

## CompactRIO Motor Control Basics Tutorial

This hands-on session is an introduction to performing high speed FPGA-based position control of a brushed DC motor using National Instruments CompactRIO and the NI 9505 DC Servo Drive Module. You will create a new project and build a LabVIEW FPGA application from scratch that reads the position of the motor, performs proportional-integral-derivative (PID) control, and drives the motor using a pulse width modulation (PWM) signal. You will then deploy a LabVIEW Real-Time host application that takes advantage of the built-in remote panel web server to enable you to control and monitor the system using a web browser.

### Hands-On Summary

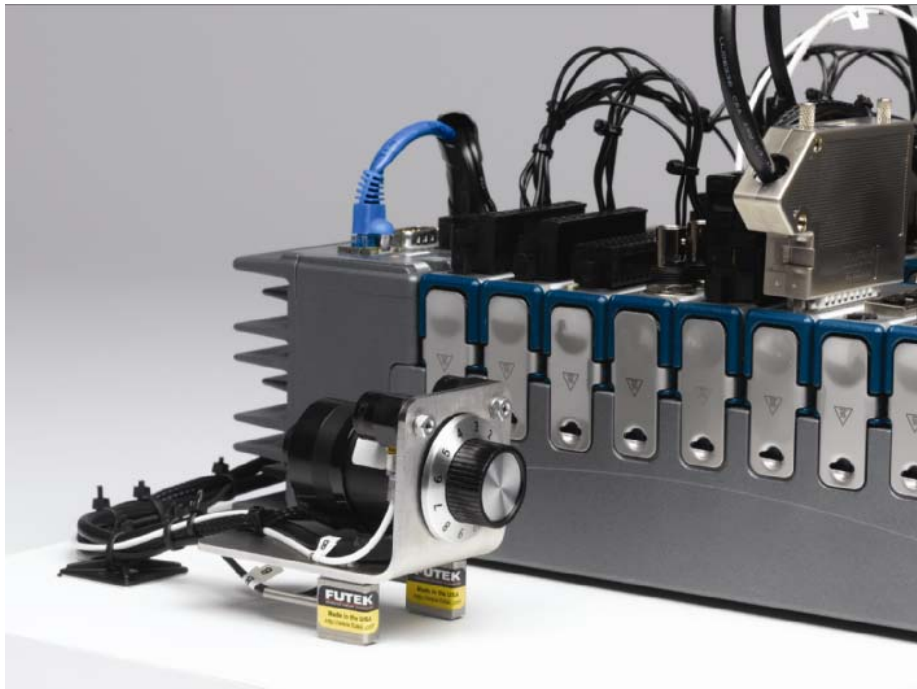
- Create a LabVIEW Project to manage your code and shared variable networking tags
- Create an FPGA application to process motor encoder signals and perform high speed closed loop control
- Run a precompiled FPGA and real-time processor application
- Use remote panel technology to control your embedded motor control system from a web browser

For more information on LabVIEW FPGA, CompactRIO, and NI motion control technology visit these pages:

<http://www.ni.com/fpga>

<http://www.ni.com/compactrio>

<http://www.ni.com/motion>



# Table of Contents

- Recommended Software and Hardware.....3
- Using the Target Control Application to Turn on Power to the System.....4
- Creating a New LabVIEW Project .....5
- Developing the LabVIEW FPGA Application .....17
- Running the Application.....25
- Controlling the System through a Web Browser .....28

## Recommended Software and Hardware

### Software

- [LabVIEW 8.20 Developer Suite Core](#)
- [Real-Time and FPGA Deployment Option for NI Developer Suite](#)
- [Industrial Monitoring Option for NI Developer Suite](#)
- [NI SoftMotion Development Module for LabVIEW](#)
- [NI-RIO Version 2.1 \(or later\) for R Series and CompactRIO Embedded Targets](#)

### Hardware

- [NI cRIO-9004 Real-Time Controller with 64 MB DRAM, 512 MB Compact Flash](#)
- [NI cRIO-9104 8-Slot, 3 M Gate CompactRIO Reconfigurable Embedded Chassis](#)
- [NI 9505 Full H-Bridge Brushed DC Servo Drive Module \(Install in Slot 5\)](#)
- [NI PS-2 24 VDC, 0.8 A, Power Supply \(Connect to NI 9505 module\)](#)

Or:

- [NI cRIO-9004 Real-Time Controller with 64 MB DRAM, 512 MB CompactFlash](#)
- [NI cRIO-9103 4-Slot, 3 M Gate CompactRIO Reconfigurable Embedded Chassis](#)
- [NI 9505 Full H-Bridge Brushed DC Servo Drive Module \(Install in Slot 4\)](#)
- [NI PS-2 24 VDC, 0.8 A, Power Supply \(Connect to NI 9505 module\)](#)

For help configuring your CompactRIO system and selecting accessories, use the [CompactRIO Advisor](#).

### Hardware Setup

Connect the MicroMo Electronics [3242 high precision coreless DC micromotor](#) to the NI 9505 drive module. Connect the encoder signals to the 9-pin female DSUB connector. Connect the PS-2 power supply and motor power leads to the 4-position screw-terminal connector.


### See Also:

[Learn more about the NI CompactRIO reconfigurable embedded control and acquisition system](#)

[Learn more about LabVIEW FPGA software](#)

## Using the Target Control Application to Turn on Power to the System

Follow the instructions below to make sure your National Instruments CompactRIO and CompactDAQ systems are powered up.

1. Double click the **Target Control** icon () on the desktop. To verify that system power is turned on, confirm that the green light is on. If the light is not on, click on the **Power Control** button.

