compiled FPGA bitfile is called within the real-time VI so it is not required to deploy it separately.



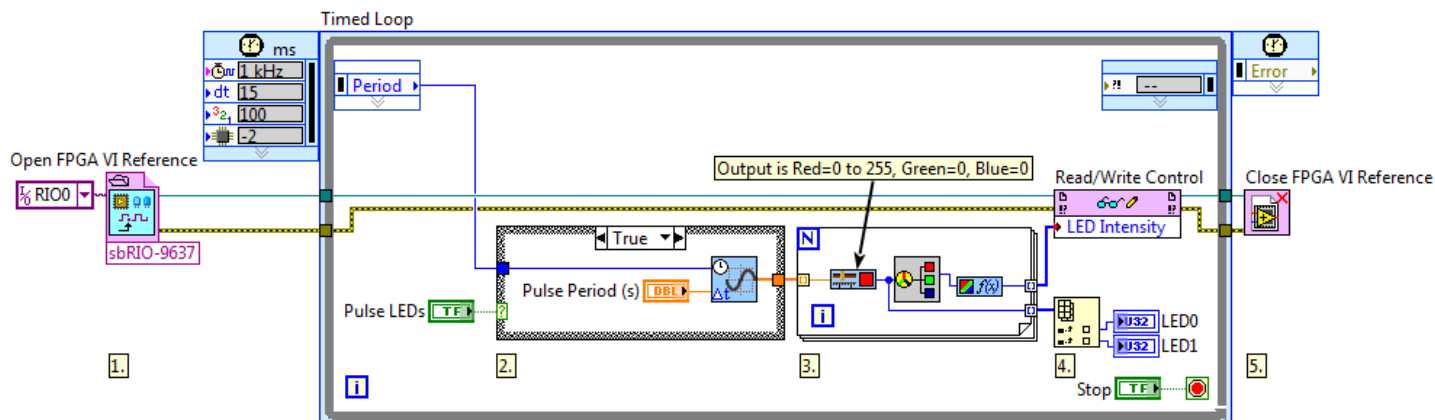
The LabVIEW Real-Time application interfaces with the FPGA application and hosts a simple UI. This VI will allow you to manually set the intensity of each LED or generate a sine wave for each LED so that they pulse automatically. To manually set the intensities, first make sure the Pulse control is off, then use the slide controls to the left of each LED intensity indicator. Notice LED 0 and LED 1 light up



To generate a sine wave that drives each LED to pulse automatically, turn on the Pulse control by pressing the **Pulse** button and then set the Pulse Period (s).



5. Press the **Stop** button on the front panel.
6. Open the block diagram by pressing <Ctrl-E> or navigating to **Window»Show Block Diagram** review the code. Press <Ctrl-H> to open the **Context Help** window, which gives details about the code your cursor interacts with.

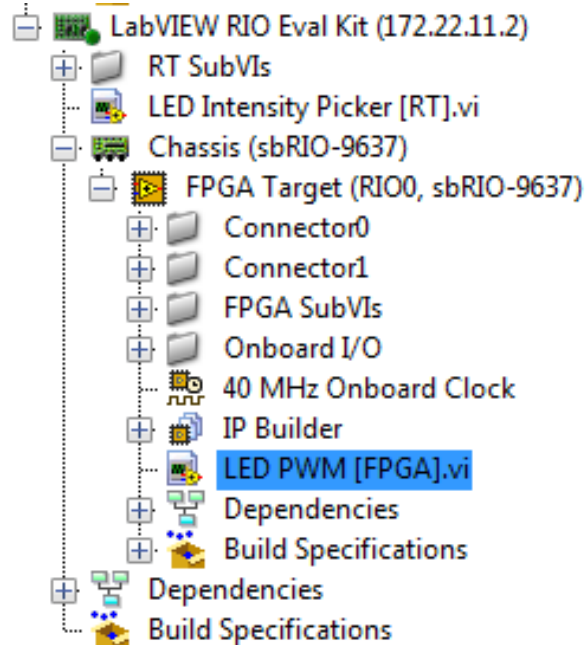


This VI first opens a reference to a pre-compiled bitfile deployed your FPGA Target (sbRIO-9637, RIO0), and then runs a **Timed Loop** to update the PWM channel's duty cycle (which translates into LED intensity). The timed loop iterates at a set rate defined in the node on the left-hand side. The user can adjust the intensities manually by percent, or choose the automatic pulse mode where the LEDs are pulsed at a user-defined period. In automatic mode, the pulses are modeled on a sine wave where the LEDs are 180 degrees out of phase.

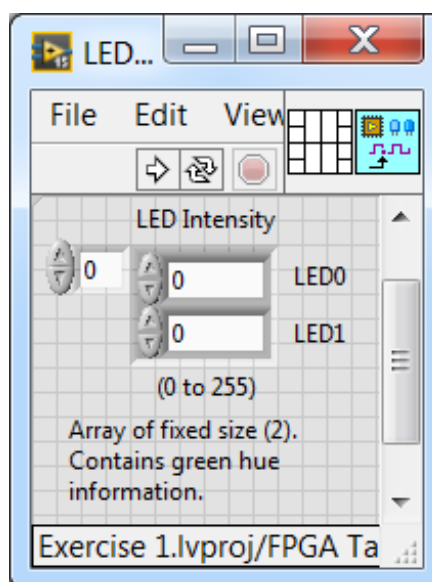
Then the application implements custom logarithmic scaling so that the change in intensity appears linear to the human eye (since humans do not see changes in light intensity on a linear scale). Finally, the **FPGA Read/Write Control** node to communicate

the intensity values to the FPGA. The LEDs on your kit are only one color, but this code is portable if you plan to manipulate multicolor LEDs in the future.

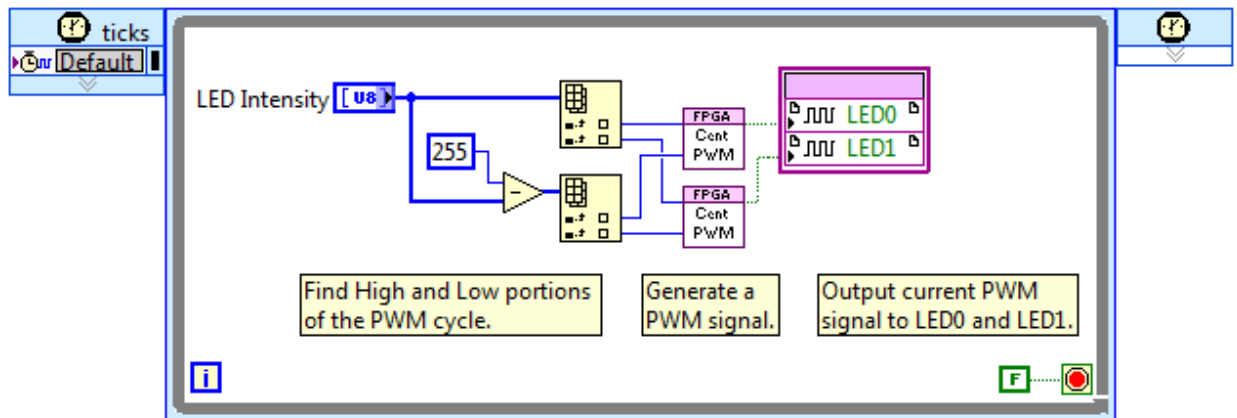
7. Close LED Intensity Picker [RT].vi and open LED PWM [FPGA].vi



8. 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 *LED Intensity* array control represents an FPGA register that is accessible for communication between the FPGA and the real-time processor, and in this case receives the intensity values coming from the LED Intensity Picker [RT] vi read/write control.

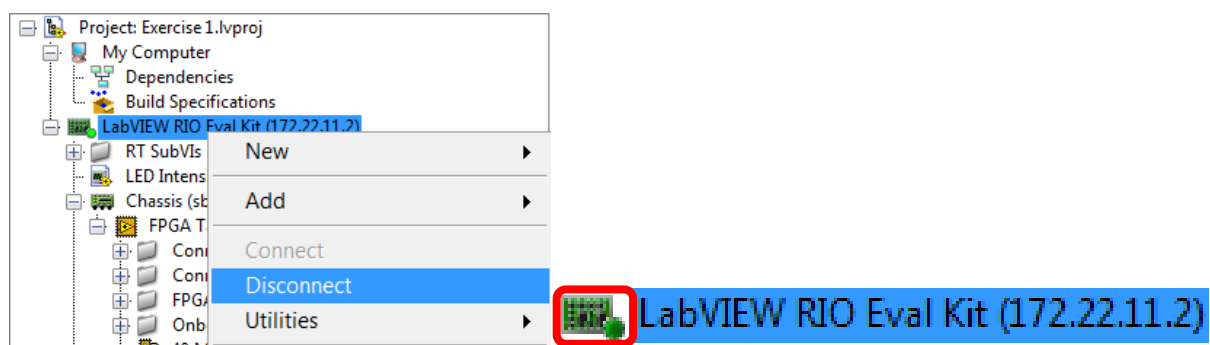


9. Open the block diagram by pressing <Ctrl-E>.



This LabVIEW FPGA code implements a pair of simple PWM channels. They are controlled independently, but are synchronized using LabVIEW Real-Time. Because they are implemented in hardware on the FPGA, the PWM channels are highly reliable and execute with virtually no latency. Placing the channels in a Timed Loop on the FPGA implements the code within one tick of the default hardware clock (40 MHz), ensuring they execute at a high frequency. The PWM channels change the intensities of the onboard LEDs on your kit labeled **LED0** and **LED1**.

10. Double click on one of the **Center Aligned PWM Out- SCTL** sub-VIs to open and see the PWM logic. A center-aligned PWM waveform is symmetric, aligning PWM signals to a reference point, such that half of the PWM signal occurs before the reference point and the remaining half of the signal occurs after the reference point.
11. When you are finished exploring the project, right-click on the *LabVIEW RIO Eval Kit* target in the LabVIEW project and select **Disconnect**. Note the bright green Boolean turns off, indicating the target is disconnected.



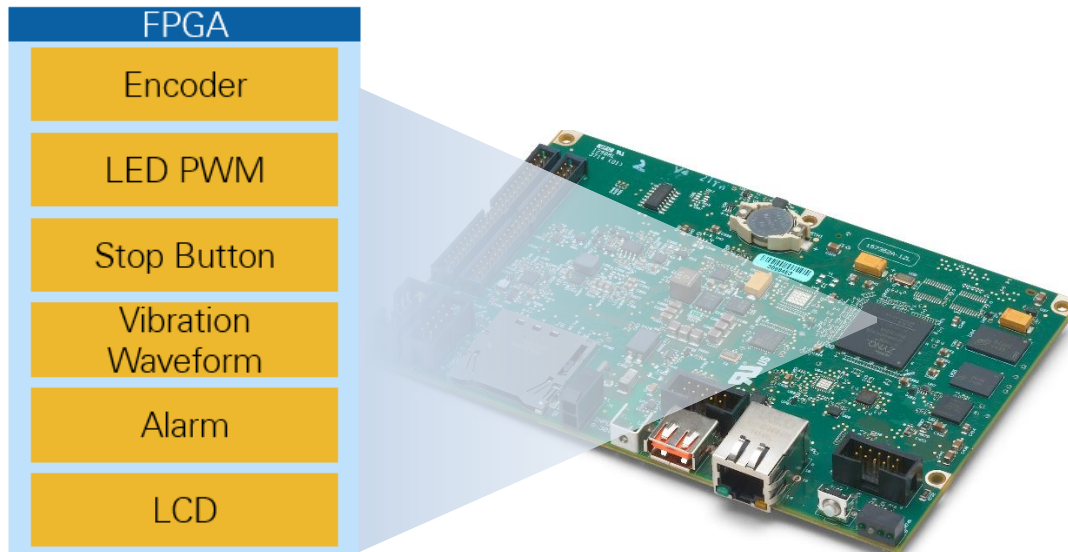
12. Finally, close any open LabVIEW files, and do not save the files if prompted.

Tip: Only one project can connect to your target at a time. 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 and clear the error, press the reset button next to the Ethernet port on the device to reboot it.

Exercise 2 | Create a Monitoring and Control FPGA Application

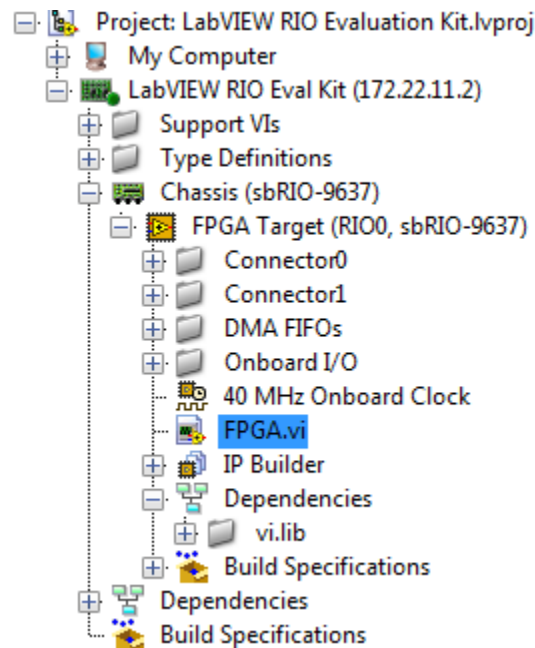
Summary

Learn how to create and control I/O on your kit's daughter board through the FPGA. The FPGA code controls the conveyor belt and measures sensor signals. You will complete the logic then compile it to run on the FPGA. You will interact with the following I/O on the board: Quadrature Encoder, LCD, push buttons, LEDs, amplitude and frequency knobs.

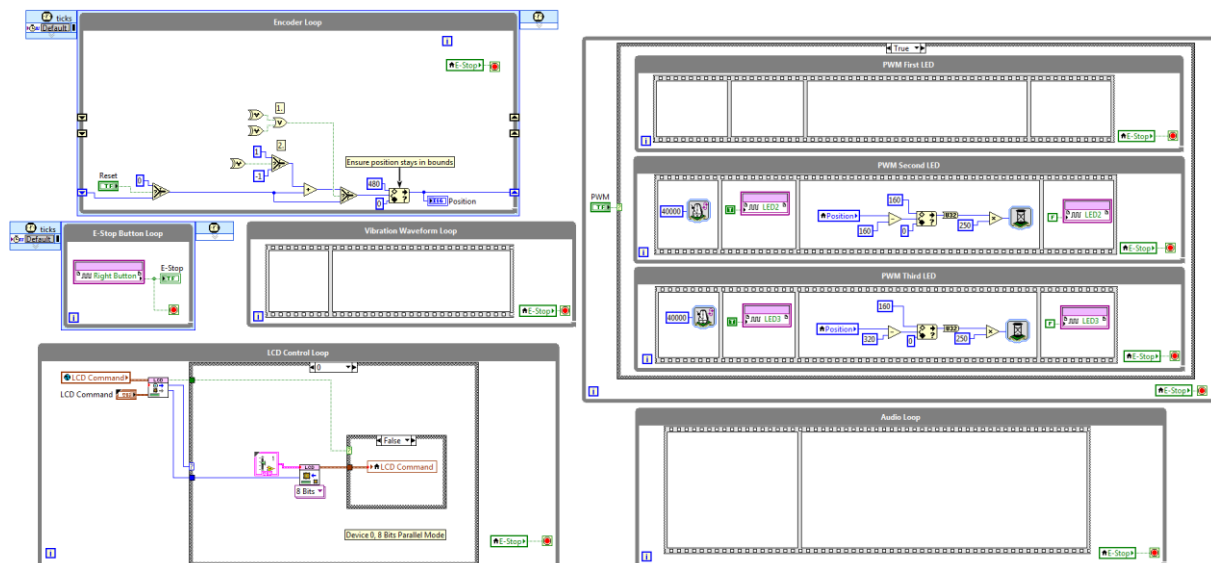


Implementation

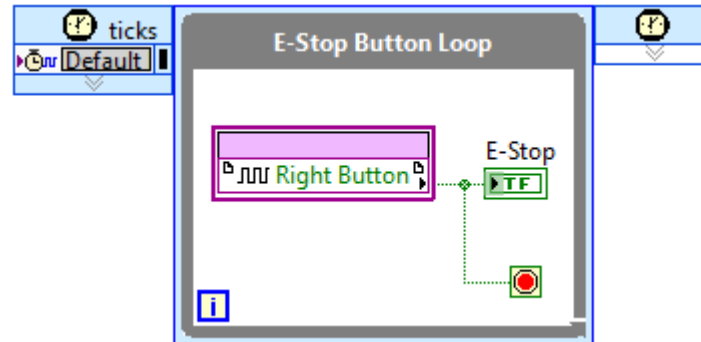
1. Open LabVIEW RIO Evaluation Kit.lvproj from .\2- Create FPGA Application directory.
2. Expand the LabVIEW RIO Evaluation Kit target > Chassis > FPGA Target in the project and double click to open the **FPGA.vi**



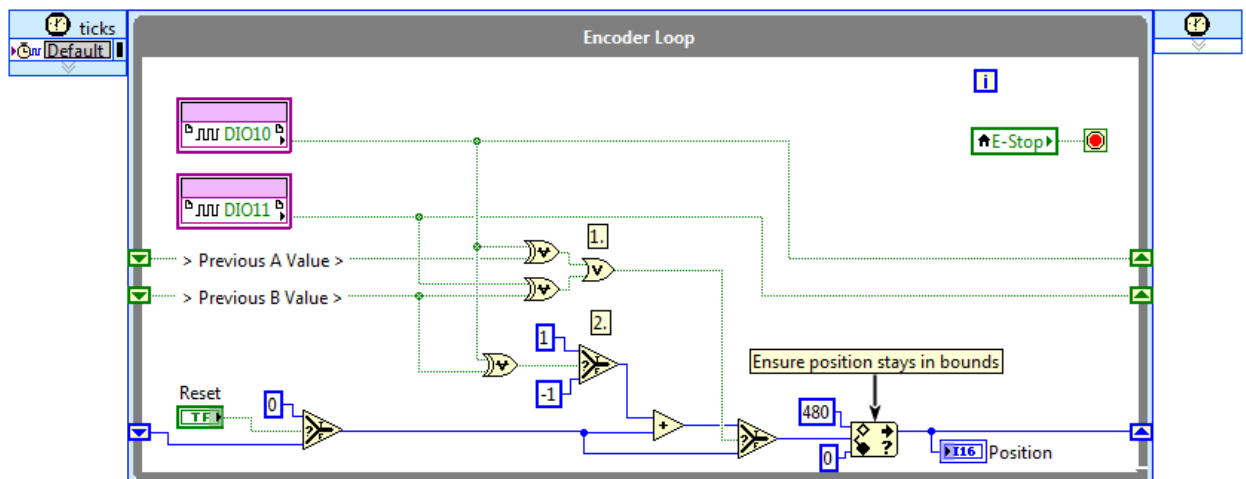
3. Open the block diagram by pressing <Ctrl-E>. Notice that the outline of several loops have already been created for you. This is mainly infrastructure logic, and you will be completing the core functionality.



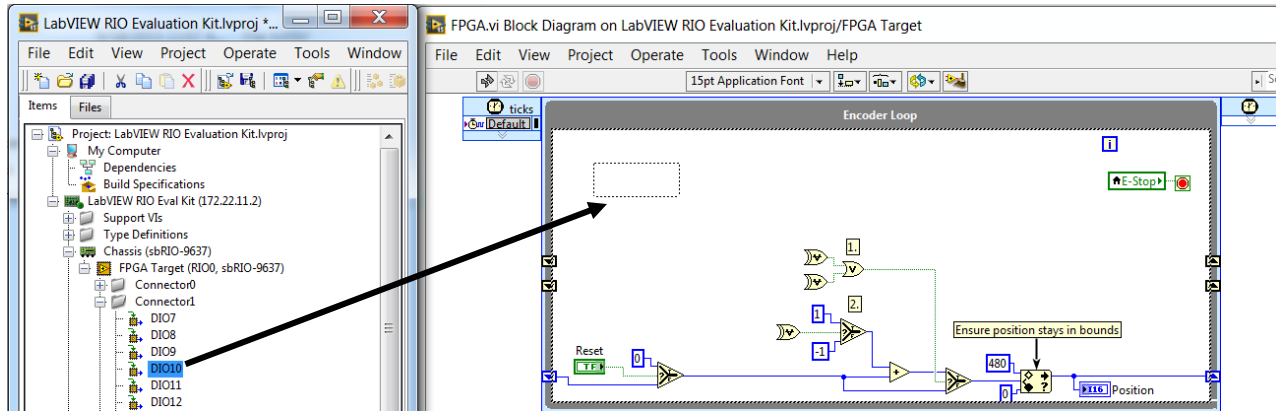
- First, view the *E-Stop Button Loop* as shown below. If the Right 'Stop' button (PB2) is pressed on the eval board, it will set a flag to stop all the other FPGA loops.



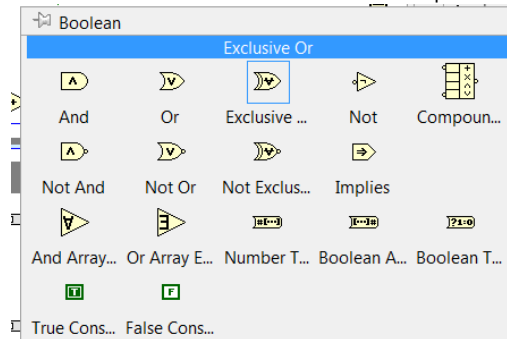
- I/O Node**—Underneath *FPGA Target* in your project, open the *Connector1* folder and find *Right Button* (DIO 13, digital pin). This I/O node was dragged into the loop from the project and represents the right push button (PB2) on your board. You can rename I/O to something meaningful by selecting the item and pressing F2 on your keyboard.
 - E-Stop**—This is the global stop button for the application, a local variable has been created to stop all the other FPGA loops based off of this boolean value.
 - Timed Loop**— a Timed Loop on the FPGA implements the code within one tick of the default hardware clock (40 MHz)
- Complete the *Encoder Loop* as shown below. This loop will constantly read the value of the quadrature encoder.



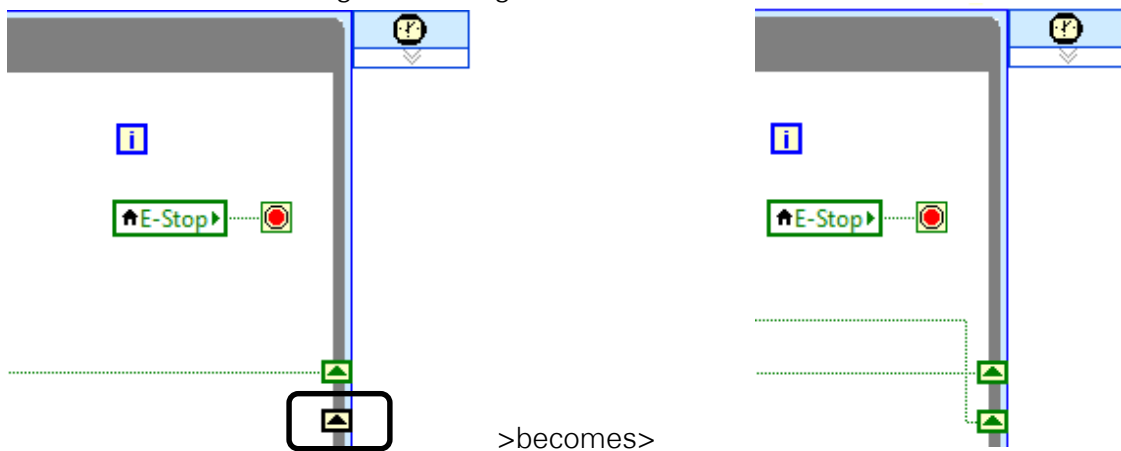
- **I/O Nodes**—Underneath *FPGA Target* in your project, open the *Connector1* folder. Drag I/O items DIO10 and DIO11 onto the block diagram. These digital lines are connected to the encoder.



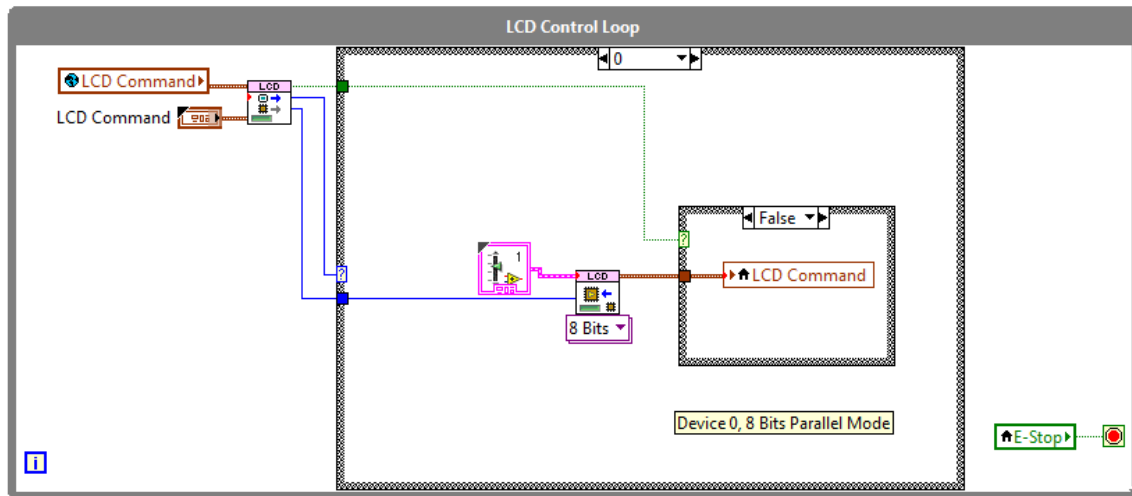
- **Boolean Logic**—Right-click to bring up the *Functions* palette and find the **Exclusive Or**, and **Or** functions in the *Boolean* subpalette.



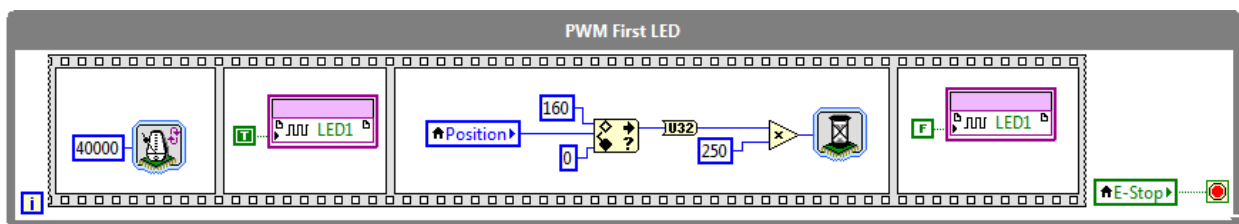
- **Shift Register**—Wire the boolean output value from **DIO 10** to the upper shift register and **DIO 11** into the lower shift register on the right edge of your loop, as shown below. These shift registers will carry those values between loop iterations to compare the position.
- Wire the remaining boolean logic



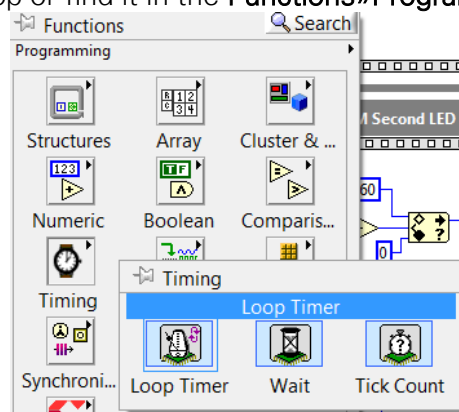
6. The *LCD Control Loop* is already completed for you and serves as the driver that controls the LCD screen through the 16 DIO pins connecting it.



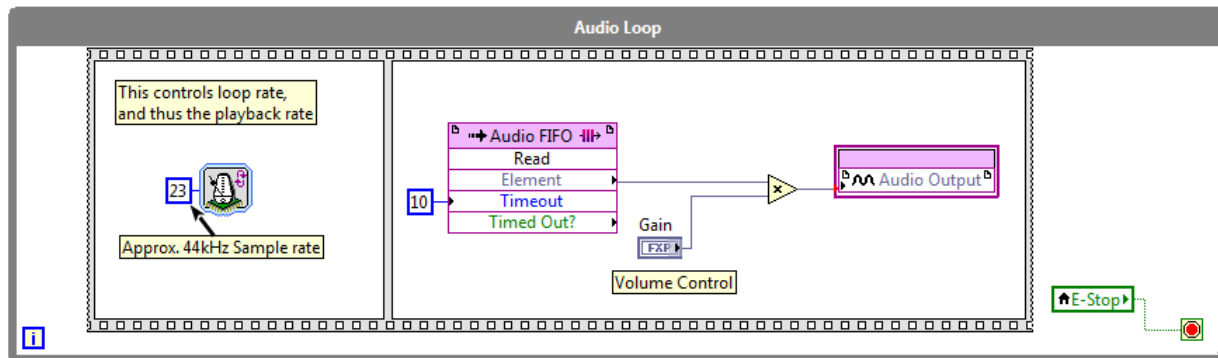
7. Complete the *PWM First LED* loop as shown below. This loop will be nearly identical to the two loops below it. You can highlight and copy the logic from the other loops, but take note to update the integer constant values. The goal of this loop is to turn an LED on with increasing brightness and off using PWM based on the position of the encoder. LED1 LED2, and LED3 on your board are connected to the digital lines DIO4, DIO5, and DIO6 respectively.



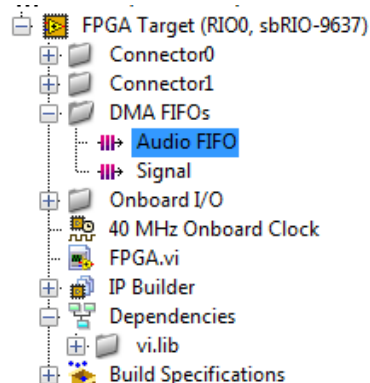
- **I/O Nodes**—Drag in two instances of LED1 from the project into frame two and four of the sequence structure as shown. I/O Nodes can be changed to write by right clicking on the middle of the node and selecting **Change to Write**.
- **Loop Timer**—This function controls how fast the loop executes. In this case, the loop executes once every 40,000 clock cycles (ticks). You can copy this from the *PWM Second LED* loop or find it in the **Functions»Programming»Timing** palette.



- **Position local variable**—create a local variable from the **Position** indicator in the *Encoder Loop* by Right-clicking on the indicator and selecting **Create»Local Variable**. Drag it from the Encoder Loop down to the PWM First LED loop and wire as shown.
 - **In Range and Coerce**—This logic prevents the loop from having timing violations when the position of the encoder continues to increase. Use the **Quick Drop** menu <Ctrl-Spacebar> to find and insert this VI if you did not copy it from another loop.
 - **Wait Timer**—The wait timer delays the execution of the loop for the specified number of clock cycles and is also found at **Functions»Programming»Timing**.
 - Complete the remaining integer logic as shown above.
 - **E-Stop Local Variable**—Note the local variable for the E-Stop indicator is wired to the stop terminal.
8. Complete the *Audio Loop* as shown below. This loop will read waveform data from the Audio_Left FIFO being sent from your real-time VI and output audio to the speaker connected to the AO0 terminal.

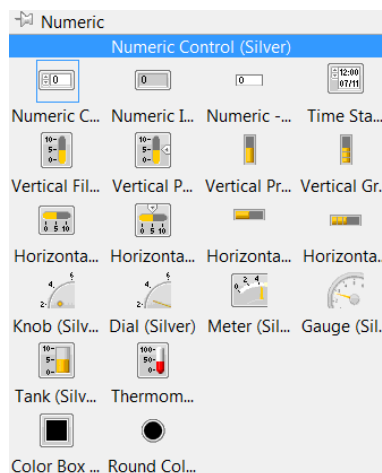


- **Loop Timer**—Find it in the **Functions»Programming»Timing** palette. Double-click the VI and set *Counter Units* to **uSec**. Place and wire a **Numeric Constant**. A loop rate of 23 uSec is approximately 44kHz, which is typical for audio applications.
- **I/O Node**—The **Audio Output** node can be found in the project *Connector0* folder. Drag it into the block diagram in the second frame like you did earlier.
- **FIFO Read**—Open the *DMA FIFOs* folder beneath the *FPGA target* in your project and drag **Audio_FIFO** onto the block diagram. This adds a FIFO read method. This function will allow the FPGA VI to read waveform data streamed from the RT VI you will create in the next exercise.

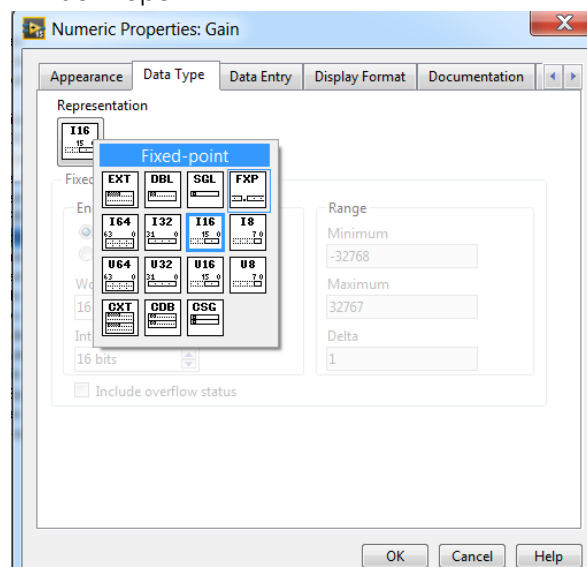


Direct Memory Access (DMA) FIFOs directly accesses memory to transfer data from FPGA target VIs to host VIs and vice versa, allocating memory on both the host computer and the FPGA target, yet acts as a single FIFO. Right-click the *Timeout* input and **Create»Constant** with the value as **10**, which specifies the number of clock ticks that the method waits for available space in the FIFO if the FIFO is full.

- **E-Stop Local Variable**—Note the local variable for the E-Stop indicator is wired to the stop terminal.
- **Gain**—The waveform data from the FIFO is multiplied by the **Gain** as a form of volume control. Press <Ctrl-E> to switch to the Front Panel and right click to bring up the *Controls* palette. Place down a control from **Controls»Silver»Numeric»Numeric Control**. Rename the control **Gain**.



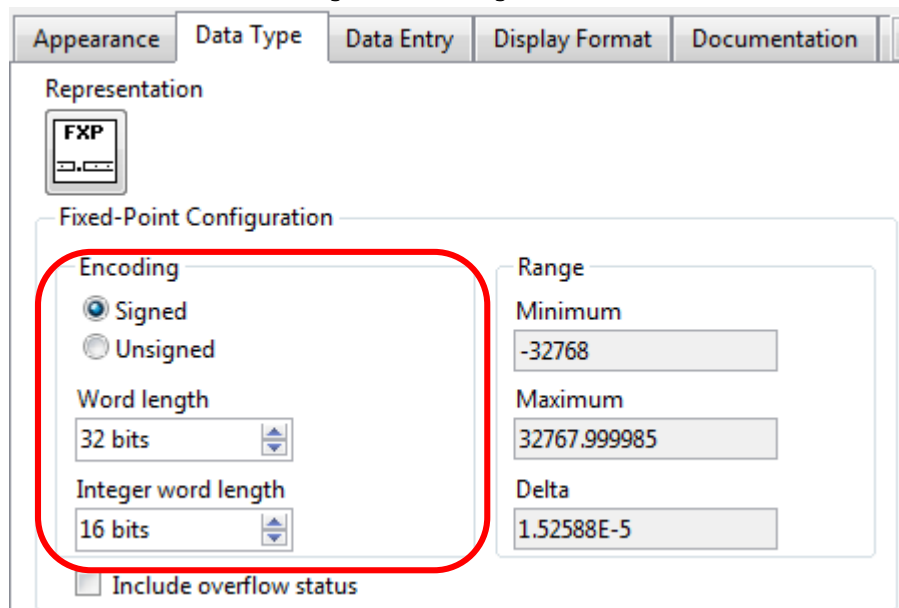
Right-click on the control and select **Properties**. In the *Numeric Properties* window that opens, click on the **Data Type** tab. Click on the Representation icon that currently shows **I16**, and in the menu that appears select **FXP** (Fixed-Point). Leave the window open.



Note: The FPGA will be acquiring data in decimal format, however in contrast to processors, 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* in the LabVIEW Help.

In the same window, under *Fixed-Point Configuration*, 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.