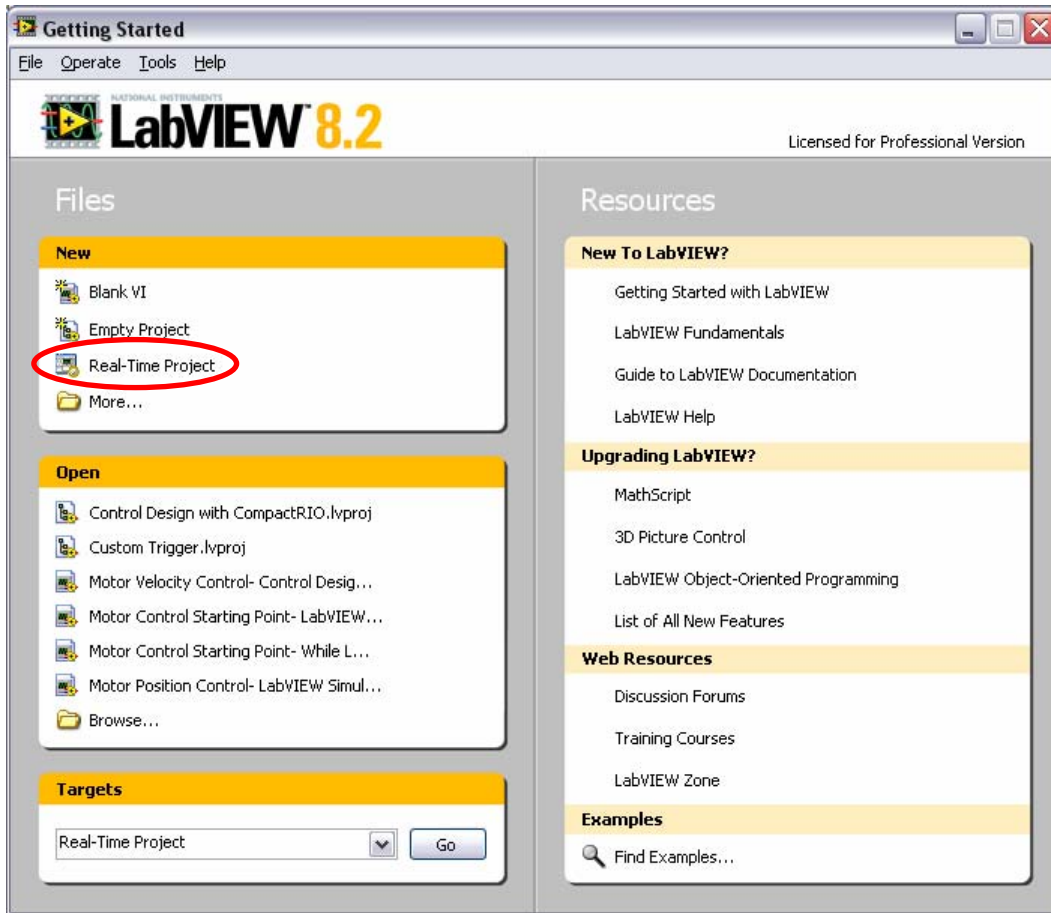


# Creating a New LabVIEW Project

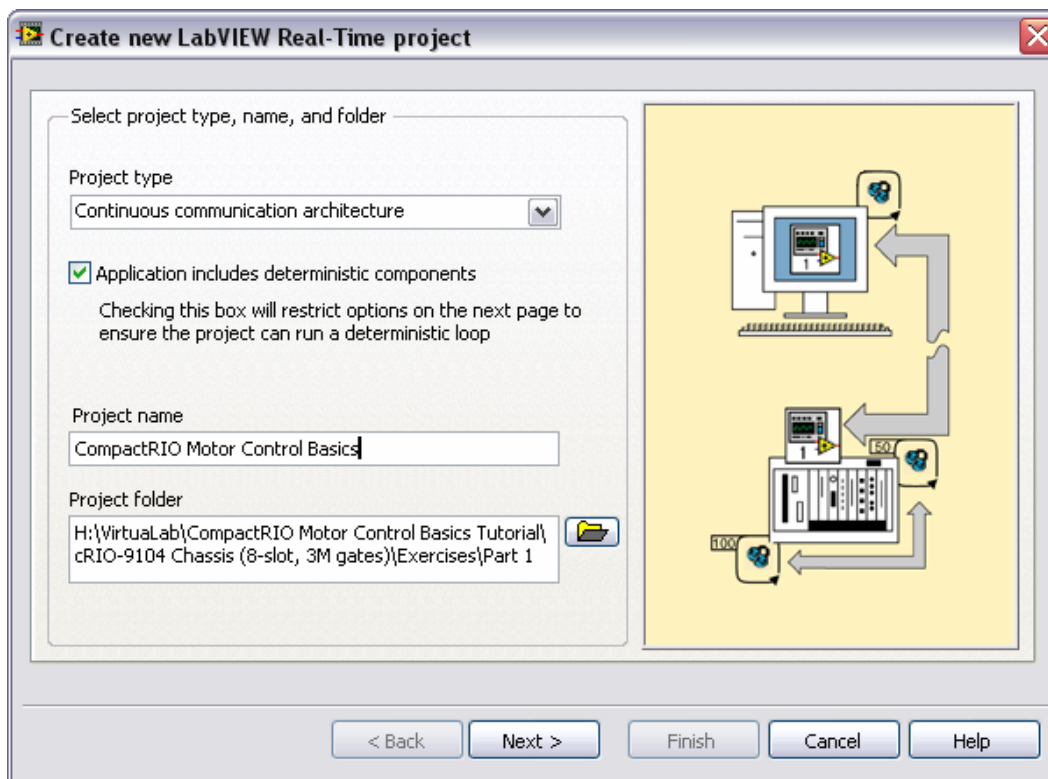
In this section you will learn the following:

- How to configure a CompactRIO system in the LabVIEW Project Explorer
- Auto-detection of the C Series I/O modules
- How to develop, target, download, and run an application on the reconfigurable FPGA

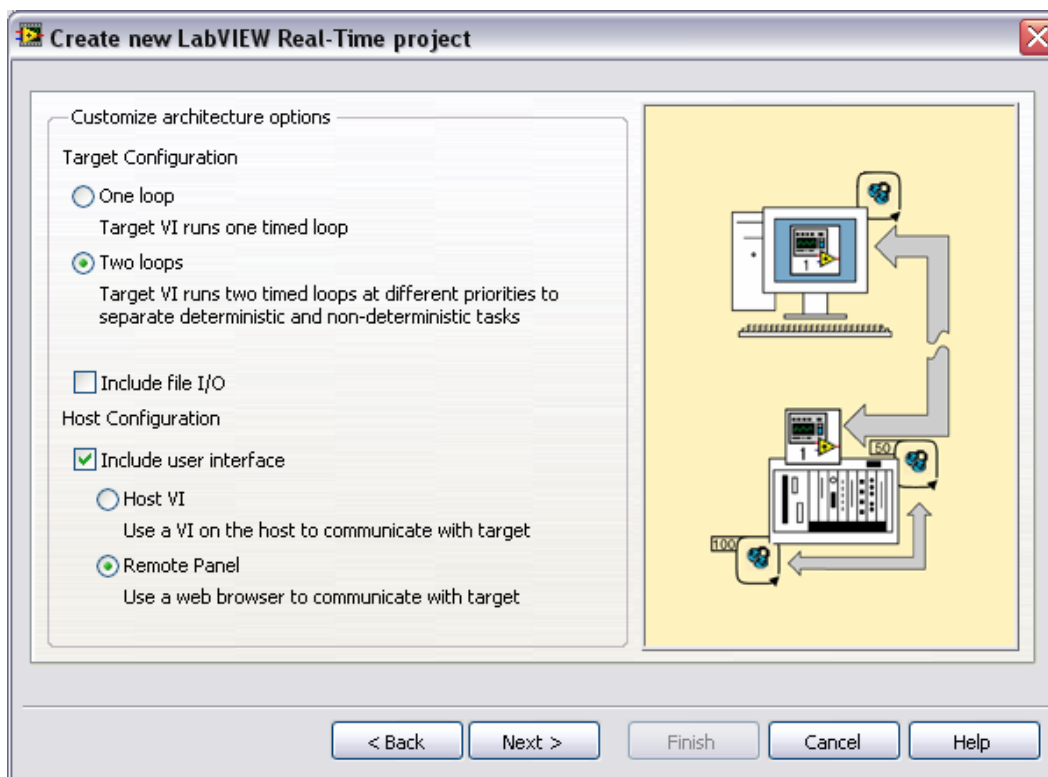
1. Launch **NI LabVIEW** by clicking on the desktop icon. Then click on the **Real-Time Project** link to start a new LabVIEW project for your NI CompactRIO system.