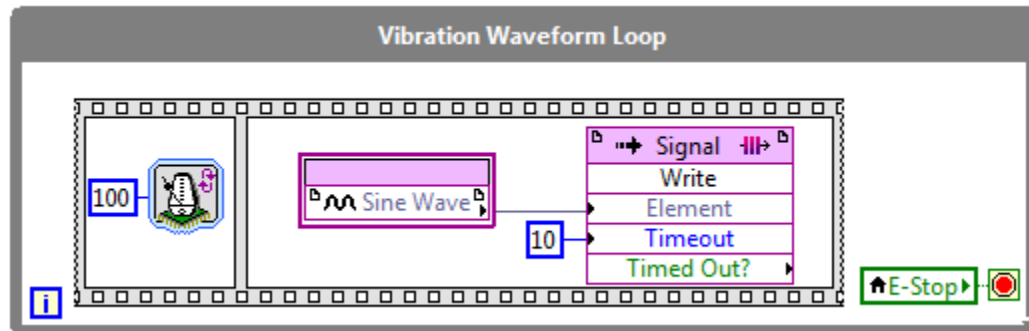
Encoding: Signed
Word length: 32 bits
Integer word length: 16 bits



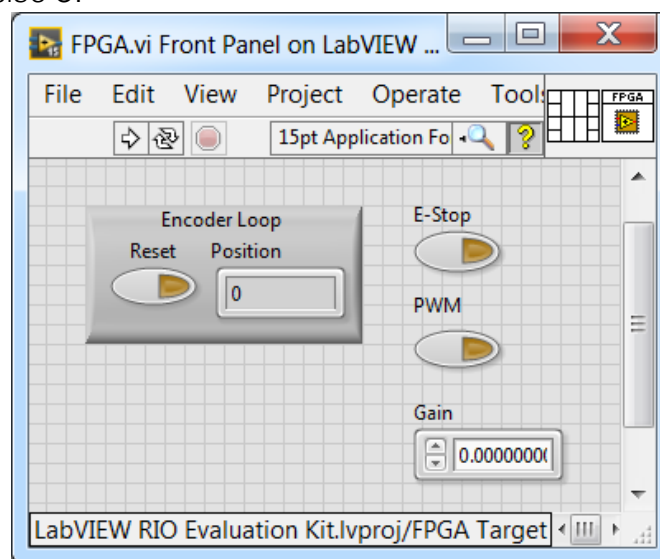
Click **OK** to close the *Numeric Properties* window.

- Wire the **Element** and **Gain** outputs to a **Multiply** node (Functions»Programming»Numeric) and then into the **Audio Output I/O** node.

9. Complete the *Vibration Waveform Loop* as shown below. This loop is taking a simulated vibration measurement, generated by the sine wave function generator on your board, at a given frequency and amplitude that you change with the knobs on your board.



- **Loop Timer**— Find it in the **Functions»Programming»Timing** palette. Double-click the VI and set *Counter Units* to **uSec**. Right-click on the input terminal and select **Create»Constant**. Type in a loop rate of 100 uSec.
 - **I/O Node**—The **Sine Wave** I/O node (AI0) can be found in the project in the *Connector0* folder. Drag it into the block diagram.
 - **FIFO Write**— Open the *DMA FIFOs* folder beneath the *FPGA target* in your project and drag **Signal** onto the block diagram. This adds a FIFO write method. This function will allow the FPGA to stream waveform data to the RT VI. Wire the **Sine Wave** output into *Element* input. Right-click the *Timeout* input and **Create»Constant** with the value as **10**.
 - **E-Stop Local Variable**—Note the local variable for the E-Stop indicator is wired to the stop terminal.
10. Arrange your front panel controls and indicators as shown below and ensure the spelling and capitalization are correct, as we will be referencing them by name in Exercise 3.

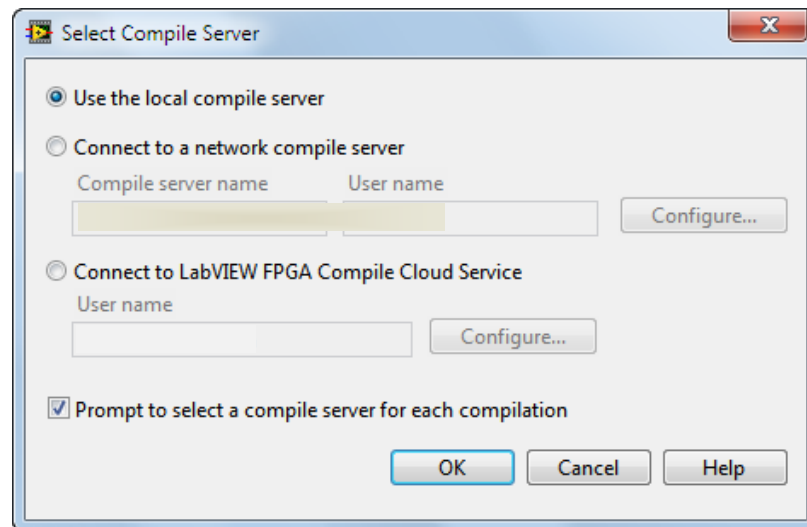


Start LabVIEW FPGA Compilation Process

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

In contrast to LabVIEW applications running on a processor, an FPGA 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 in hardware. This is a two-step process. First LabVIEW generates intermediate files and then it transfers them to the Xilinx compilation tools for the final compilation stages. To learn more about simulating your LabVIEW FPGA code before compiling, please read [Testing and Debugging LabVIEW FPGA Code](#).

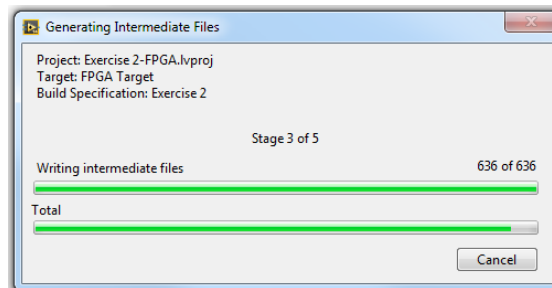
2. In the *Select Compile Server* dialog box that appears, there are three options to compile your code. We will not consider the network compile server in this evaluation experience.



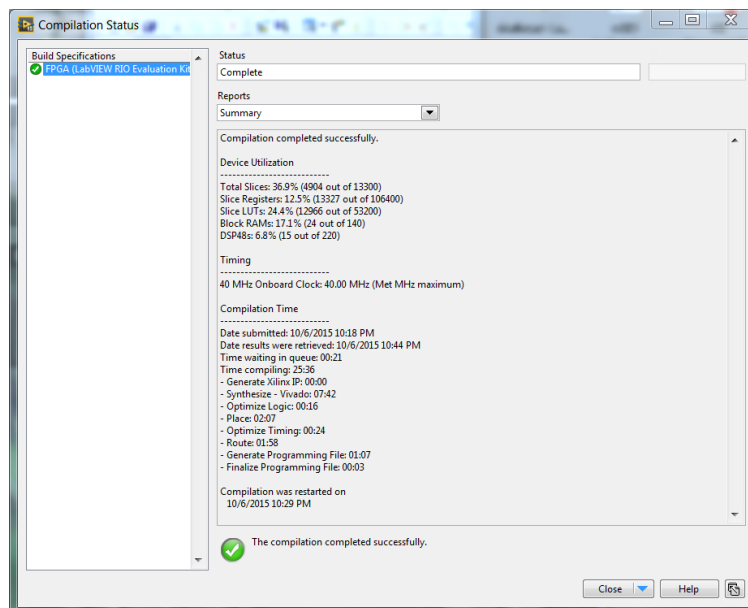
- a. The recommended option is the [NI LabVIEW FPGA Compile Cloud Service](#), which utilizes high-performance servers that compile 30-50% faster than on a typical development machine and can execute multiple parallel compilations. Click on the link above or navigate to ni.com/trycompilecloud and enter your preferred email address. NI will send you information to quickly setup your NI Cloud Services Portal account. This option is required if you're using Windows 8 as the Xilinx Compilation Tools are not supported.

To enter your credentials in LabVIEW, select the **Connect to LabVIEW FPGA Compile Cloud Service** option. Press the **Configure...** button to enter the NI Cloud Services Portal login credentials you just created. You will have full access to this service during your 90 day evaluation and long term when you purchase LabVIEW you will have access included for free with your active Standard Service Program (SSP) membership.

- b. The other option is to install the *LabVIEW FPGA Xilinx Vivado 2014.4 Compilation Tools* locally to your computer from the second DVD that was included in your kit accessories box or download it from ni.com [here](#). Note you will need around 5 GB of free space on your hard drive to install these tools and it is required if you do not have internet access. Once installation is complete, select **Use the local compile server** to use this option.
3. Click **OK** and the intermediate file generation process will automatically start, where LabVIEW translates your graphical implementation to the native Hardware Description Language for an FPGA. Once complete, LabVIEW will kick off the Xilinx compilation tools. In total, the process should take about 20 minutes to finish.



Once complete, you can see the device utilization, or various FPGA resources that your code uses on the chip. Unlike a processor, it is safe to use nearly 100% of the FPGA resources, since it is implemented hardware.



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

Test the FPGA Application

When the FPGA VI finishes compiling, you can run the following tests on the application.

1. Double click on the **FPGA.vi** from the Exercise 2 project that should still be open in the background.

Note: If you did not correctly complete and compile the Exercise 2 FPGA application, you can reference the solution in the `.\Solutions\2- Create FPGA Application` folder. Run the **FPGA.vi** directly but note that you may need to recompile the *FPGA.vi* for your specific target. Reference the *Using the Solutions* section at the front of the manual for more details on using the solution.

2. Click the **Run** arrow if the VI is not already running and complete the following tests:
 - ✓ Click the **PWM** control to True on the front panel. LEDs 1-3 should light up with increasing intensity as you turn the encoder to the right.



- ✓ Press the Stop button (PB2) on your board and ensure the E-Stop indicator on the FPGA VI lights up.



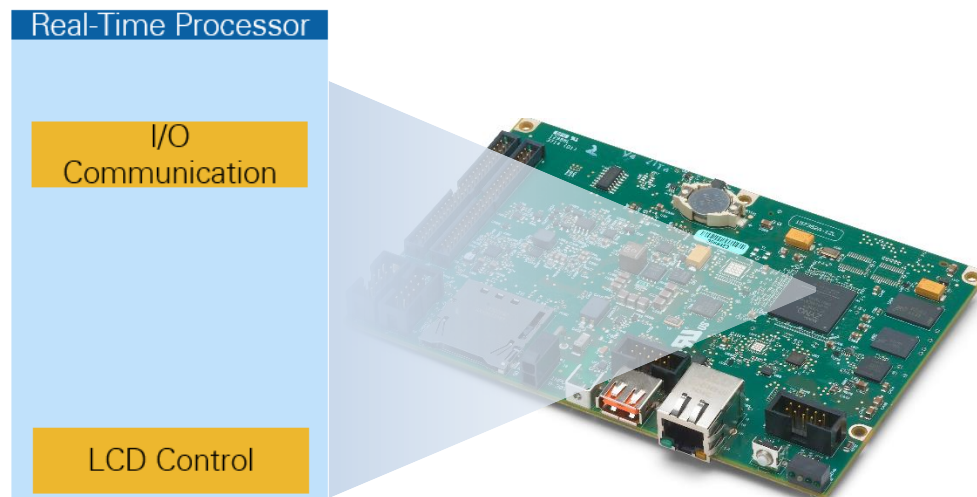
Note: Since the real-time application contains the logic to write to the LCD and exchange data with the audio and signal FIFOs, they will not yet update until you complete exercise 3. The next testing section for the real-time application will exercise this functionality.

3. Save LabVIEW RIO Evaluation Kit.lvproj.

Exercise 3 | Develop the Real-Time Application

Summary

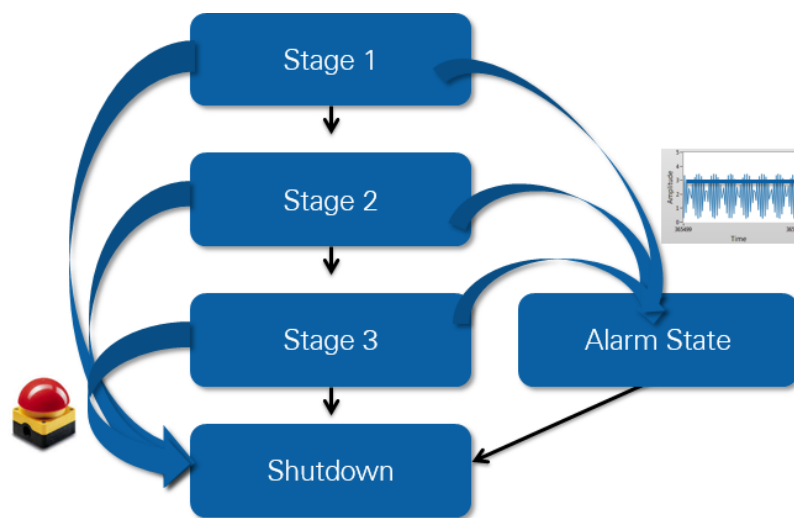
Create an RT VI with two loops to exchange data with the FPGA VI and write to the onboard LCD.



State Diagram

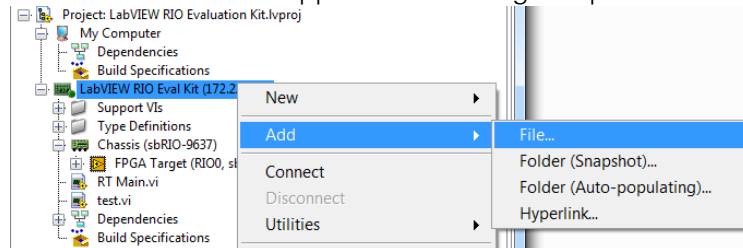
Normal machine operation will progress from stage 1 » 2 » 3 then proper shutdown. If the alarm condition is tripped, when the machine's vibration waveform amplitude crosses the threshold, potentially indicating a damaged bearing that a technician should investigate, the application will enter the alarm state and automatically progress to proper shutdown. If you press the Stop button on your board (PB2) you will automatically progress to proper shutdown.

Example: Stage 1-Align part » Stage 2-Weld part » Stage 3-Scan to inspect welded joint

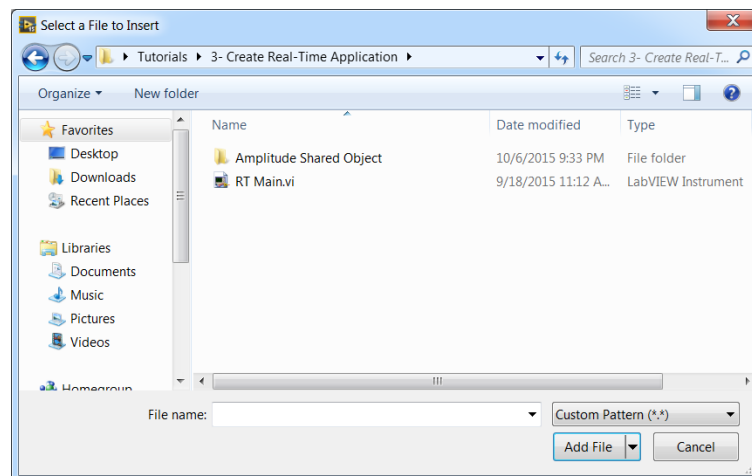


Implementation

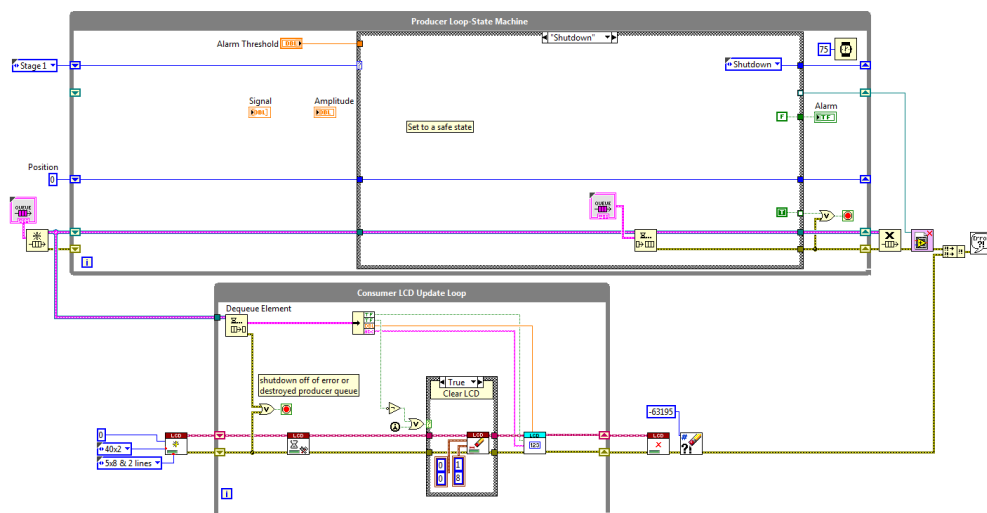
1. Open LabVIEW RIO Evaluation Kit.lvproj, which you started during Exercise 2, if it isn't already open.
2. In the LabVIEW Project Explorer window, right-click on the *LabVIEW RIO Eval Kit* target and select **Add»File...** to add the RT application starting template.



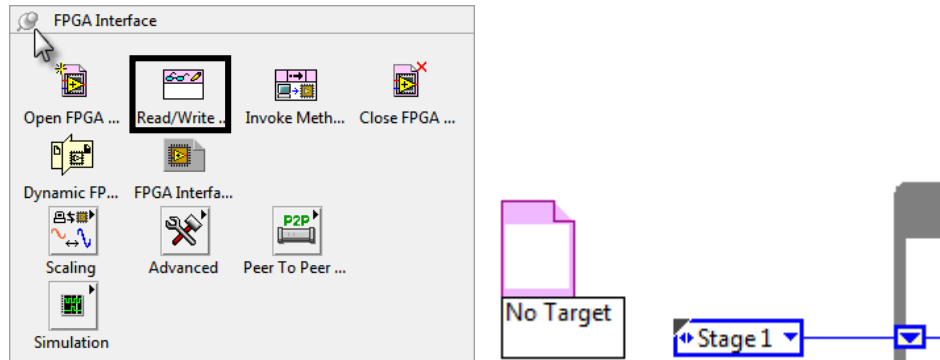
3. Navigate to the *3- Create Real-Time Application* folder and select **RT Main.vi**.



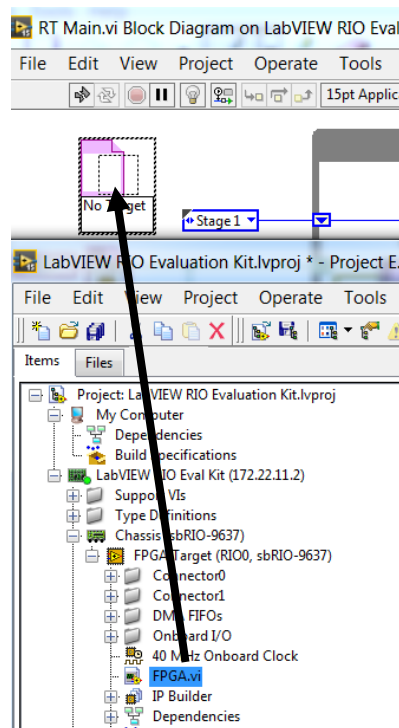
4. Double-click to open **RT Main.vi** and notice again that the infrastructure is already completed for you, a state machine producer/consumer architecture with a queue to pass data between loops. The Producer Loop will receive data from the FPGA and the Consumer loop will write text to the LCD.