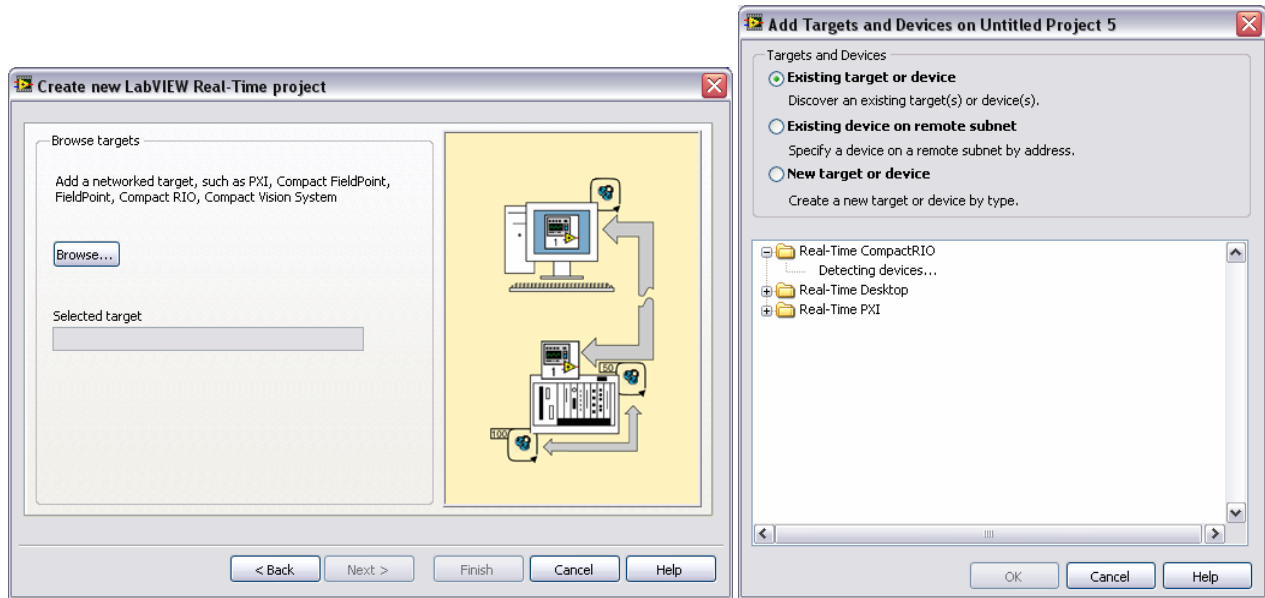
2. To select the working folder for the project, click the folder (📁) icon, navigate to **H:\VirtualLab\CompactRIO Motor Control Basics Tutorial\cRIO-9104 Chassis (8-slot, 3M gates)\Exercises\Part 1**, and then click the **Current Folder** button. Name your project **CompactRIO Motor Control Basics**. Keep the other default settings as shown below and click the **Next** button.



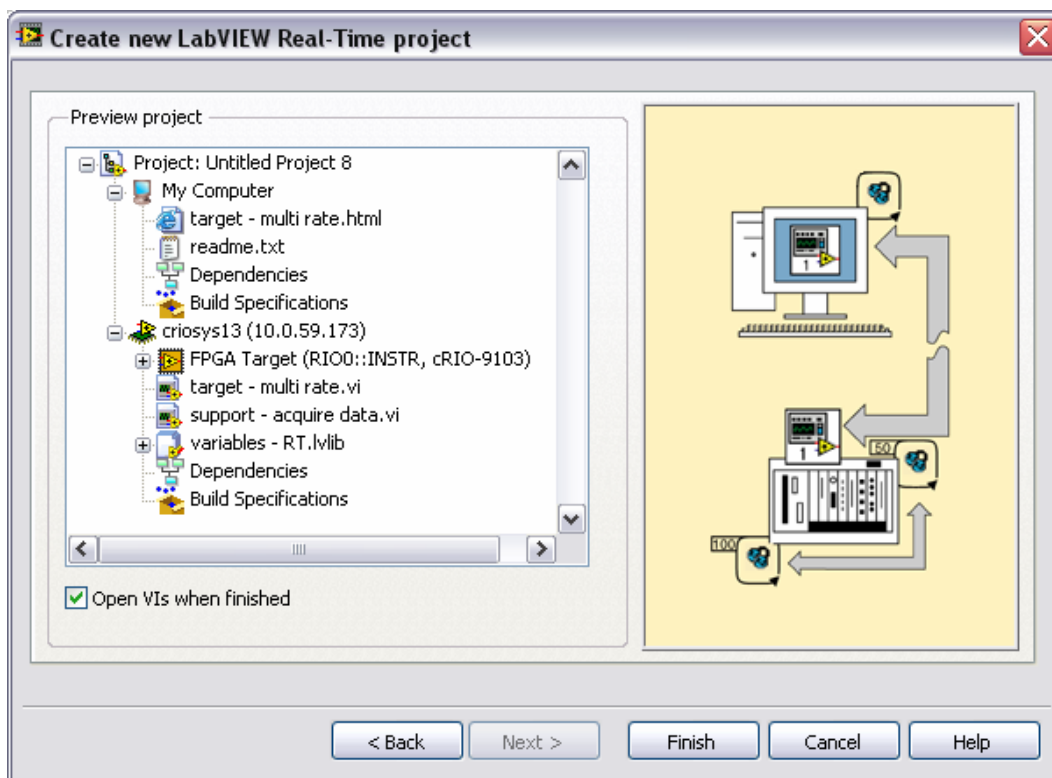
3. Change your **Target Configuration** to **Two loops**. Under the **Host Configuration** section, check the **Include user interface** box and then select the **Remote Panel** option. Then click **Next**.



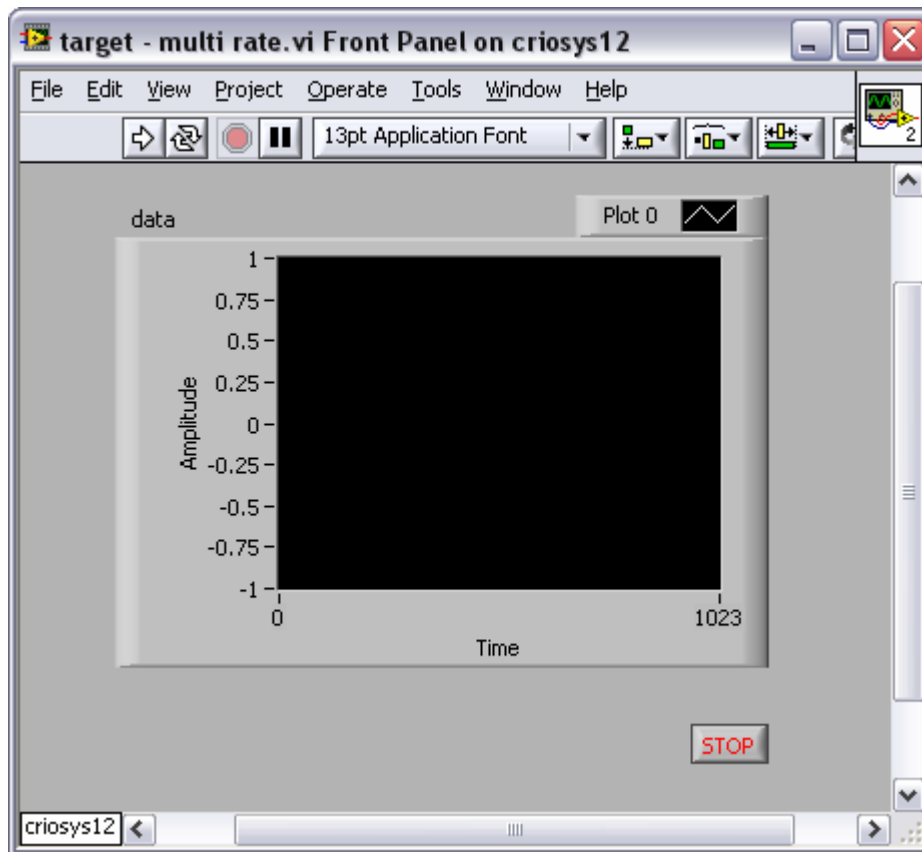
4. Click the **Browse** button to find the networked target you configured in MAX. Expand **Real-Time CompactRIO** folder and highlight the network name or IP address of your CompactRIO system and click **OK**. Then click **Next** to continue creating the real-time project.