- To begin programming, first open a reference to the FPGA code you compiled. Navigate to the FPGA Interface palette (Functions»FPGA Interface) and pin it to the block diagram. From this palette drag an **Open FPGA VI Reference** to your block diagram, to the left of the top while loop and Stage 1 constant.

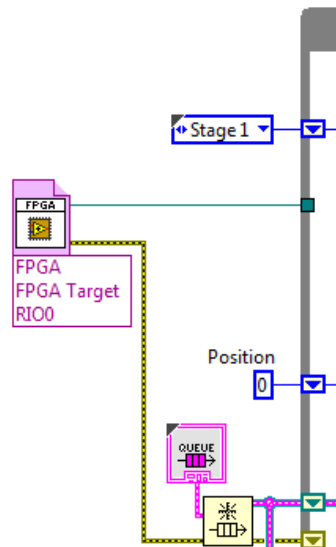


- If your VI from exercise 2 is still compiling, find the **FPGA VI** from your project under *Chassis»FPGA Target* and drag it into the unconfigured **Open FPGA VI Reference** to the left of the top while loop. **Note:** The Run Arrow may be broken if your FPGA VI is not yet done compiling.



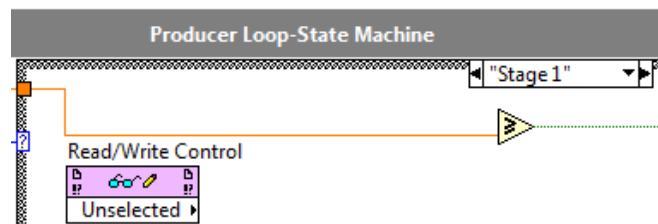
If your VI is done compiling, instead right-click on the **Open FPGA VI Reference** VI and select **Configure Open FPGA VI Reference....** Then select your compiled bitfile in the \2- Create FPGA Application\FPGA Bitfiles directory.

7. Wire the **FPGA VI Reference Out** to the border of the top while loop as shown below.
8. Wire the **Error out** of the Open FPGA VI Reference VI to the **error in** input of the **Obtain Queue VI** as shown below.

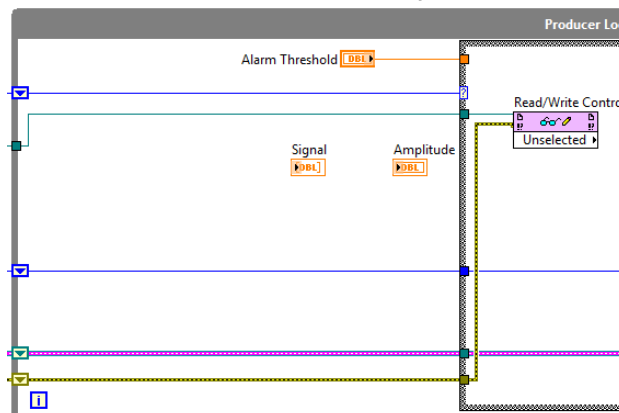


Note: The FPGA Target text underneath the **Open FPGA VI Reference VI** will differ depending on if you chose the VI or compiled bitfile.

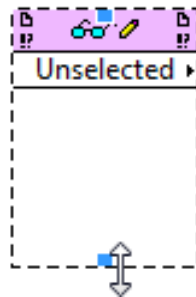
9. Now insert a **Read/Write Control** (Functions»FPGA Interface»Read/Write Control) inside the **Stage 1** case of the case structure inside the top while loop. This will define the data communicated between the FPGA and real-time application.



10. Wire the FPGA reference and error wire from the border of the while loop through the case structure border into the **Read/Write Control input** terminals.



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



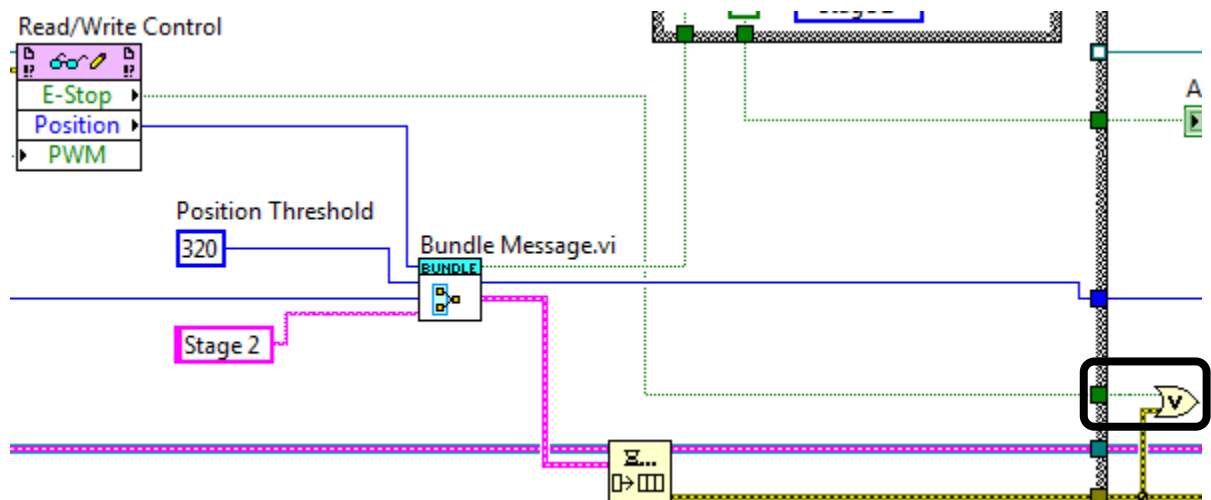
12. To configure the unselected terminals, left-click on each of them and select the control or indicator name from the FPGA VI front panel that should be polled. Configure the terminals as shown below to include **E-Stop**, **Position**, and **PWM** from top to bottom.



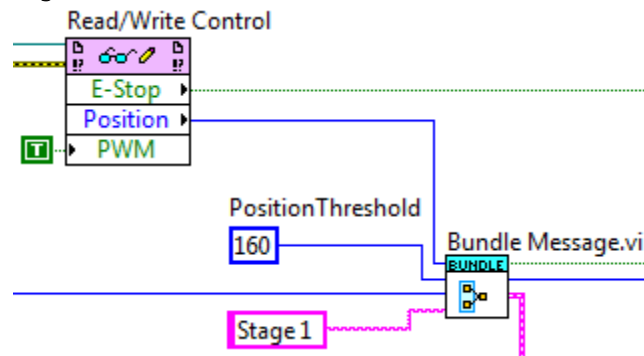
If any of the **Read/Write Control** terminals are facing the wrong direction, right-click on the terminal text and select **Change to Read** or **Change to Write** to match the image.

13. Wire a **True Constant** to **PWM**, during normal execution we want the three LEDs to light up depending on what stage the application is in and the progress made.

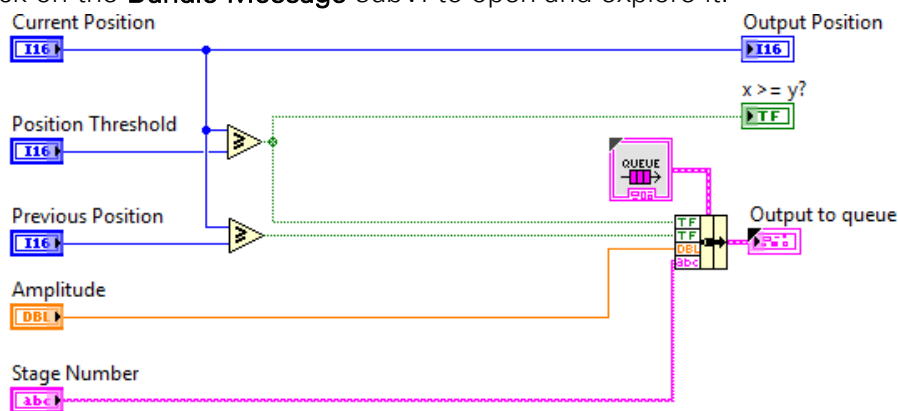
14. Wire the Read/Write Control **E-Stop** boolean output through the edge of case structure and into the top terminal of the **Or** node as one way to stop the loop, the other is if there is an error.



15. Wire the Read/Write Control **Position** integer output into the **Current Position** terminal on the **Bundle Message** subVI.

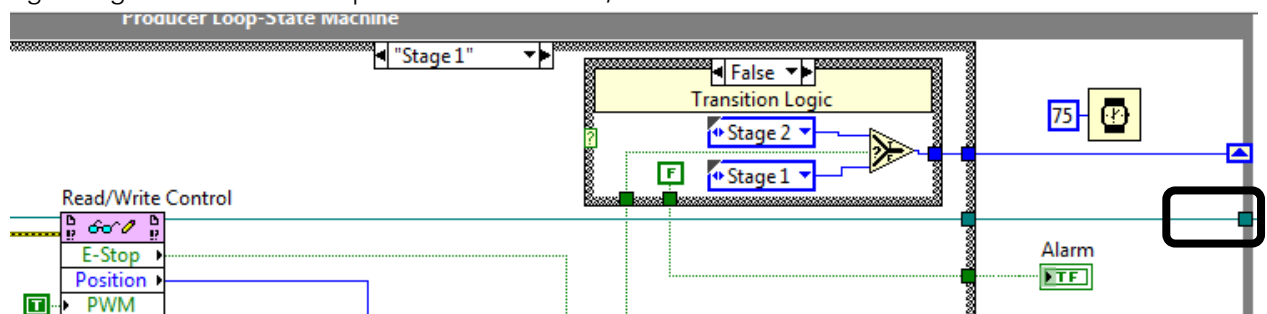



16. Double-click on the **Bundle Message** subVI to open and explore it.



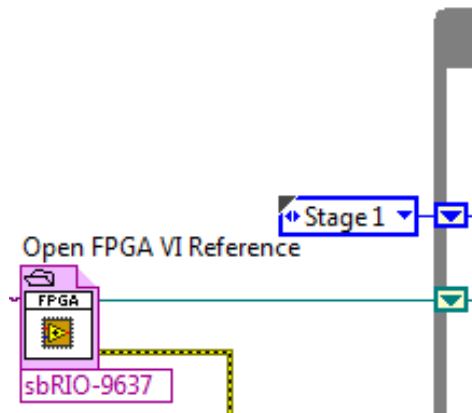
This VI checks if the current encoder position has passed the threshold for the current stage, and if so will set the Transition Boolean indicator to true. It also bundles position, amplitude, and the stage number string to add to the queue. The consumer loop will then take this information and write the LCD text accordingly.

17. Close the **Bundle Message** subVI
18. Wire Read/Write Control **FPGA VI reference** out through the case structure and to the right edge of the while loop to create a tunnel, as shown below.

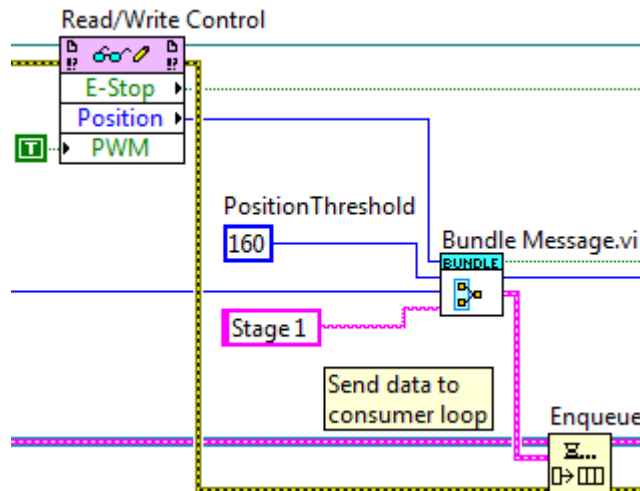


19. Right-click on the while loop tunnel you just created, as shown above, and select **Replace with Shift Register**. A shift register enables the passing of data from one loop iteration to the next. LabVIEW replaces the tunnel you right-clicked with a shift register terminal, and the cursor becomes a shift register icon (). Hover over the tunnel for

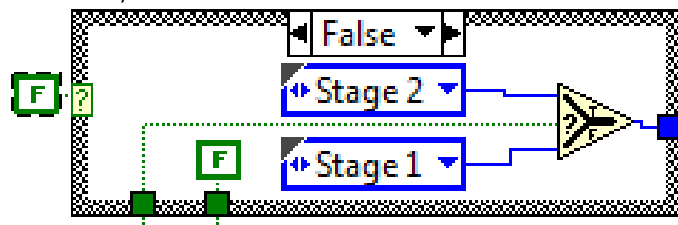
the FPGA VI Reference on the left side of the loop until it flashes, then click the tunnel to replace it with a shift register. It should look like this:



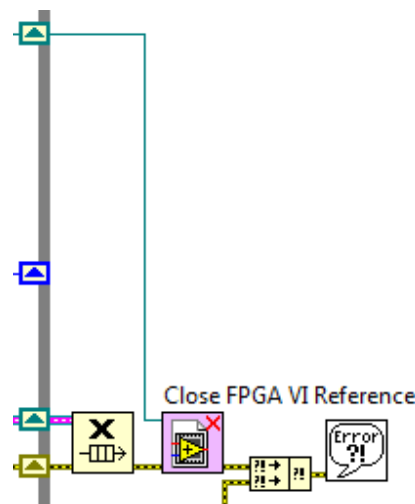
20. Wire the Read/Write Control **error out** to the **error in** terminal on the **Enqueue Element** VI as shown below. This will ensure proper dataflow in the application.



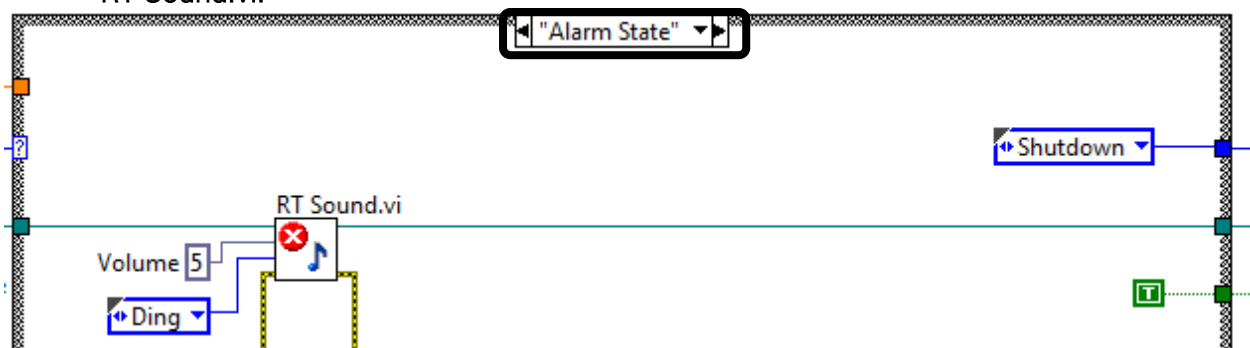
21. Finally, look at the State Transition logic. Eventually you will add in the alarm condition code to monitor if the waveform amplitude crosses the threshold. For now, wire a **False** Constant to this case structure's **case selector (?)** and you'll see that the code will progress to stage 2 when you've turned the encoder 160 clicks (Position Threshold).



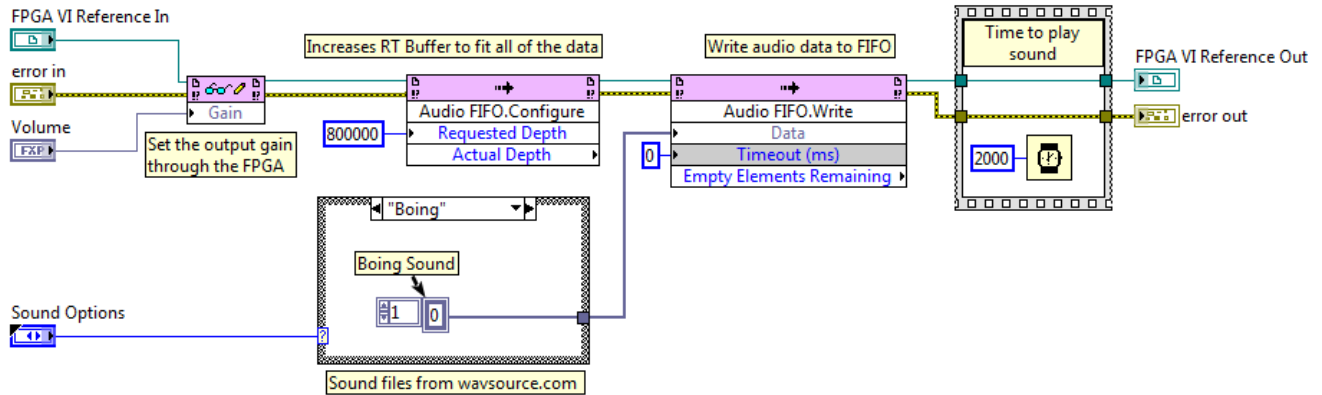
22. Wire the FPGA reference from the shift register on the edge of the loop into the **FPGA VI Reference In** terminal on the **Close FPGA VI Reference VI** as shown below, to properly close the reference.



23. Click the Case Structure selector label, highlighted below to open the **Alarm State**, which will play a sound when the alarm threshold has been hit. Wire the FPGA VI Reference from the case structure border into the **FPGA VI Reference In** terminal on the **RT Sound.vi**.



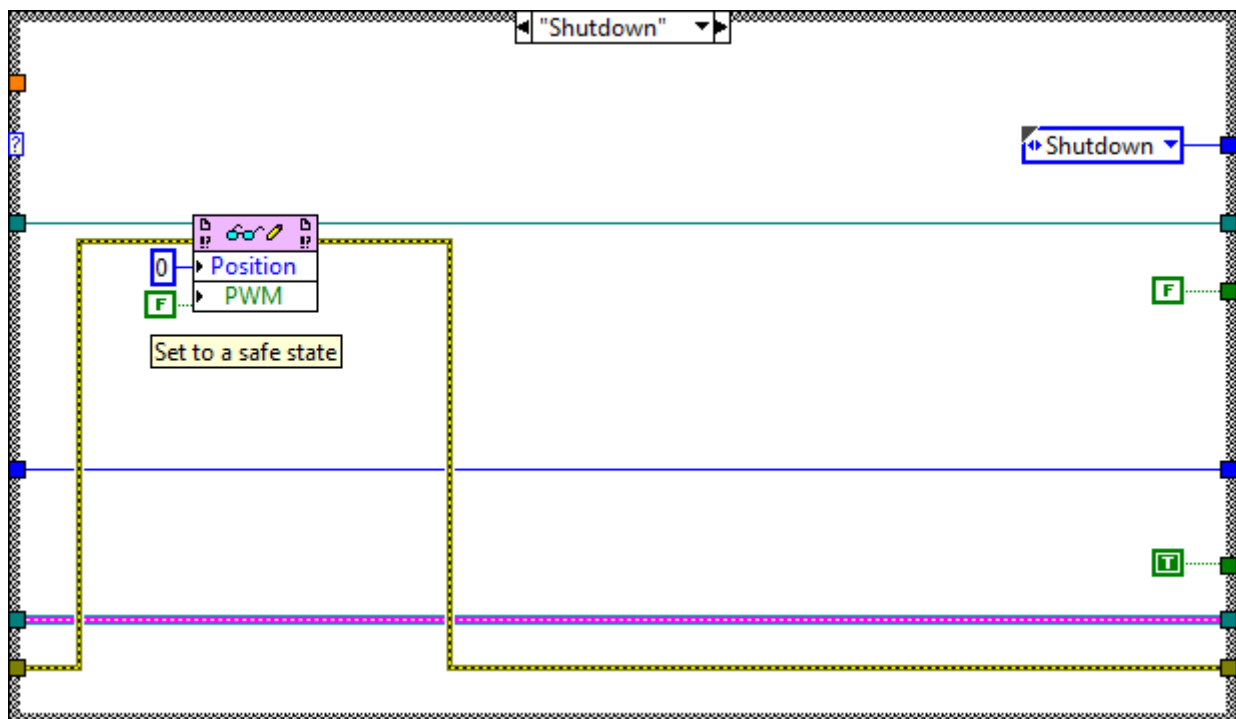
24. Wire the **FPGA VI Reference Out** from the **RT Sound.vi** to the existing tunnel on the border of the case structure.
25. Open the **RT Sound.vi**, which has been completed for you. This VI streams a sound array through a DMA FIFO to the FPGA to play an alarm through the speaker by utilizing the analog output on the board. Each alarm sound is stored in a pre-built array containing waveform data.
- Before the waveform is streamed, the gain of the output must be set. The waveform data is multiplied by this value, allowing for volume control. The RT to FPGA DMA FIFO must also be configured to be large enough to allow seamless data streaming. Look at the different cases for the sound options (Boing, Buzzer, Ding, and Horn). Ding is the default.



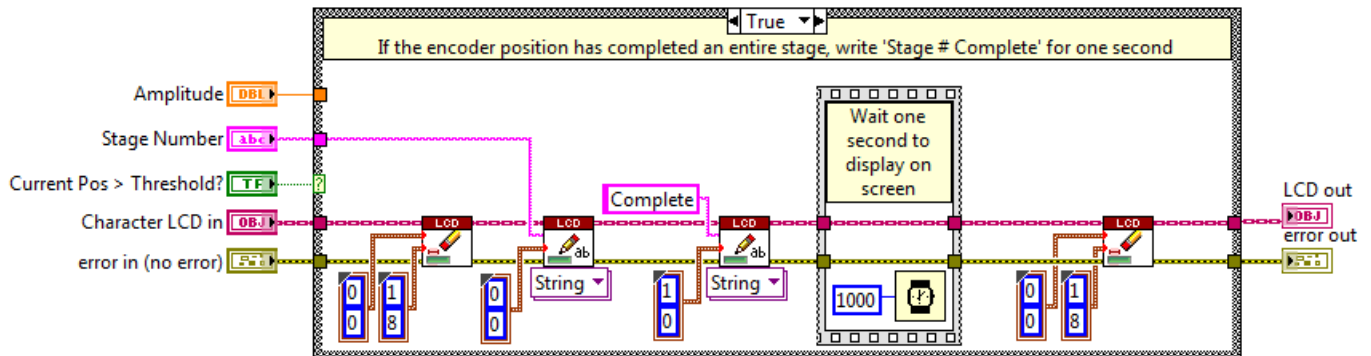
26. Close the RT Sound.vi

After the Alarm State the code transitions into Shutdown State, where it stops the Producer loop and the subsequent releasing of the queue creates an error at the Dequeue Element VI in the Consumer loop that forces it to stop as well.

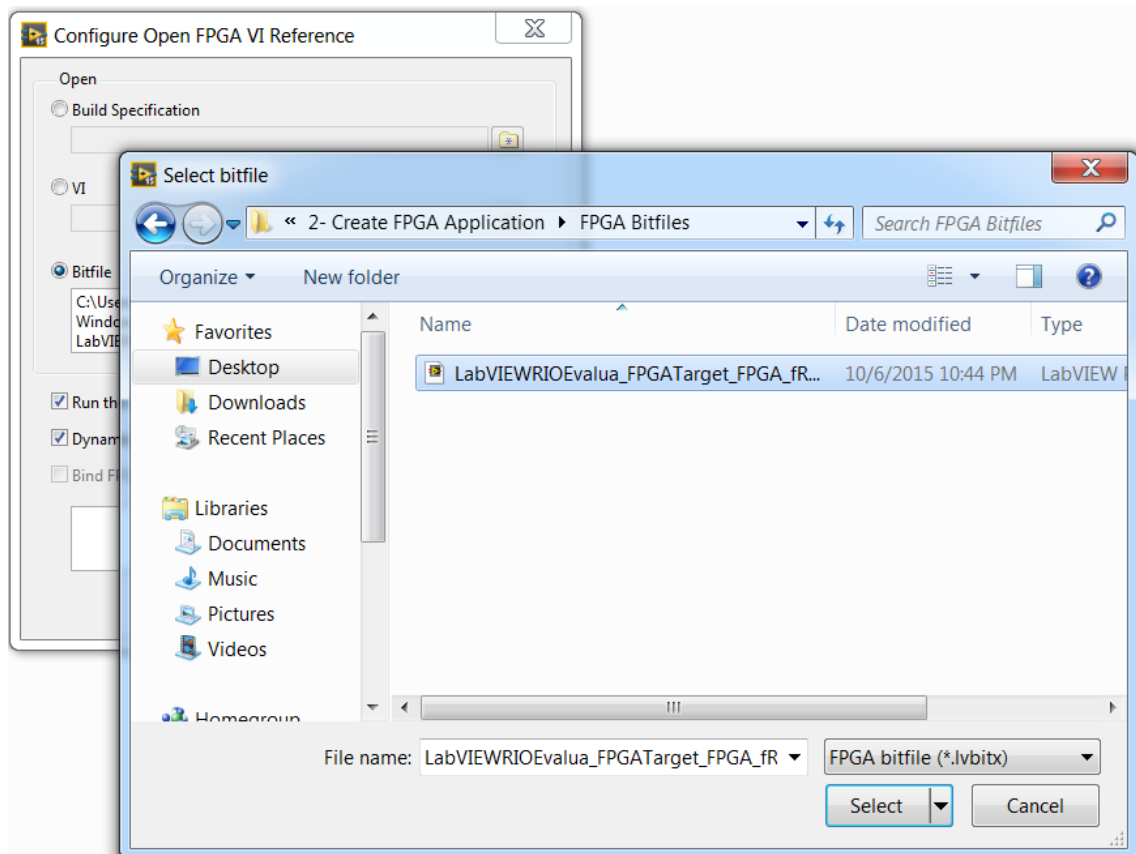
27. Place down another FPGA **Read/Write Control** in the Shutdown case and wire the FPGA VI Reference and error inputs. Drag down and select **Position** and **PWM**.
28. Select **Position** and right-click to select **Change to Write**. Right-click and select **Create»Constant** on the Position integer input and select zero to reset the encoder position to a default safe state upon shutdown, as shown below.
29. Create a **False Constant** and wire it to **PWM** to turn off the LEDs upon shutdown.
30. Wire the **error out** and **FPGA VI Reference Out** to the tunnels on the right side of the case structure as shown below.



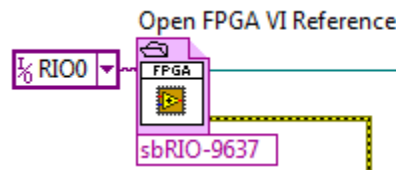
34. Double-click to open the **LCD Write** sub VI which contains the logic to write to the LCD. During normal stage operation **AMP** (amplitude) and the current amplitude value will be written to the screen. Otherwise, when a stage has completed, **Stage # Complete** will appear for 1 second.



35. Close the **LCD Write.vi**
36. If your FPGA VI was not compiled at the start of the exercise, the bitfile, or compiled FPGA static file, now needs to be referenced. Right-click on the **Open FPGA VI Reference** VI and select **Configure Open FPGA VI Reference...**
37. In the dialog that appears click the radio button next to **Bitfile** and browse to your FPGA bitfile from the previous exercise. Click **Select** and then **OK** in the dialog window to exit.

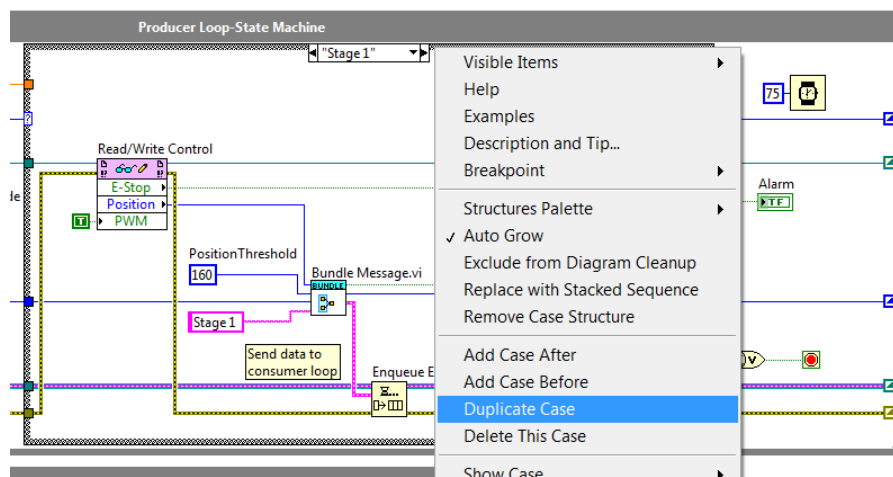


38. To finish the FPGA setup, right-click on the **resource name** input terminal on the **Open FPGA VI Reference** and select **Create»Constant**. From the constant drop down select **RIO0**. This is the name of your FPGA target resource from the LabVIEW Project.

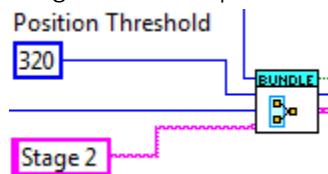


Create Additional Stages

39. Create two more instances of this *Stage 1* case. Right-click on the case structure border and choose **Duplicate Case**, ensuring that the new case is titled **Stage 2**.

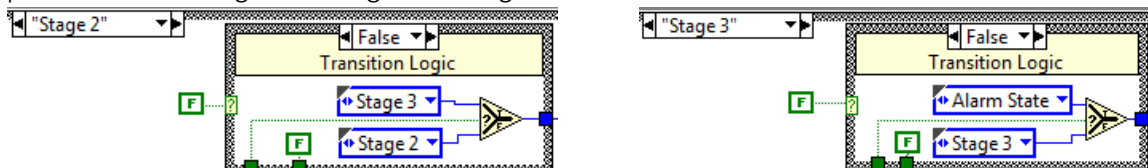


40. Change the **Position Threshold** constant to **320**. This allows for the second LED on your board to fill sequentially from during Stage 2 as the encoder position value increases from 160 to 320. Also edit the string constant input to say **Stage 2**



41. Duplicate the case another time using the same procedure, ensuring that the new case is titled **Stage 3**. Change the **Position Threshold** Constant to **480** and string constant to **Stage 3**.

42. Edit the state transition logic in Stage 2 and Stage 3 so that normal operation will proceed from Stage 1 » Stage 2 » Stage 3 » Alarm State » Shutdown.



43. Save and **Run** the VI.

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.

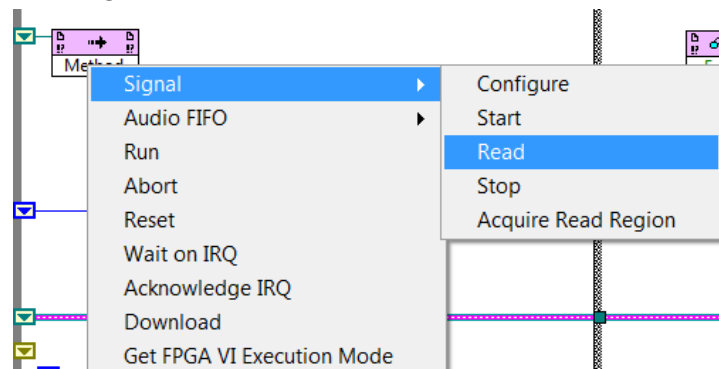
44. Turn the encoder right, when the first position threshold has completed it will say 'Stage 1 Complete'. It will show *Amp 0.00* otherwise, which you will complete in the next section. Continue through Stage 2 and 3 accordingly. As you turn the encoder, the intensity of LED 1, 2, or 3 will increase via PWM. After Stage 3 the alarm will sound (if speaker connected) and the code will shutdown automatically. You can press the Stop button on the board (PB2) at any time to stop the code.

Add Waveform Acquisition and Alarm Condition

45. Add in the vibration waveform *Signal FIFO* and alarm functionality. Right click on the FPGA VI reference shift register and select **FPGA Interface Palette»Invoke Method**. Delete the reference wire that is there and re-wire the reference through the method and to the case structure as shown below.



46. Click on the Method text and select **Signal»Read**. This is how we will read waveform data from the DMA FIFO.

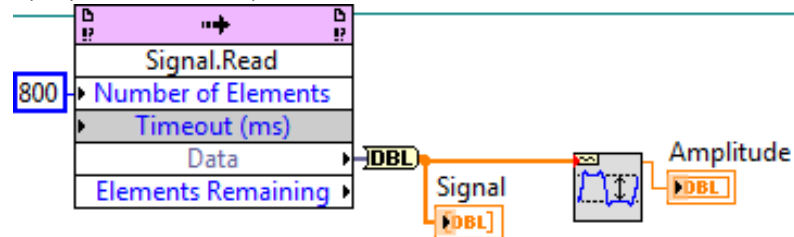


47. Right click on the **Number of Elements** input terminal and **Create»Constant**. Type **800** to read 800 elements off the FIFO every loop iteration.

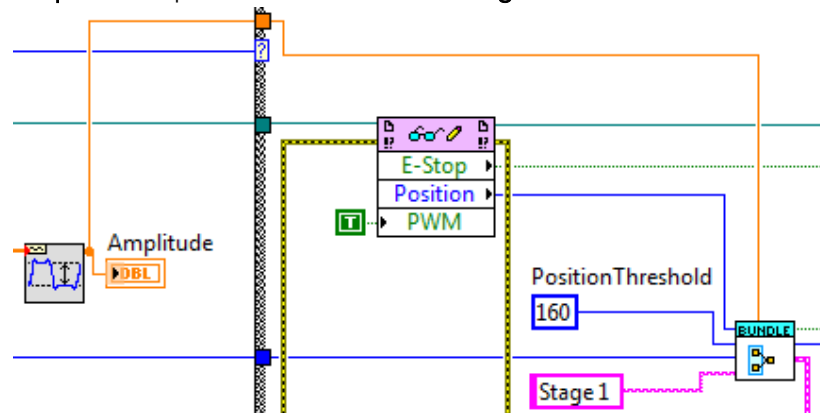
48. Place down a **To Double Precision Float** conversion node to convert the native FPGA Fixed Point Data type to Double Precision Floating Point. Wire the **Data** output to the conversion node and into the **Signal** indicator. This will display the raw sinewave on the front panel.

49. Place down an **Amplitude and Levels.vi** using quick drop and wire the data into the **signal in** terminal. You will notice a red dot on the terminal, meaning LabVIEW is coercing the data type for this VI to accept it, which is normal.

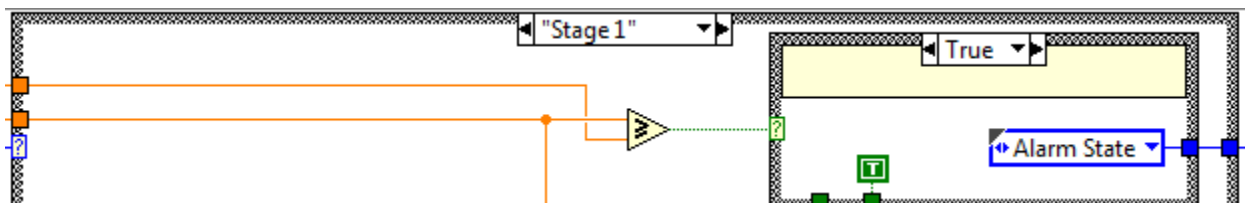
50. Wire the **amplitude** output from the **Amplitude and Levels.vi** into the **Amplitude** indicator, to display on the front panel.



51. Wire a branch off of the Amplitude output up to the case structure. In **Stages 1-3**, wire this into the **Amplitude** input of the **Bundle Message** subVI.



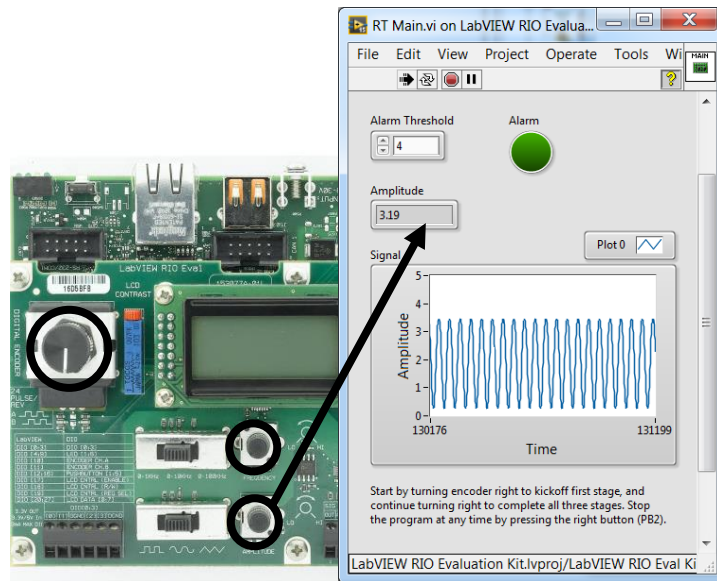
52. Delete the **False Constant** placeholder for the State Transition logic and replace it with a **Greater or Equal** node. Wire in the **Amplitude** to the top terminal and the **Alarm Threshold** control into the bottom terminal. If the amplitude exceeds the threshold the code will enter the Alarm State.