- Notice that the project wizard displays a preview of the project you configured. Click **Finish** to finalize the creation of the new real-time project and generate the application template code.

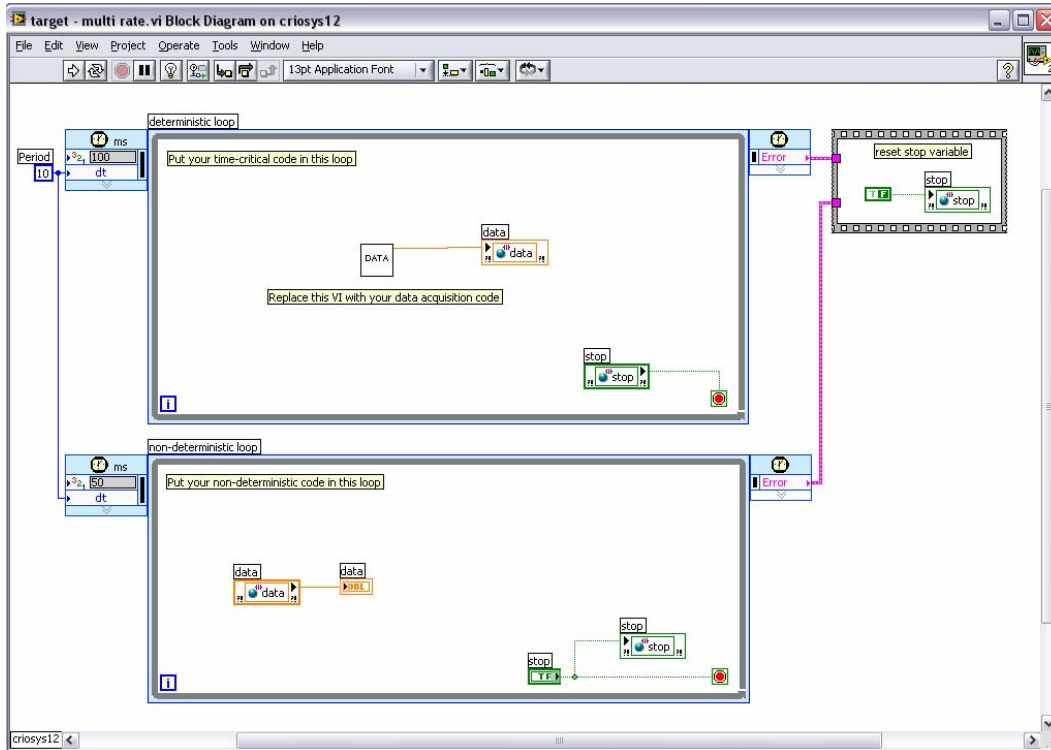


*When code generation is complete, a pre-built template application will automatically open.*



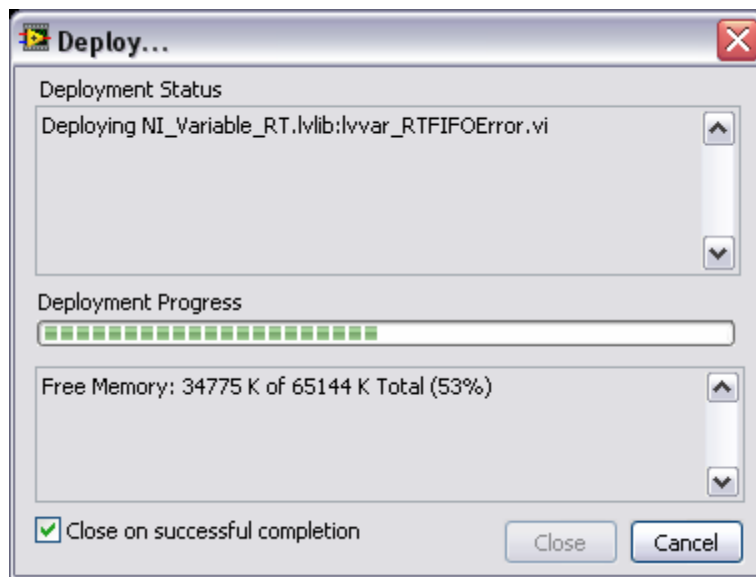
The real-time processor application (**target - multi rate.vi**) includes a chart to plot **data**, and a **stop** shared variable that is used to halt execution of the real-time embedded application.

6. In the **target - multi rate.vi** real-time processor application, navigate to **Window>>Show Block Diagram**.

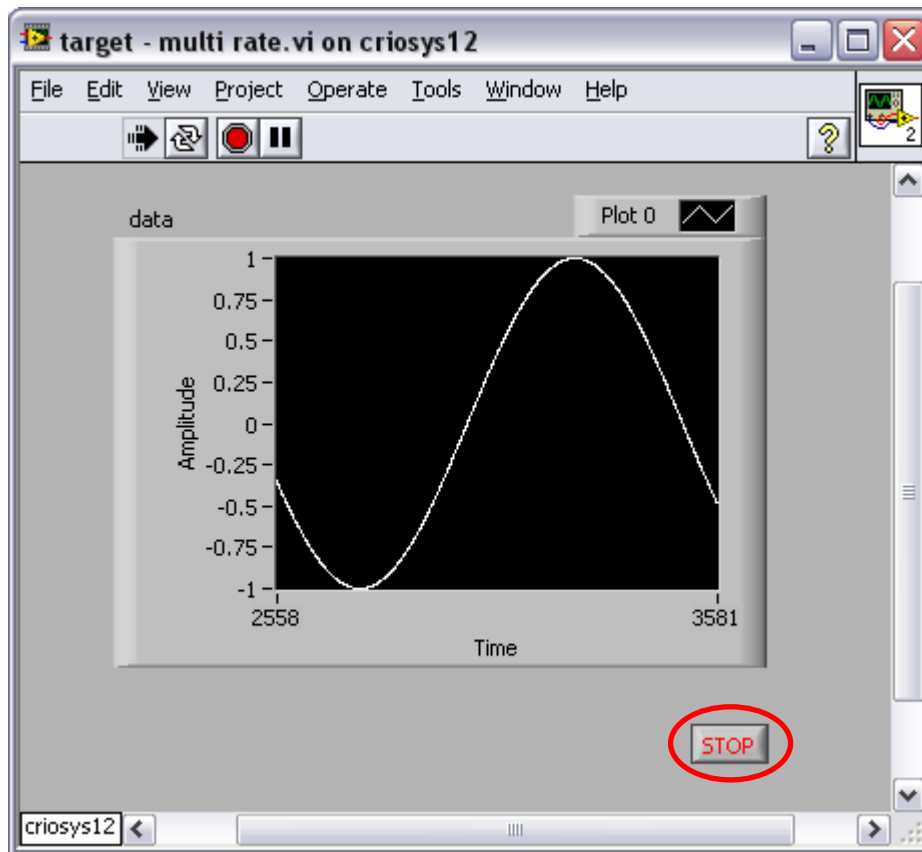


This embedded processor application produces a simulated I/O signal. You would place any time critical routines, such as code to interface with your FPGA application within the top deterministic loop, which is set to a priority of 100. Any lower priority non-deterministic tasks such as the user interface are placed in the bottom lower priority loop, which is set to a priority of 50. Notice that no user interface controls or indicators are used in the upper higher-priority loop.