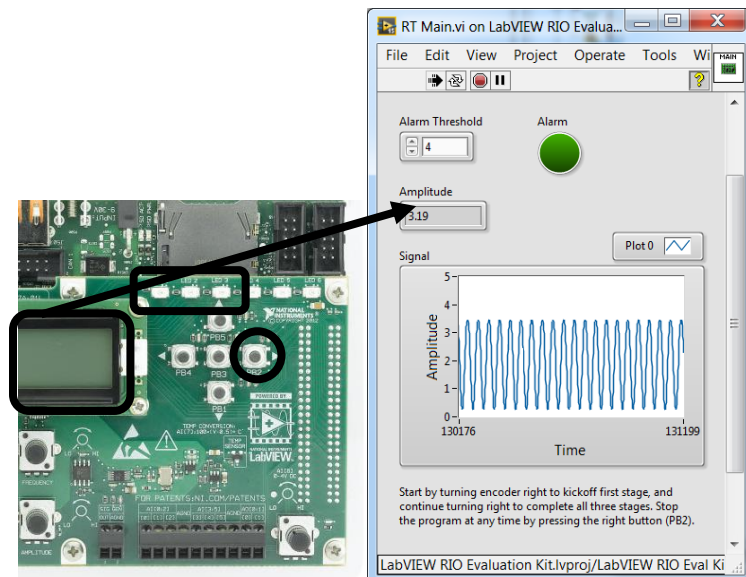
53. Copy and paste this logic in **Stages 2 and 3**.

Test the Real-Time Application

1. Click the **Run** arrow to compile and deploy the real-time application to the target, which also includes the FPGA application since the VI references the FPGA bitfile you just selected. Save the VI if prompted.
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.



Turning the Encoder to the right should progress through the three stages and turning the frequency knob will change your waveform, and turning the amplitude knob will allow you to test your alarm logic.



You should see AMP and the current value on the LCD match the front panel value and pressing PB2 should stop the application and I/O. The three LEDs should light up as you progress through the stages. The LCD now displays the different menus that the real-time application generates and transfers to the FPGA.

3. When you are finished testing the real-time VI, turn the encoder to complete all three stages to exit the application normally.
4. Save all of the project files by selecting **File»Save All** in the Project Explorer window.

Optional Exercise-Call existing C library from LabVIEW RT VI

For many applications, it is important to consider opportunities for code reuse whenever moving forward with a new project or platform. The **Call Library Function Node** in LabVIEW provides an interface to preexisting C code compiled into a shared library.

In order to call external code from within LabVIEW on the NI Linux Real-Time operating system, the code must be compiled into a shared object file, or .so file. This is the general equivalent to a .dll in the Windows environment. National Instruments provides [C & C++ Development Tools for NI Linux Real-Time, Eclipse Edition](#), which were included on your installation DVD, including the Eclipse IDE and GNU C cross-compilers for Linux. You can use your preferred IDE. This exercise walks you through transferring and running a C shared library on your target, using the Eclipse IDE. In this exercise, you will replace the **Amplitude and Levels.vi** in **RT Main.vi** with a c library to perform the same task. The shared library has already been compiled for your target.

If you do not plan to reuse existing C code, feel free to skip this exercise. For information on reusing .m scripts in LabVIEW to quickly run them on real-time hardware, read [here](#). For information on importing existing VHDL/Verilog code into LabVIEW FPGA, read [here](#).

Transferring your shared library to the target

1. Open **cal_amp.c** from `.\3- Create Real-Time Application\Amplitude Shared Object`. The screenshot below shows the code in Eclipse that returns the amplitude of the waveform input.

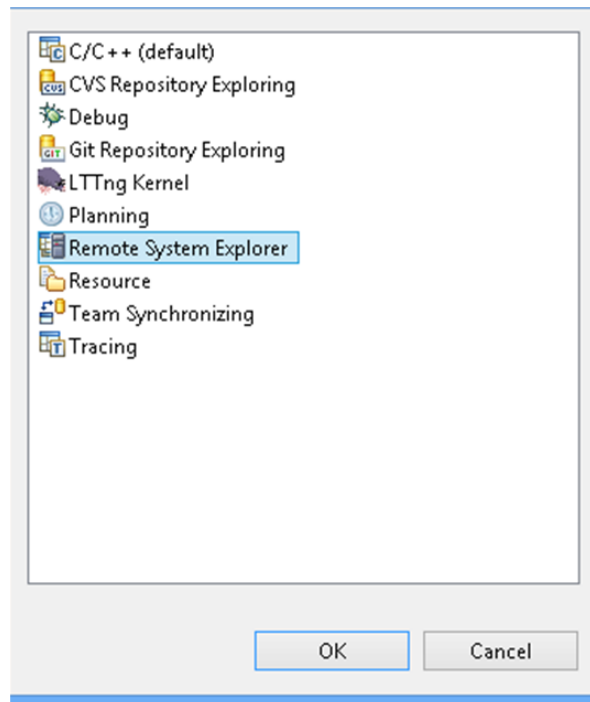
```
* cal_amp.c
#include <stdio.h>

double calc_amp(double waveform[], int length)
{
    double amp = 0.0;
    double max = waveform[0];
    double min = waveform[0];
    int i;

    // Iterate through array elements
    // calculate maximum and minimum value
    for (i=1 ; i<length ; i++)
    {
        if (waveform[i] > max)
            max = waveform[i];

        if (waveform[i] < min)
            min = waveform[i];
    }
    amp = max - min;
    return amp;
}
```

2. Before the C shared library can be accessed from LabVIEW on your eval kit target, the .so file will have to be present on the target. Place the file within the /usr/local/lib directory on the target so that it will be automatically loaded into memory upon startup. This can be done securely within the Eclipse environment as follows:
 1. Within Eclipse, select **Window»Open Perspective»Other...** and select **Remote System Explorer**.



3. On the Remote Systems tab, select **New»Connection** and choose **SSH Only**.
4. Type in the IP address of your target for Host name and Connection name and click **Finish**.

Remote SSH Only System Connection

Define connection information

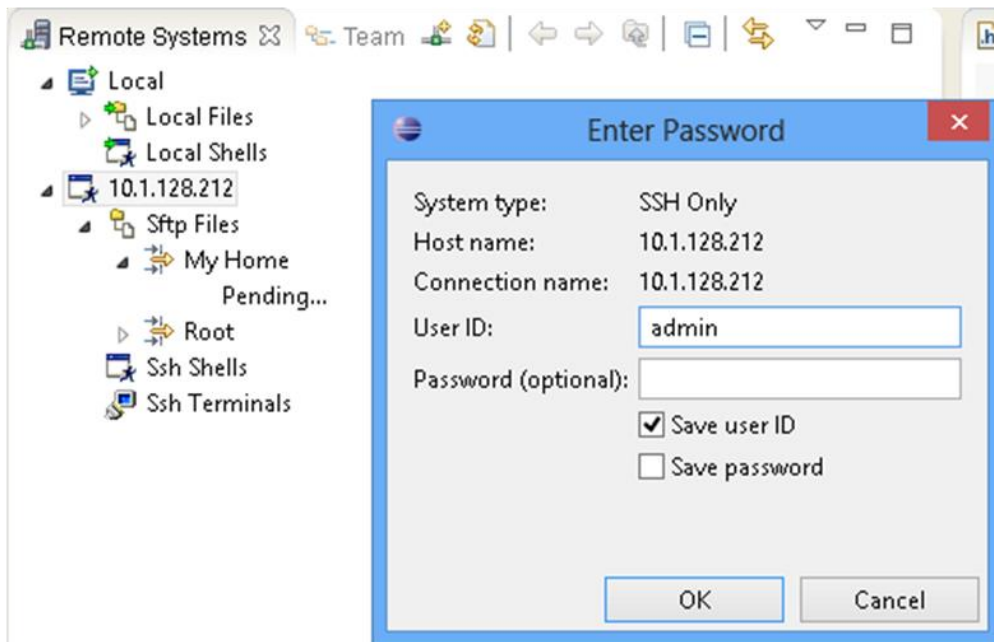
Parent profile:

Host name: 10.1.128.212

Connection name: 10.1.128.212

Description:

- Expand **Sftp Files** and a popup will appear to request a User ID and Password. By default, the User ID will be *admin* and the password will be blank. To learn how to set up user profiles and permissions for your target, read [here](#).



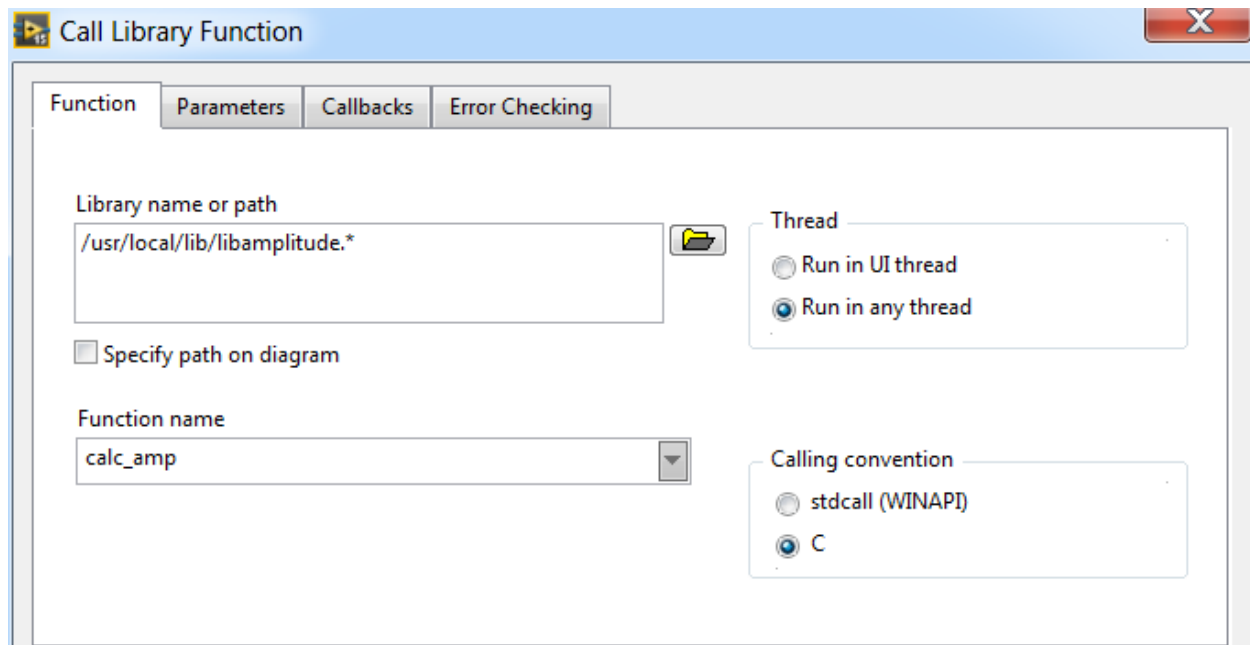
- Once you have logged in, expand the files under your target to find `/usr/local/lib`. Now, expand the files under your Local computer in the tree structure above to find your `.so` file. Drag it from the host computer to `/usr/local/lib` to transfer it to the target.
- Restart the target by pressing the reset button on the device next to the Ethernet port to load the shared object file.

Modifying your LabVIEW Real-Time VI

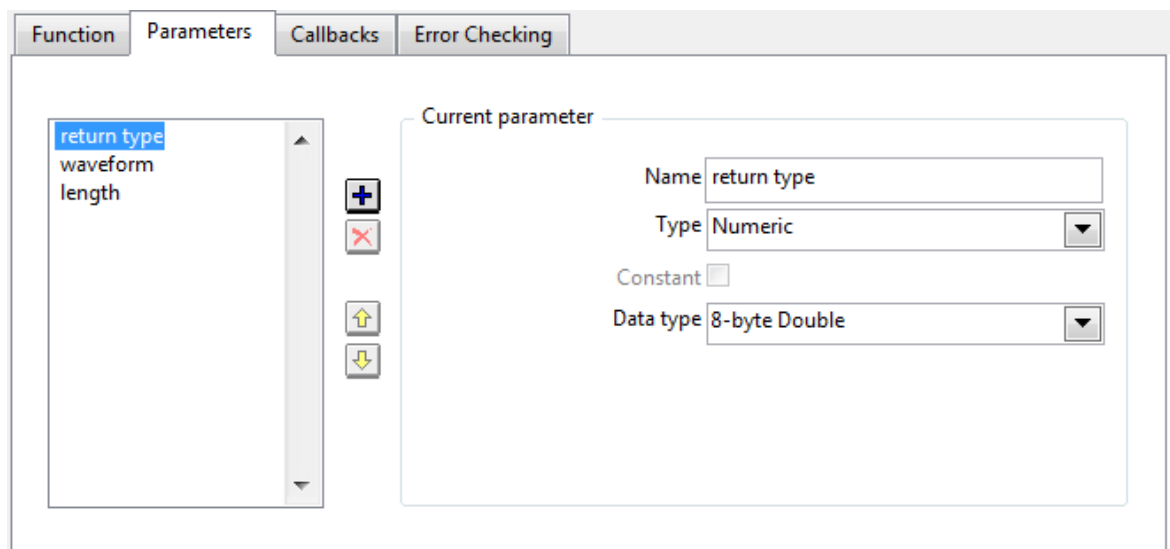
- Open **LabVIEW RIO Evaluation Kit.lvproj** and the **RT Main.vi** you completed in Exercise 3.
- Right-click on the block diagram to bring up the Functions palette and select the Call Library Function Node from **Connectivity»Libraries & Executables**. Double-click on the node to bring up the Call Library Function dialog box:



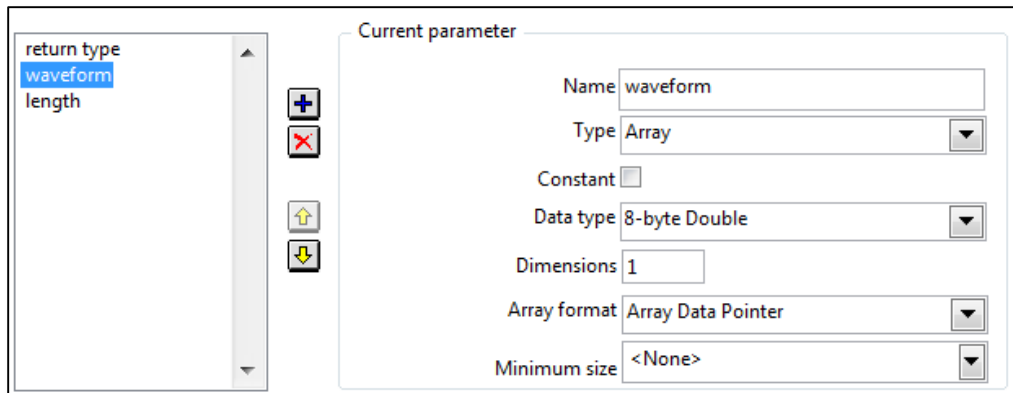
10. In the *Library name or path* box enter in `/usr/local/lib/libamplitude.*` with an asterisk as the file extension. Depending on what operating system the code is running on, LabVIEW will replace the asterisk with the appropriate extension. Note that if you are developing within the Windows environment, LabVIEW will be unable to populate the *Function name* drop down based on a .so file.



11. Select **Run in any thread** for the Thread option.
12. Configure the **Parameters** tab as shown below:



<See next page for more>



Current parameter

Name: waveform

Type: Array

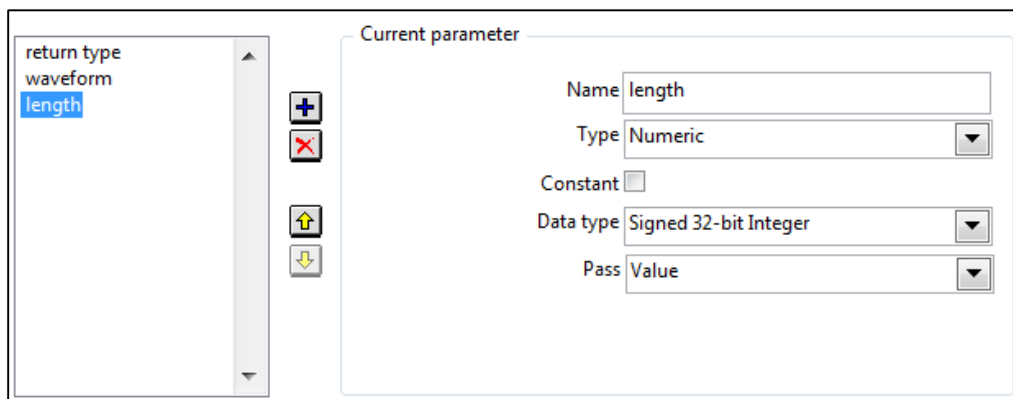
Constant: ☐

Data type: 8-byte Double

Dimensions: 1

Array format: Array Data Pointer

Minimum size: <None>



Current parameter

Name: length

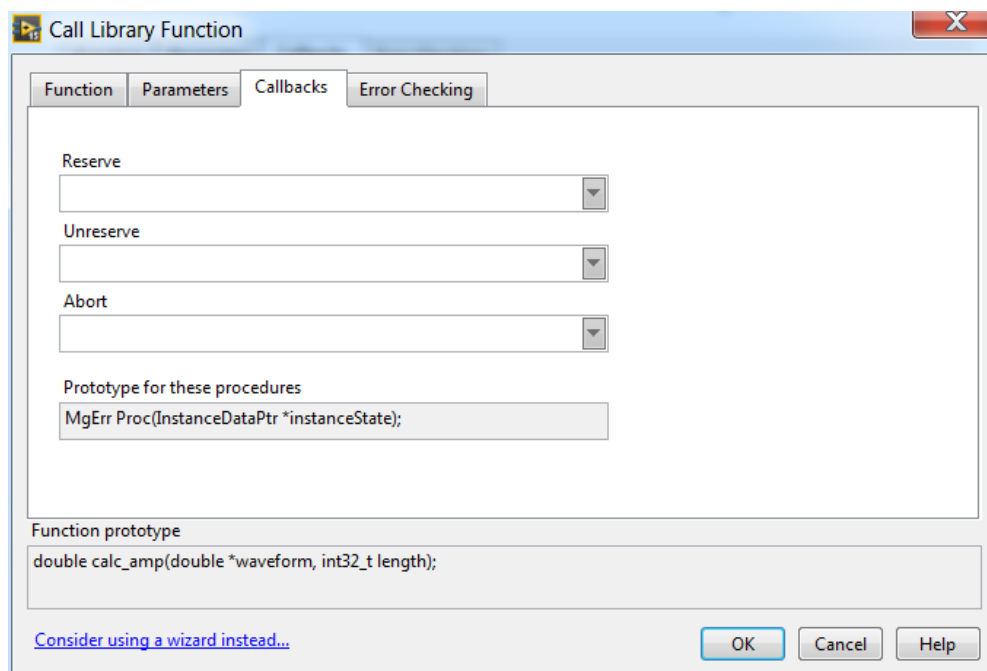
Type: Numeric

Constant: ☐

Data type: Signed 32-bit Integer

Pass: Value

13. Configure the **Callbacks** tab as shown below. Double-check that your Function prototype at the bottom matches what is shown.