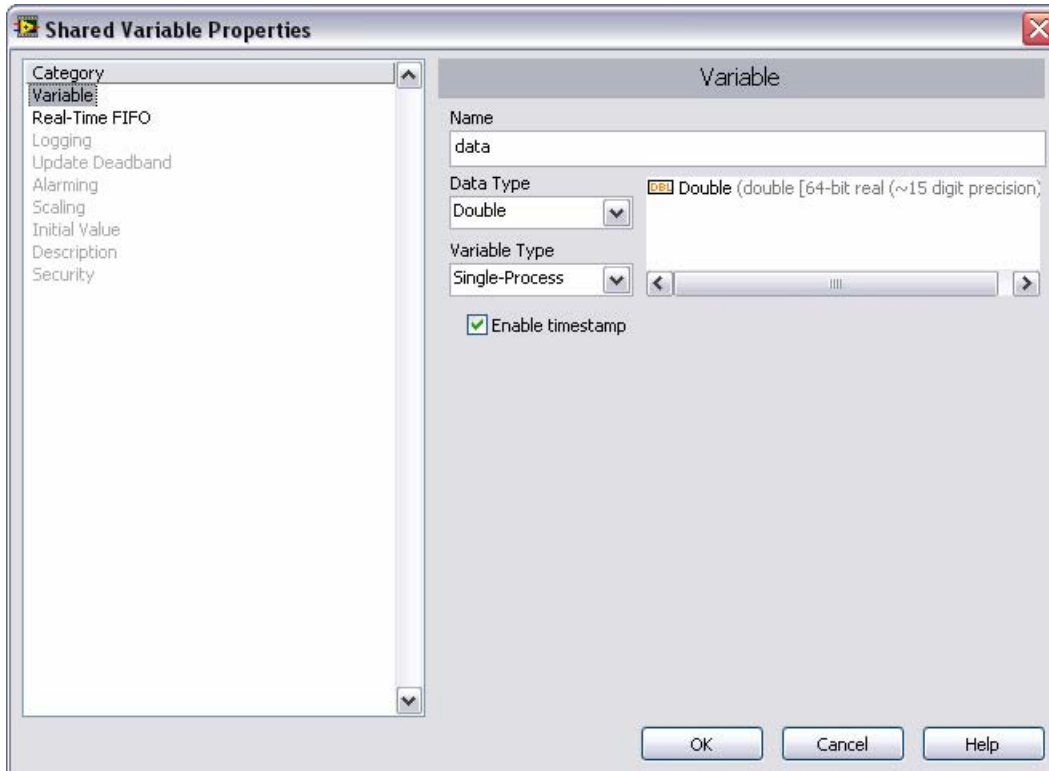
- Click the **Run** button (  ) on the real-time processor application ( *target - multi rate.vi* ). While the embedded application is being deployed, click the box to **Close on successful completion** if it is not already checked.



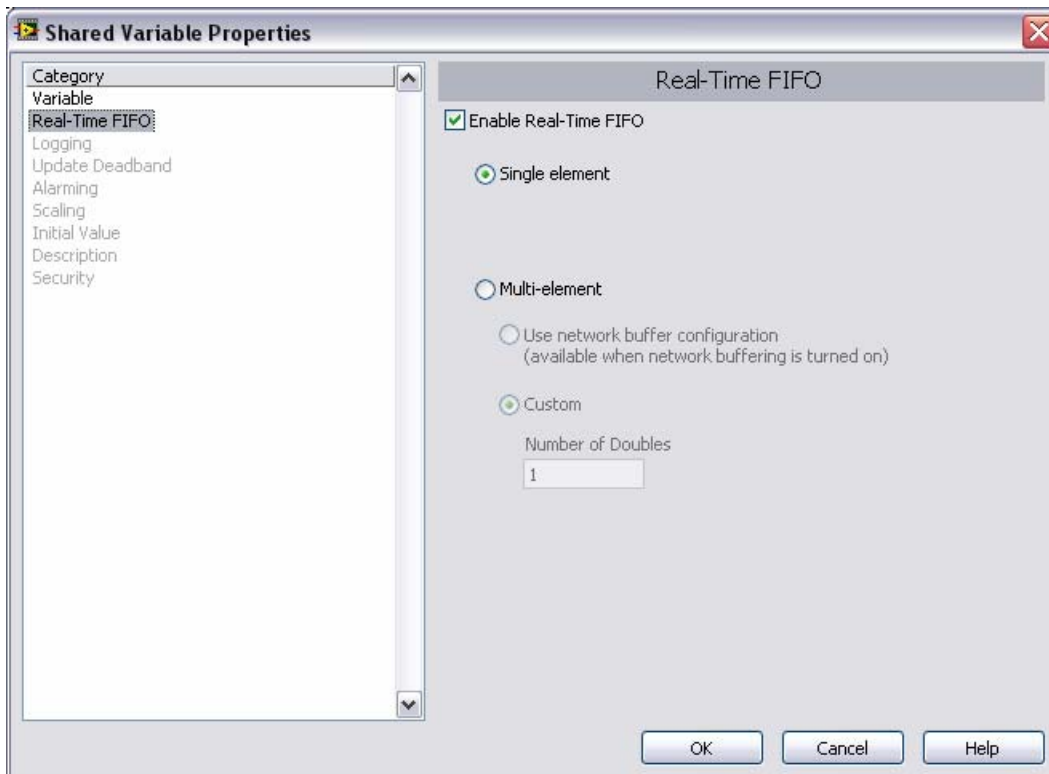
- View the sinusoidal waveform displayed on the chart. When you are finished, click the **STOP** button to stop the real-time target.



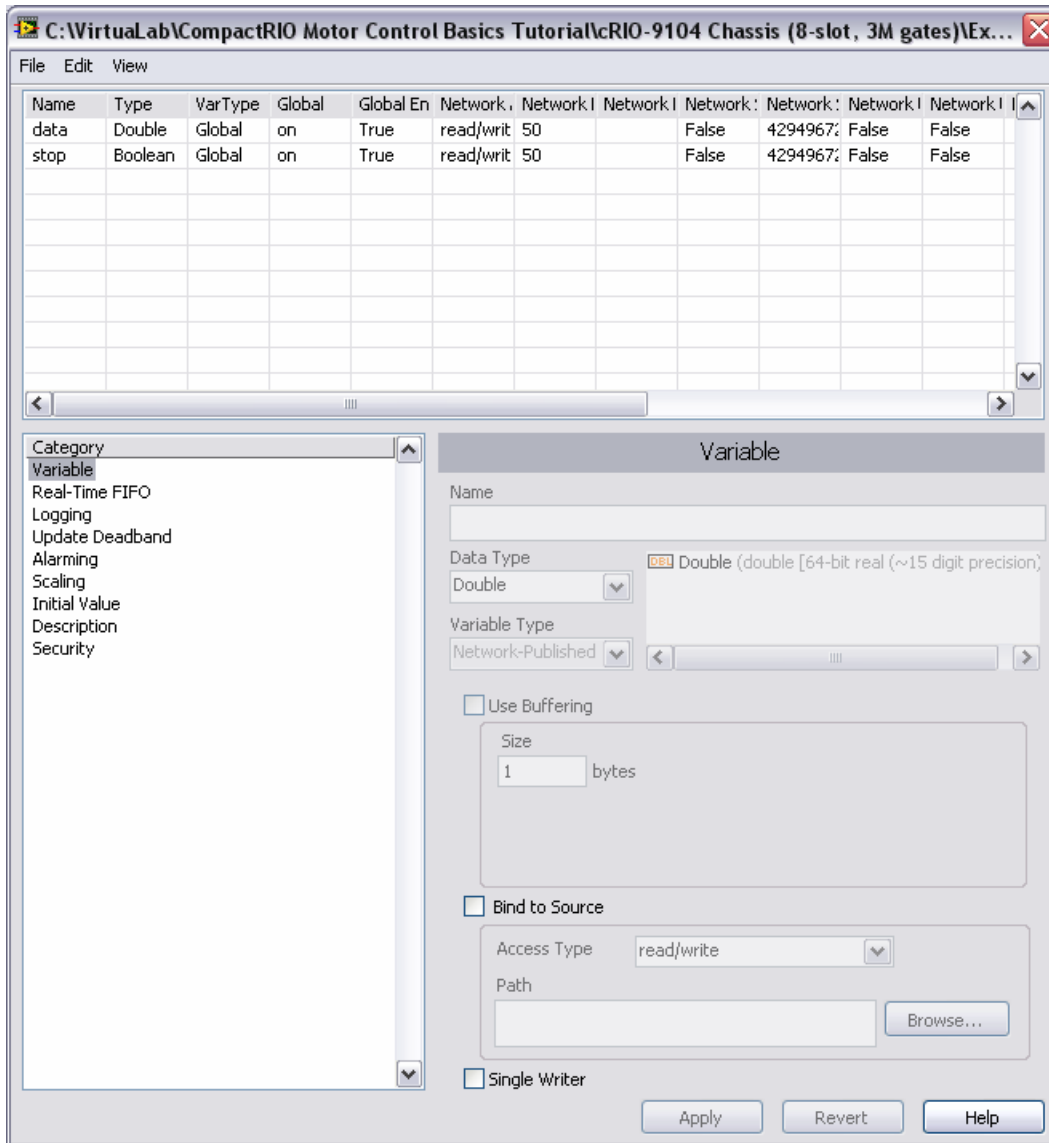
9. In the **Project Explorer** window under the folder **variables - RT.lvlib**, right-click on the shared variable named **data** and select **Properties**. Note that the **Variable Type** is set to **Single-Process** to enable the CompactRIO system to share data between loops in the embedded application.