Call Library Function

Function Parameters Callbacks Error Checking

Reserve

Unreserve

Abort

Prototype for these procedures

MgErr Proc(instanceDataPtr *instanceState);

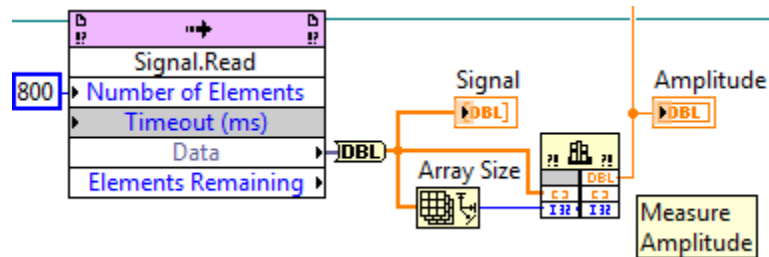
Function prototype

double calc_amp(double *waveform, int32_t length);

[Consider using a wizard instead...](#)

OK Cancel Help

14. Select **Default** in the **Error Checking** tab. Press OK to close the *Call Library Function* window.
15. Wire the Call Library Function node to replace the **Amplitude and Levels.vi** as shown below:



16. You can now press the run arrow at the top of the block diagram to deploy and run the code on your target.

Debugging

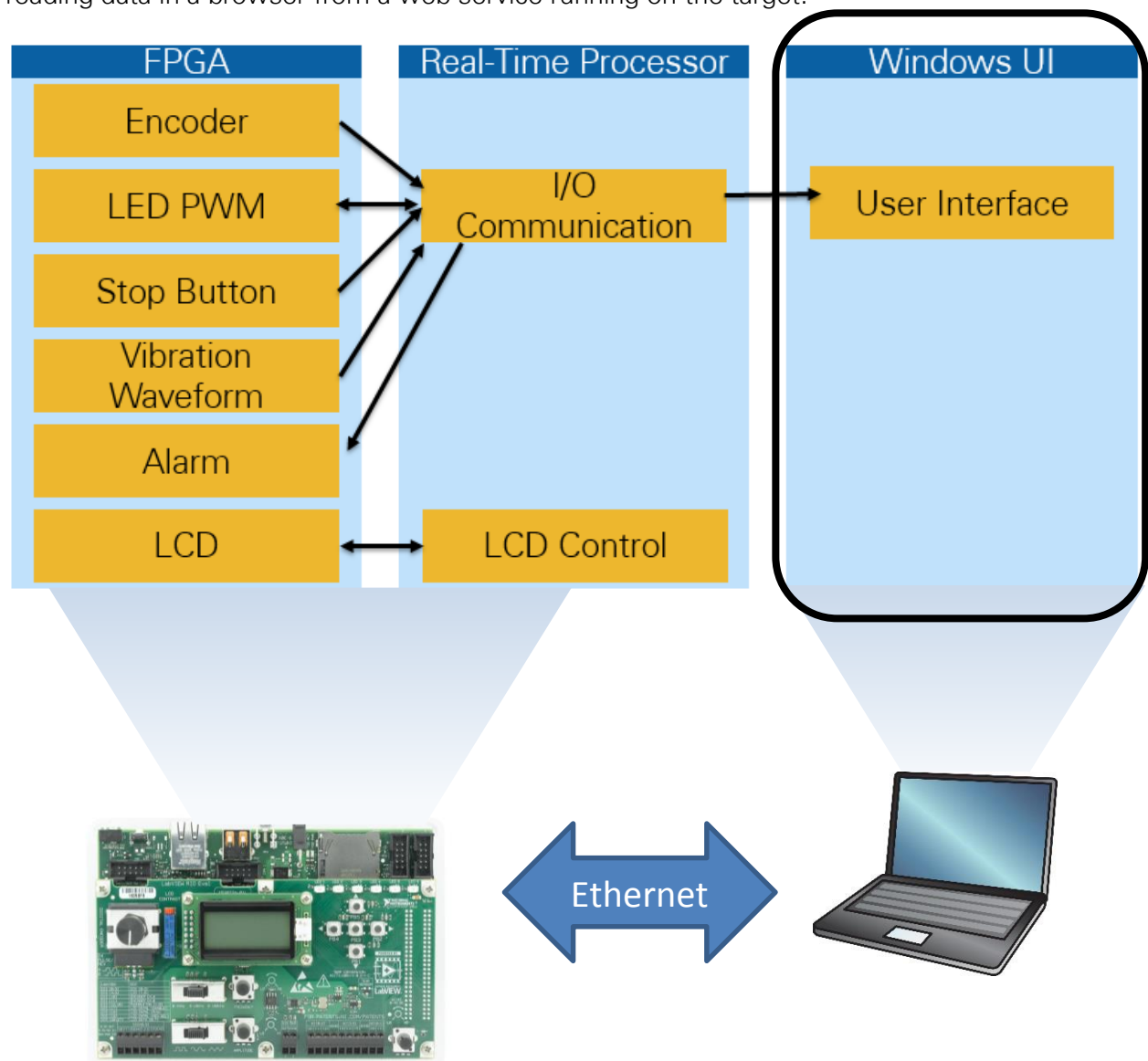
NI Linux Real-Time targets, like your evaluation kit, provide a GDB debug server to debug shared library code called by a LabVIEW VI. This can be done within the Eclipse environment or using the command line and an SSH Client. For more information when debugging your own shared library, read [Integrating C Code with LabVIEW on an NI Linux Real-Time Target](http://ni.com/rio/nextstep).

Exercise 4 | Create Windows User Interface

Summary

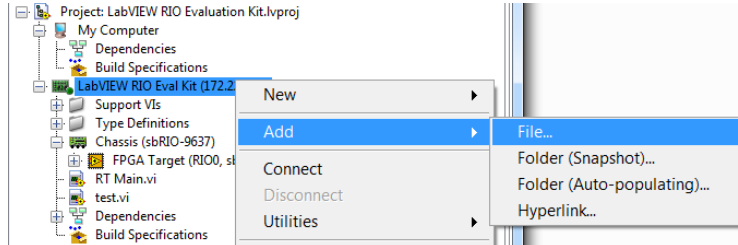
In this exercise you will finish the overall embedded system by completing a user interface running on your Windows development machine. The user interface components have already been placed for you, but the backend communication architecture needs to be completed so that you can update the user interface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

It is not recommended to use the RT VI front panel as your final user interface, because updating the UI thread takes up valuable resources. The evaluation kit does not have an integrated GPU or display port, so it is meant for true headless operation. The final UI can be as simple as the LCD, to a LabVIEW VI running on a Windows PC (Exercise 4), to a thin client reading data in a browser from a web service running on the target.

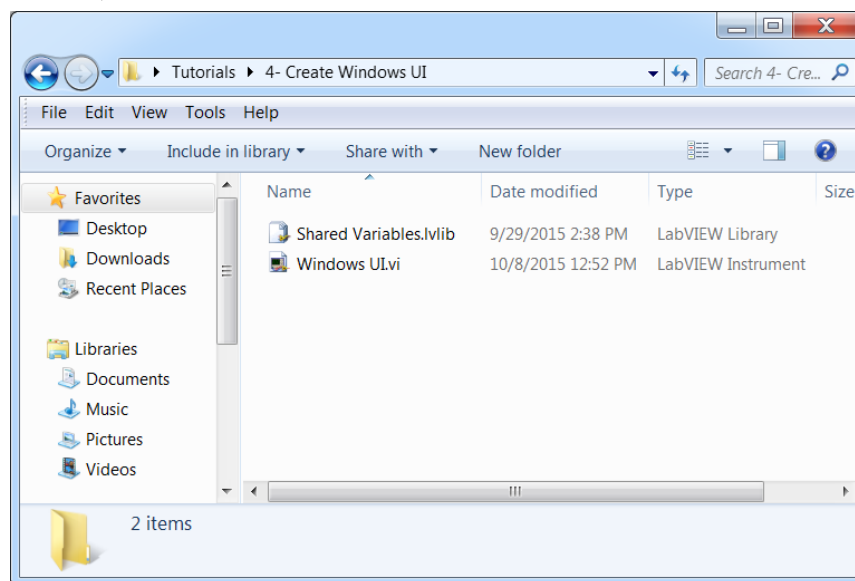


Implementation

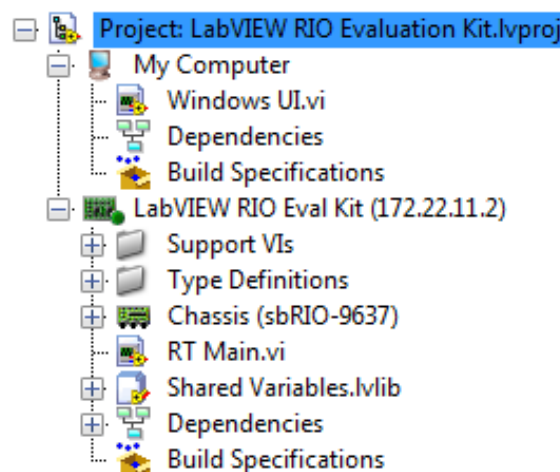
1. In the LabVIEW Project Explorer window, right-click on the *LabVIEW RIO Eval Kit* target and select **Add»File...** to add a shared variable library, which will communicate data between the RT VI and Windows User Interface VI.



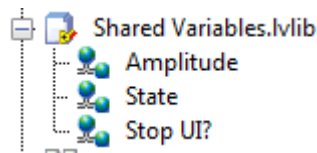
2. Navigate to the `. \4- Create Windows UI` folder and select **Shared Variables.lvlib**.



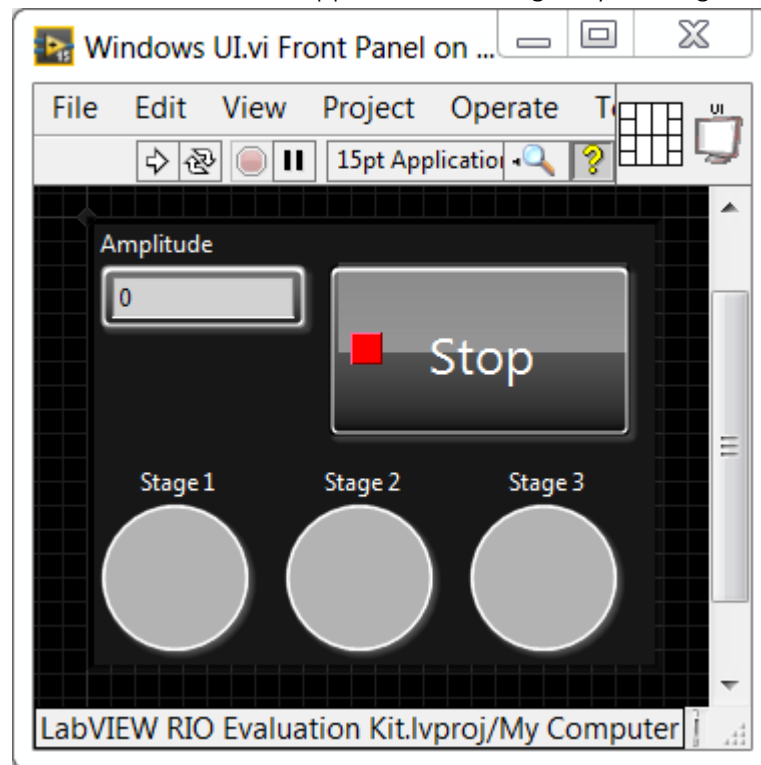
3. To add the Windows UI application already created, in the Project Explorer window right-click on *My Computer* and select **Add»File...** Navigate again to the `. \4- Create Windows UI` folder and select **Windows UI.vi**. Your project hierarchy should now look like this:



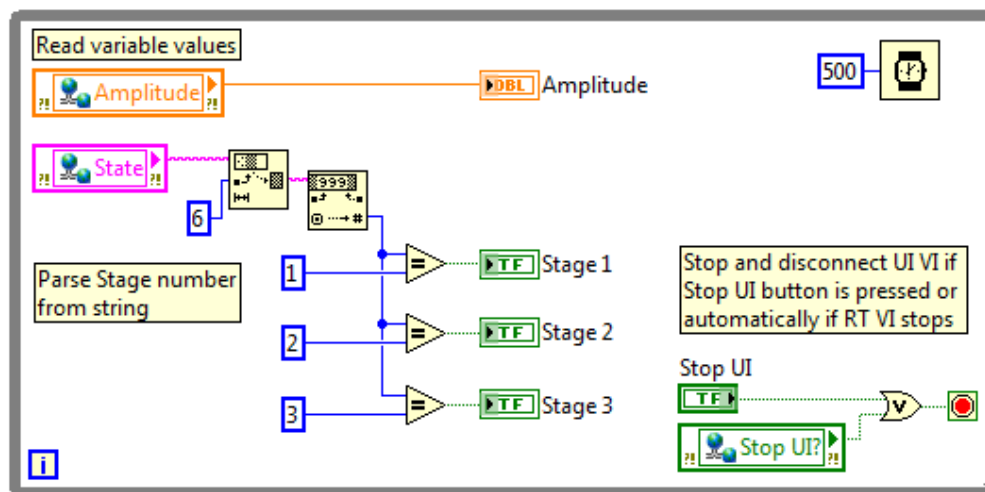
- Double-click to expand the Shared Variable library and note that there are three network-published shared variables which will communicate data between the Windows UI and RT VI over the network.



- Open **Windows UI.vi** and notice the Front Panel User Interface is simple, but has been customized. It will show the current stage and amplitude value, and provides a stop button to disconnect the UI from the application running on your target.



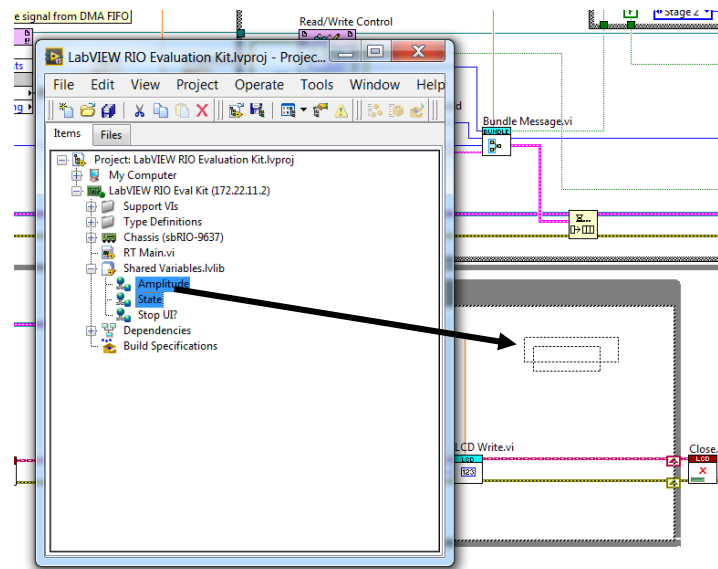
- Press <Ctrl-E> to open the block diagram and review the code.



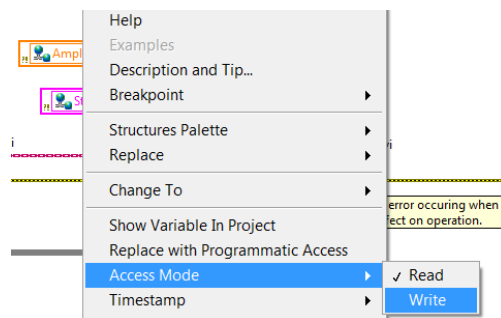
- The **Amplitude** and **State** shared variables are reading data from the RT VI to populate the front panel. There is some logic to parse the Stage number from the string to then light the appropriate LED.
- The VI can be stopped by the front panel **Stop UI** button or when the RT VI stops, it will send a true value to the **Stop UI?** shared variable.
- The loop will run at 500 ms.

Finish Development of the Real-Time VI to Windows UI VI Communication

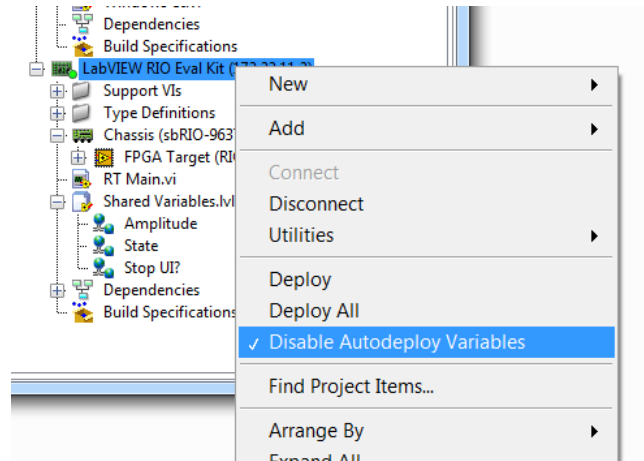
7. Open RT **Main.vi** and press <Ctrl-E> to open the block diagram.
8. From the Project Explorer window, press <Ctrl> to select both the **Amplitude** and **State** shared variables and drag them into the Consumer LCD Update Loop as shown below.



9. Separate the variables and right click on each, selecting **Access Mode » Write**

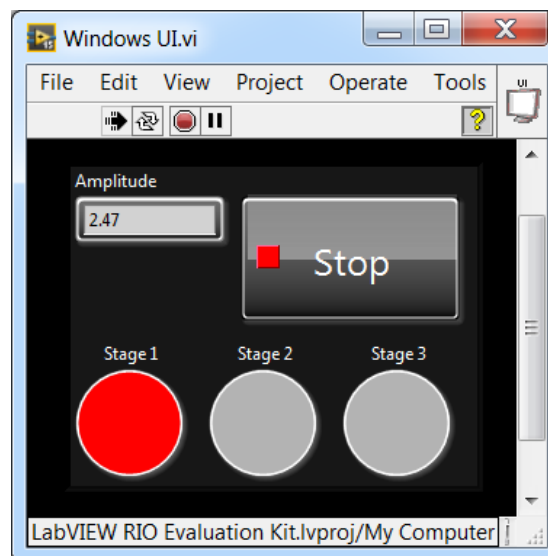


- Go back to the project explorer window. Right-click on the *LabVIEW RIO Eval Kit* target and select **Deploy All** to deploy the shared variables to the target. Right-click again and select **Disable Autodeploy Variables** which will prevent the Windows vi from trying to re-deploy the variables, since they are hosted on the RT target and will already be deployed along with the RT application.



Run and Verify the Completed System

- First click the **Run** arrow on the **RT Main.VI**, then click the **Run** arrow on the **Windows UI.vi**



- ✓ As you operate the part inspection machine through the stages, note how the Amplitude and LED indicators on the Windows UI front panel update.
- ✓ Check that when the RT VI stops, the Windows UI VI should as well.
- ✓ While the RT VI is running, press the Stop button on the Windows UI front panel and ensure your RT VI keeps running. Restart your Windows UI VI and ensure it reconnects with updated data.

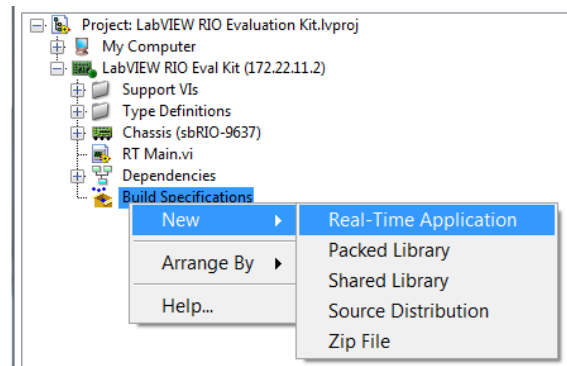
Exercise 5 | Startup Application

Summary

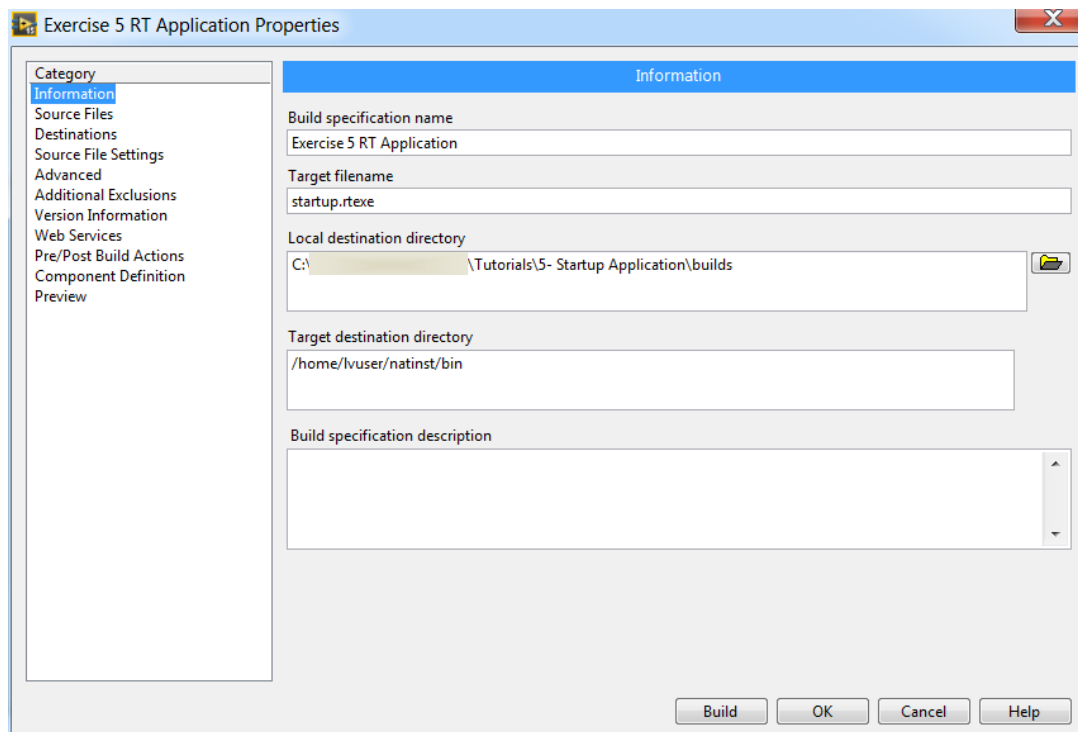
Now that you have completed the development of your embedded system, you may want to deploy the application to run standalone. In this exercise you will create a startup real-time executable.

1. Create and Deploy a Startup Real-Time Executable

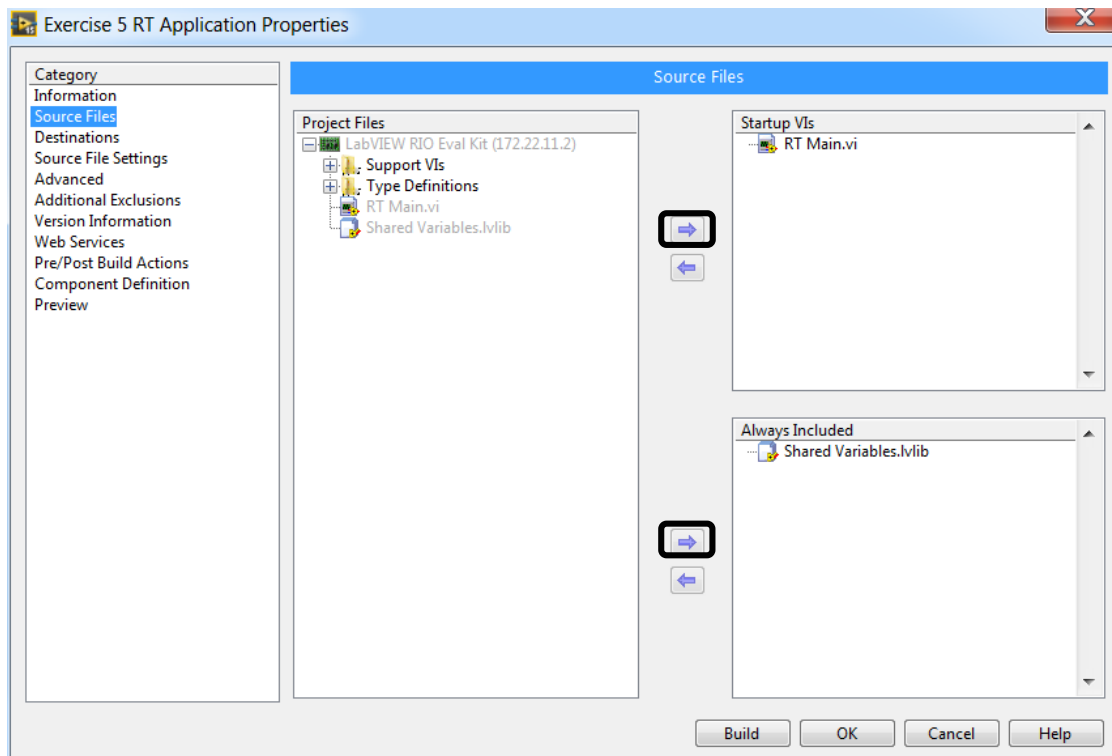
In the LabVIEW Project Explorer window expand out the *LabVIEW RIO Eval Kit* target, right-click on the bottom *Build Specification*, and select **New»Real-Time Application**.



2. In the dialog window, enter the Build specification name as *Exercise 5 RT Application*.
3. Set the Local destination directory as `.\5- Startup Application\builds`



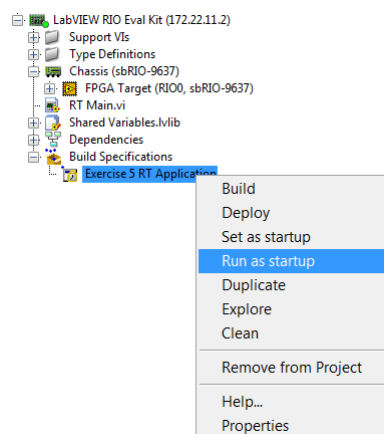
4. In the *Category* list on the left-hand side of the window, click on **Source Files**. Select **RT Main.vi** from the *Project Files* section and add the VI under *Startup VIs* by clicking the right arrow. Select **Shared Variables.lvlib** and click the right arrow to move into *Always Included*.



5. 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.

6. 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 any current real-time application that is still running on the device.

7. 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.
8. Allow for a minute for the Real-Time Operating System (RTOS) to boot up. Once it does successfully reboot, you should see AMP #.## on the screen and be able to operate the application normally.
9. 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. It should now execute, communicate with the headless real-time application, and update the LCD.
10. Run **Windows UI.vi** to view data from your RT application. Note that stopping the Windows UI does not stop the headless application.
11. Close all LabVIEW files and save if prompted.

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 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/embedded-systems.

Online Communities with Challenge 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 the LabVIEW RIO architecture, visit ni.com/rioeval/nextstep. Visit ni.com/linuxrftforum to find tutorials for NI Linux Real-Time, look for supported packages on the [NI Repository](http://ni.com/linuxrftforum), and ask questions on our active [discussion forum](http://ni.com/linuxrftforum).

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, R Series, 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, R Series, 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

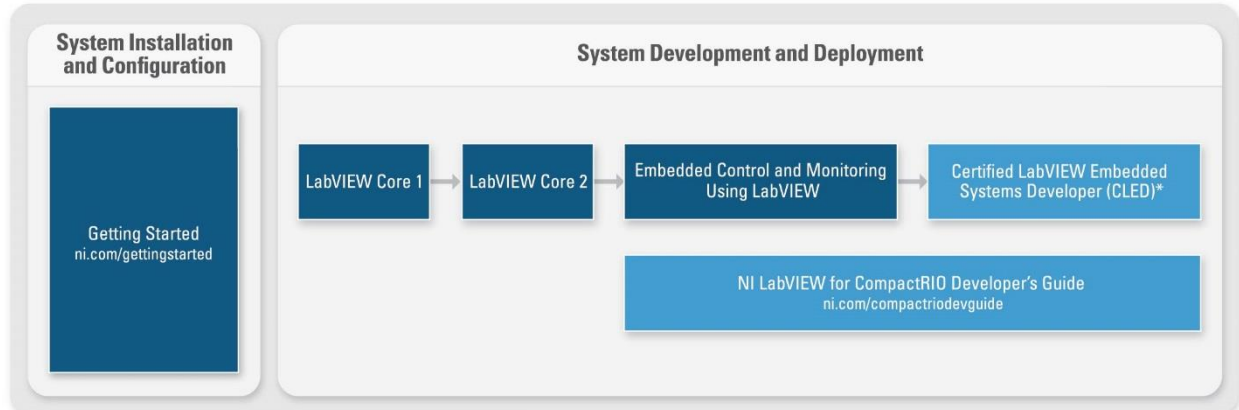
To begin with, everyone who uses LabVIEW and CompactRIO, R Series, or NI Single-Board RIO should have the ability to

- Install and configure RIO 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 - Core 3 and Embedded Control and Monitoring using NI LabVIEW classroom training courses (ni.com/training)
- LabVIEW Core 1, LabVIEW Core 2, LabVIEW Real-Time, and LabVIEW FPGA Self-Paced Online Training, free if you have an active service contract (ni.com/self-paced-training)
- LabVIEW for CompactRIO Developer's Guide (ni.com/compactriodevguide)

CompactRIO/Single-Board RIO Recommended Training Path



* A CLD or higher is required before attempting the CLED exam

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?

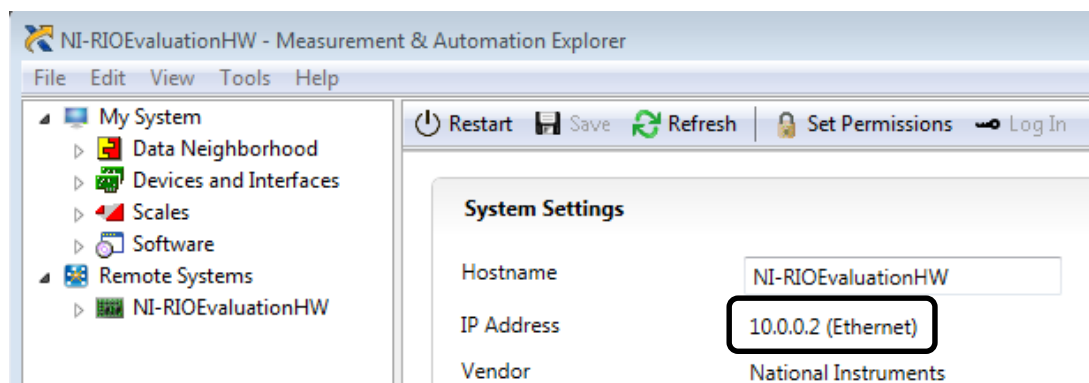
Many National Instruments Alliance Partners have already invested in the level of proficiency required for your application. If you have a CompactRIO or NI Single-Board RIO project that requires a greater skill level than you currently have and you are unable to gain the required level in the time allotted 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

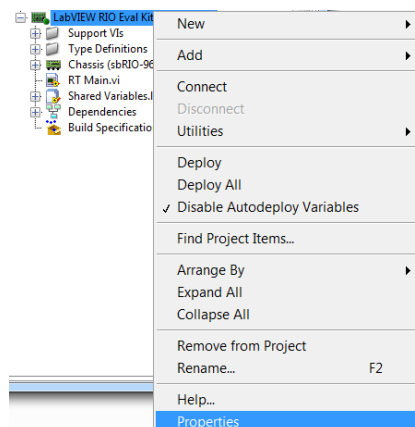
Your LabVIEW RIO Eval Kit is identified by its IP address. For each exercise, confirm that the IP address in the project matches the IP address of your device. The NI LabVIEW RIO Evaluation Setup utility should have prompted you to write down the target's IP address, but you also can locate the device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

If you are connected over the USB Host-to-Host cable, your IP Address is **172.22.11.2**.

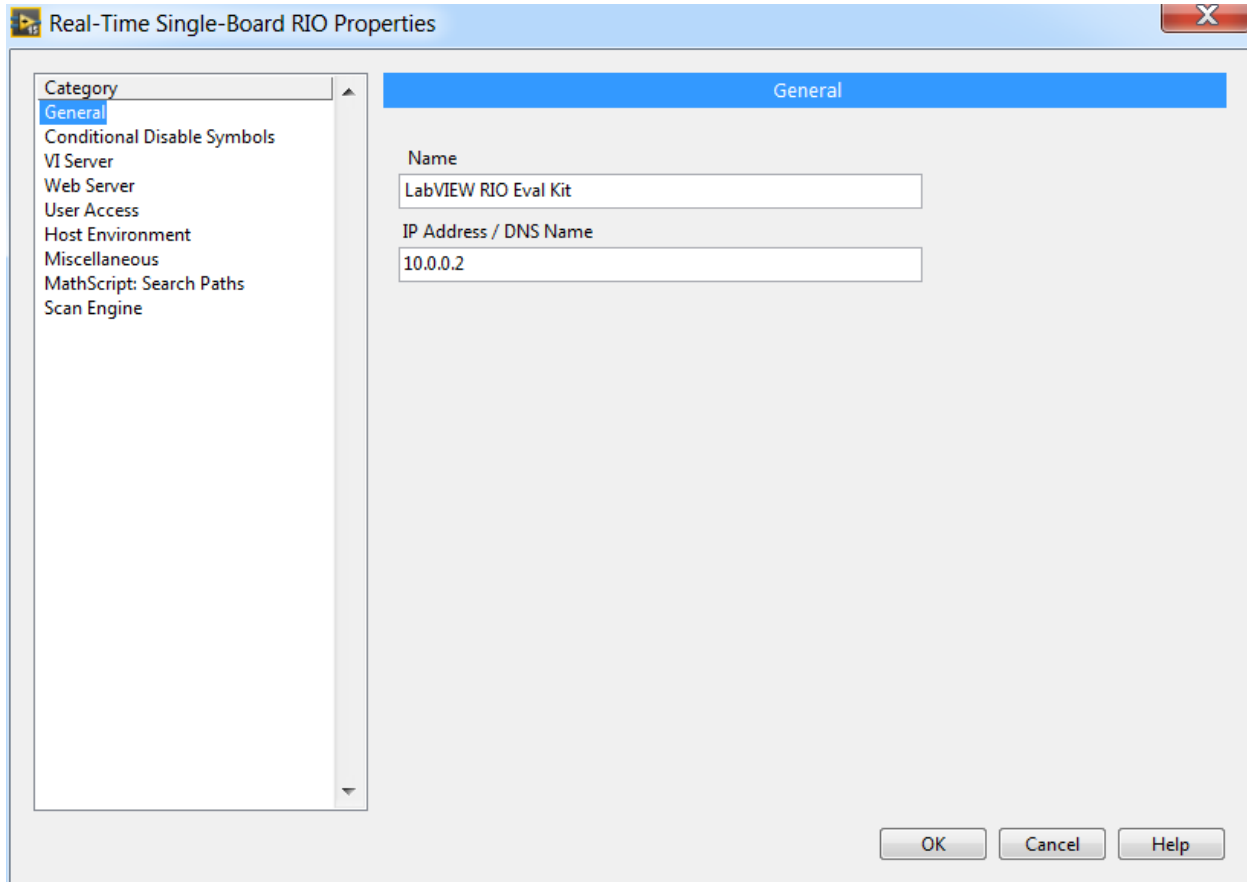
1. If you are connected over Ethernet, determine the IP address of your device by opening NI MAX (Measurement & Automation Explorer) located at **Start»All Programs»National Instruments»NI MAX**.
2. Click the triangle next to Remote Systems.
3. Click on your device in the Remote Systems tree and on the right hand side note the IP address that appears in the System Settings tab.



4. Change the IP address of the target in the LabVIEW Project Explorer window to match the IP address of your evaluation board.
 - a. Right-click the *LabVIEW RIO Eval Kit* target 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 *LabVIEW RIO Eval Kit* target in the Project Explorer window and select **Connect** to verify connection to the evaluation device.

If you cannot find your target in NI MAX, use the Remote System Discovery Utility or contact [NI Support](#).