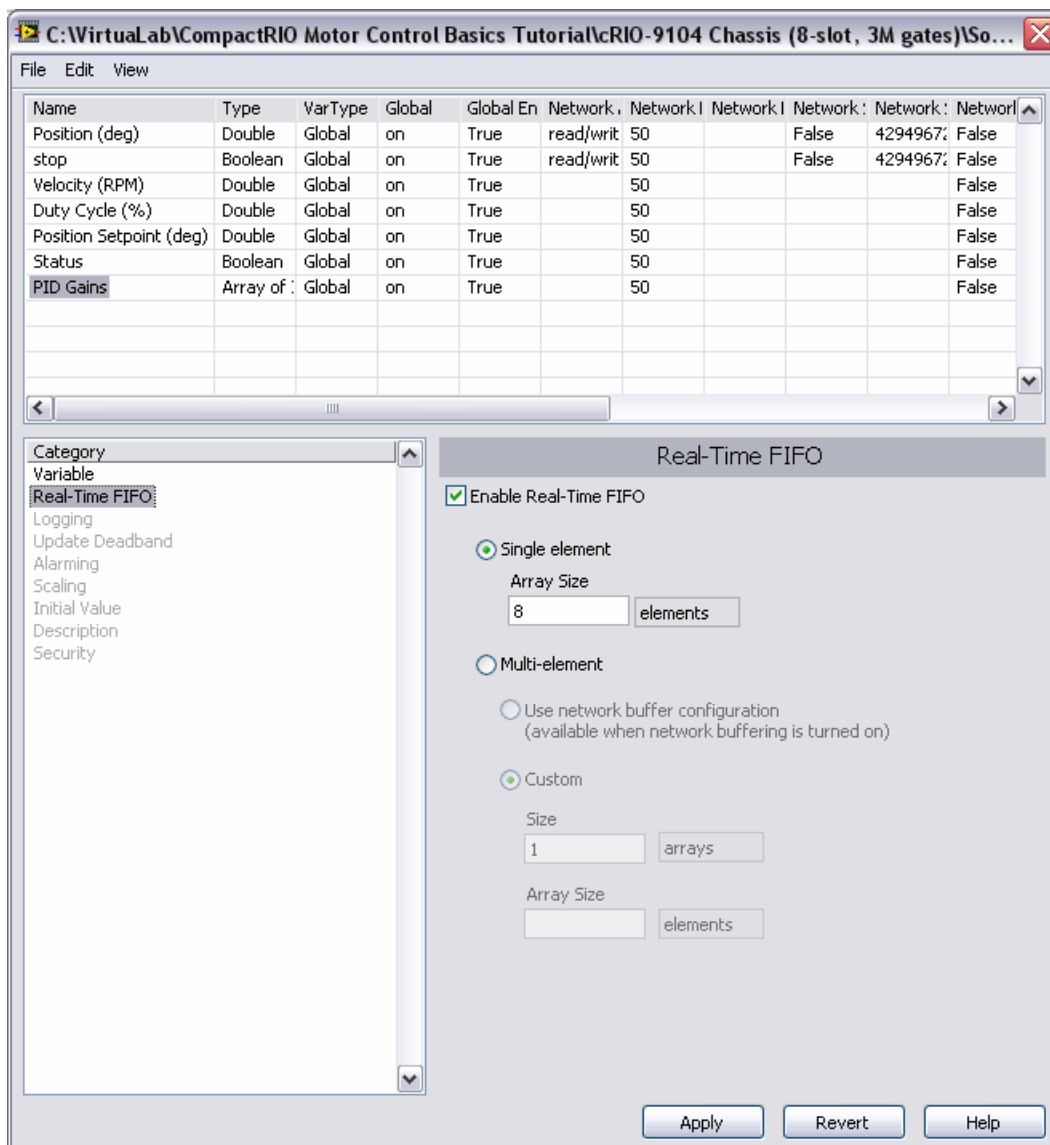
10. Click on the **Real-Time FIFO** page and note that the **Enable Real-Time FIFO** box is checked. This enables the high-priority deterministic loop on the CompactRIO real-time controller to communicate with the lower priority loop without introducing any timing jitter into the loop. Then click **Cancel** to close the shared variable configuration.



- Next you will add additional shared variables to the project that will be used by the motor control application. In the project, navigate to **Tools>>Shared Variable>>Multiple Variable Editor**. Then navigate to **File>>Open Project Library**, expand the tree and select **variables - RT.lvlib**.

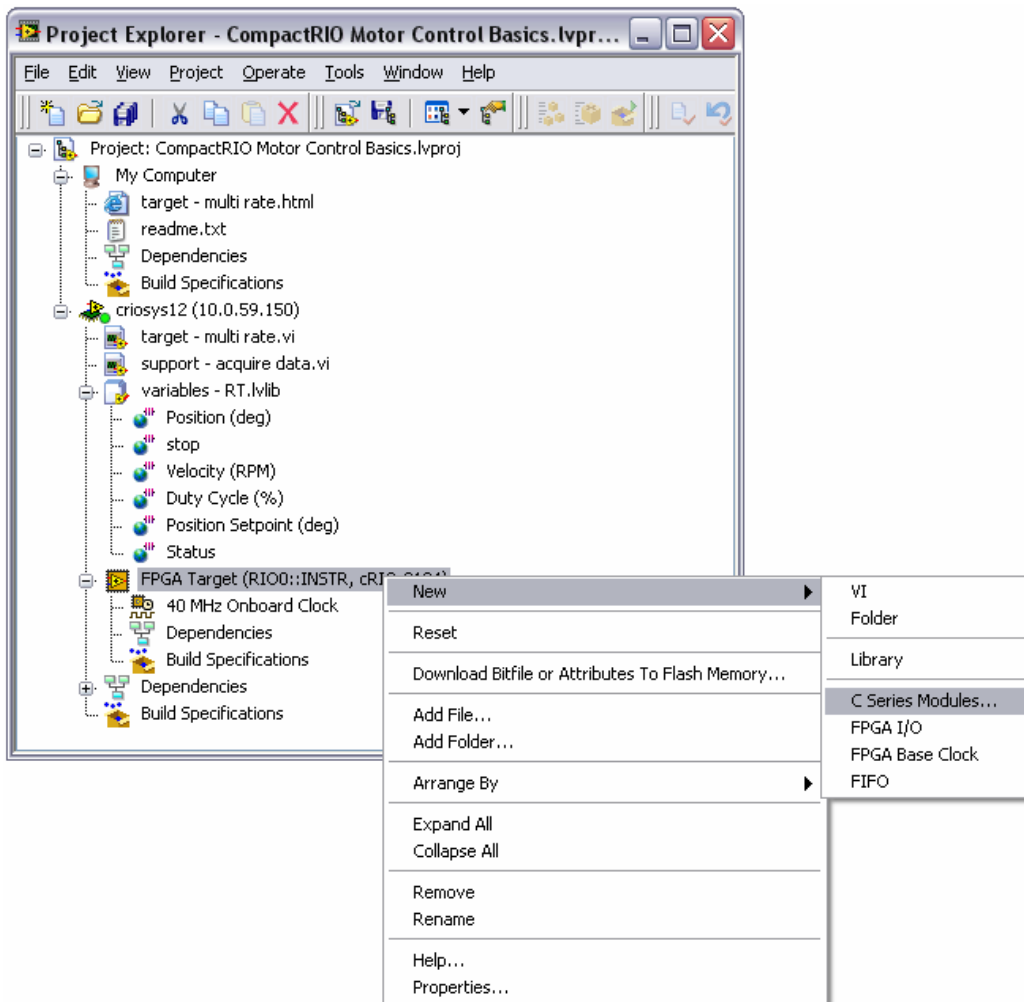


- Click on the variable named **data**, change the name to **Position (deg)**, and click **Apply**. Then navigate to **Edit>>Duplicate Variable** and change the name of the new variable that is created to **Velocity (RPM)** and click **Apply**. Repeat this process to create new shared variables for **Duty Cycle (%)** and **Position Setpoint (deg)**. Next, click on the **stop** shared variable, create a duplicate and name the duplicate **Status**.
- Next, navigate to **Edit>>Add Variable** and change the name to **PID Gains**. Then change the data type to **Array of Int16** and change the **Variable Type** to **Single Process**. Next, navigate to the **Real-Time FIFO** category, check the **Enable Real-Time FIFO** box, set the **Array Size** to **8** elements and click **Apply**.
- Finally, navigate to the File menu and save your shared variable library. Your library should appear similar to that shown below.

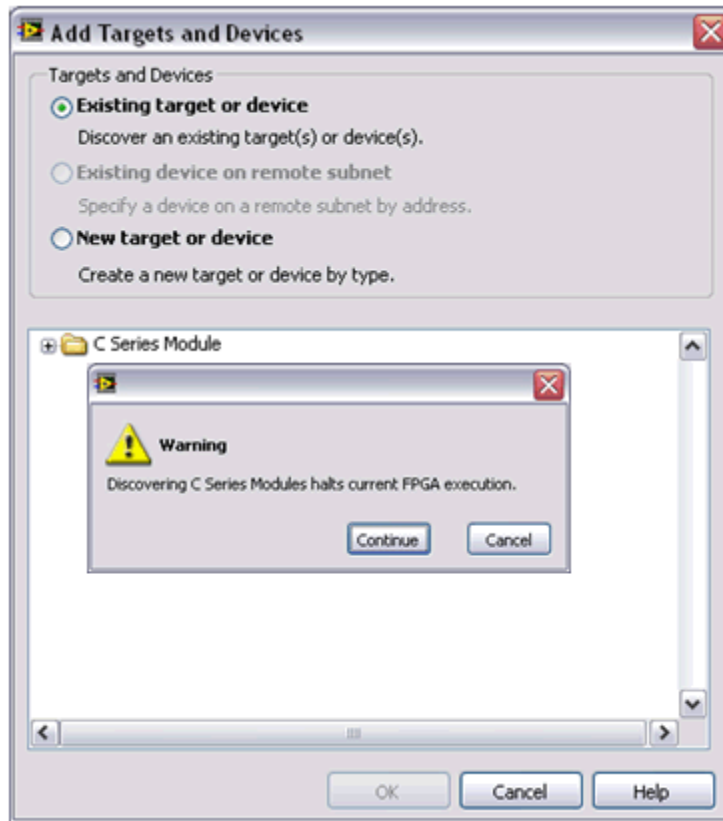


The Multiple Variable Editor included with LabVIEW Real-Time and the NI Datalogging and Supervisory Control (DSC) Module can help you manage large numbers of communication tags, including batch creation and modification of properties. To learn more, see the [LabVIEW 8 DSC application note](#).

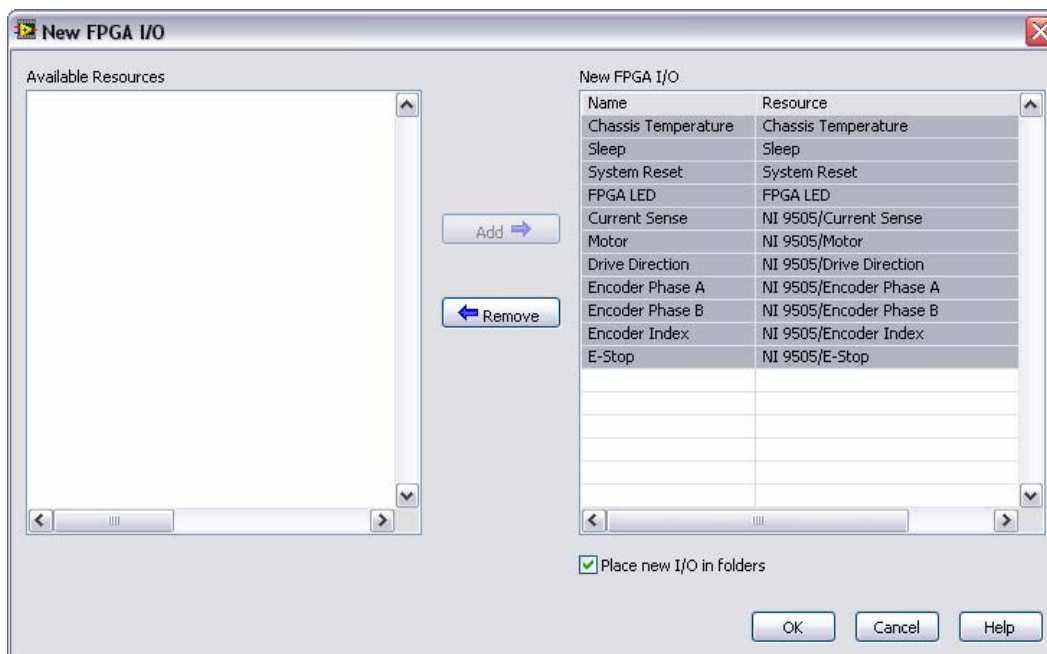
15. Navigate to **File>>Save** to update the project with your shared variable settings and then close the Multiple Variable Editor window by navigating to **File>>Close**. In the Project Explorer window, select **File>>Save All** to save the updated project and application code.
16. In the **Project Explorer** window, right-click on the **FPGA Target** and select **New>>C Series Modules** to add your I/O modules to the project.




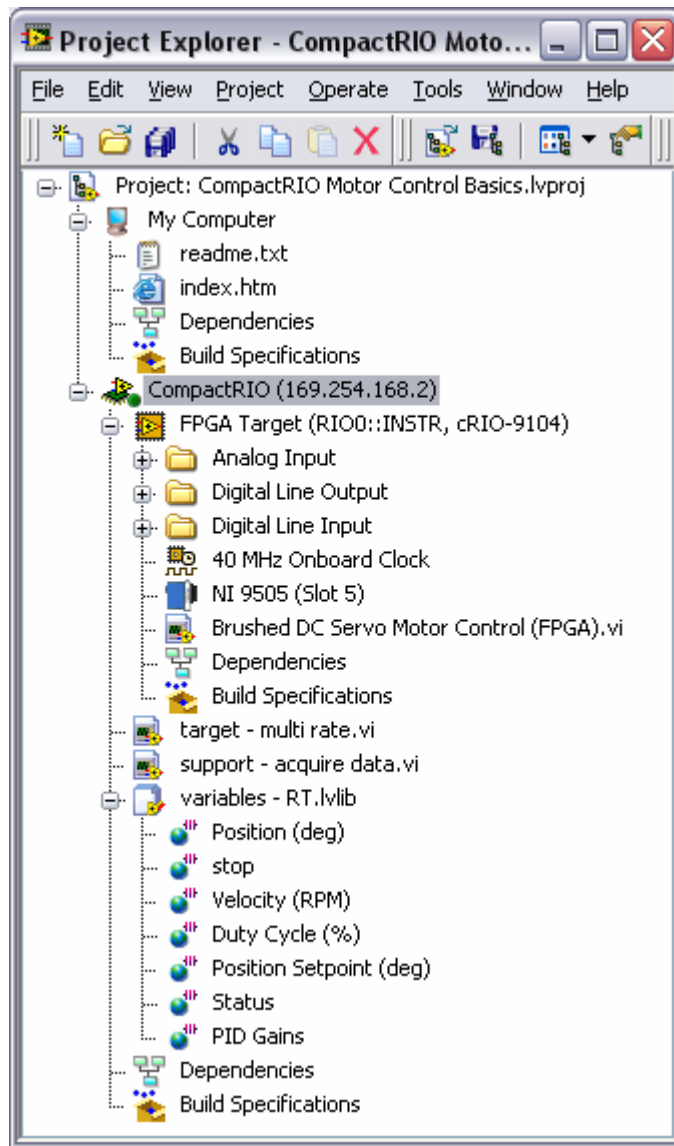
17. To automatically detect the I/O modules installed in your chassis, expand the **C Series Module** tab by clicking on the + symbol. Click **Continue** when the warning dialogue window appears. A pre-built FPGA bitstream will be downloaded to auto-detect the installed modules.



18. After the modules are detected, click on **NI 9505 (Slot 5)** and then click **OK** to add the motor drive module to your project.
19. In the **Project Explorer** window, right-click on the **FPGA Target** and select **New>>FPGA I/O** to add your I/O channels to the project. Highlight all of the sections by holding down the **Shift** button and clicking on the bottom category (**Digital Line Input**). Click the **Add** button and then click **OK** to add all of the I/O to your project.



20. If you collapse the individual I/O channel folders, your LabVIEW Project should appear similar to what is shown below. Click the **Save All** button (  ) to save the project and all subVIs.



The LabVIEW Project Explorer helps you manage the configuration of your CompactRIO system, including the built-in web server, I/O communication servers, FPGA and RT application software, I/O module configuration settings and more. To learn more, see the [Getting Started with the LabVIEW Real-Time Module Manual](#).

**See Also:**

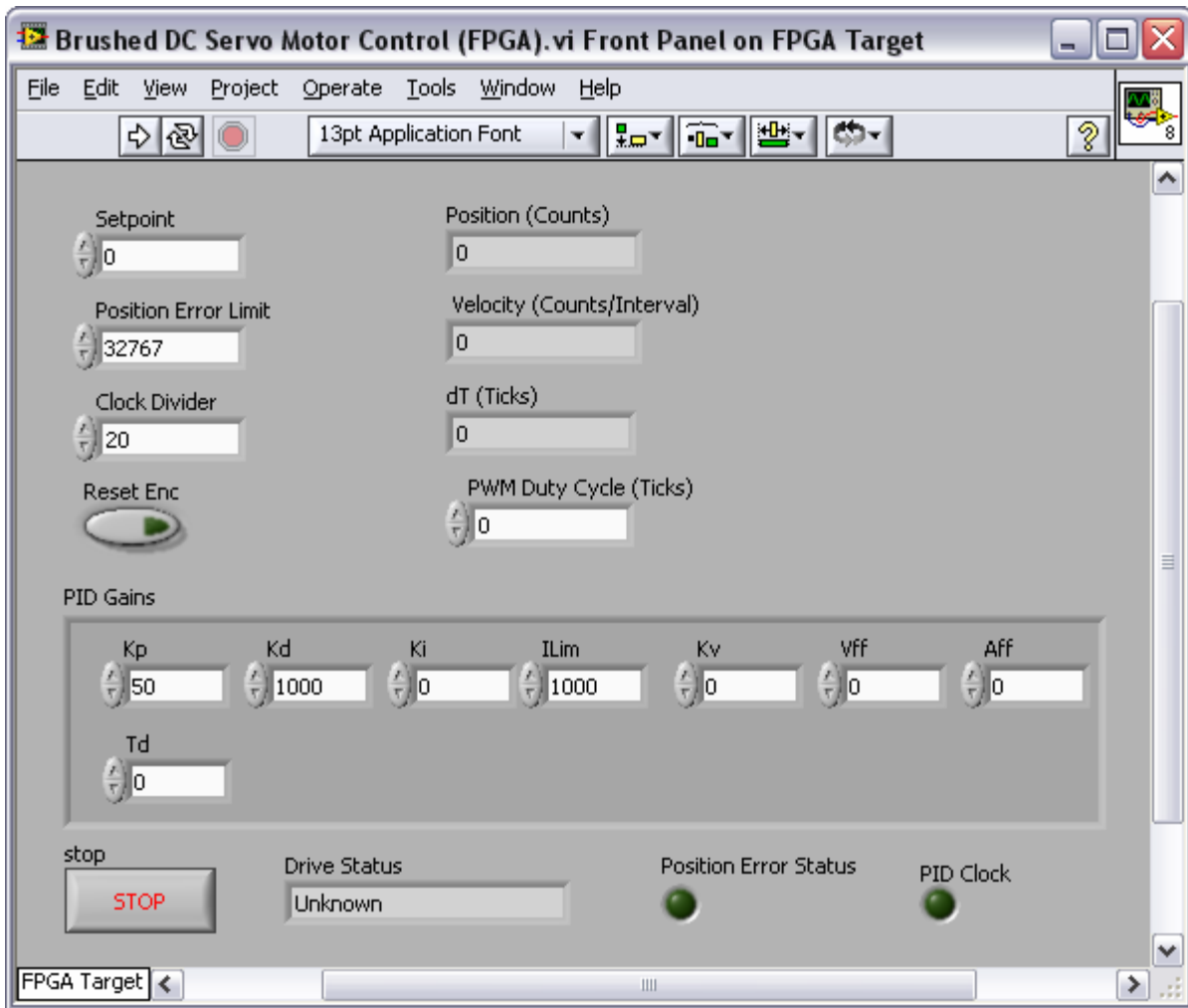
- [Learn more about using the LabVIEW shared variable networking technology](#)
- [Learn more about the LabVIEW 8.2 Datalogging and Supervisory Control \(DSC\) Module](#)
- [Learn more about creating an HMI operator interface to your CompactRIO system](#)
- [Getting Started with the LabVIEW Real-Time Module](#)
- [LabVIEW FPGA Module User Manual](#)

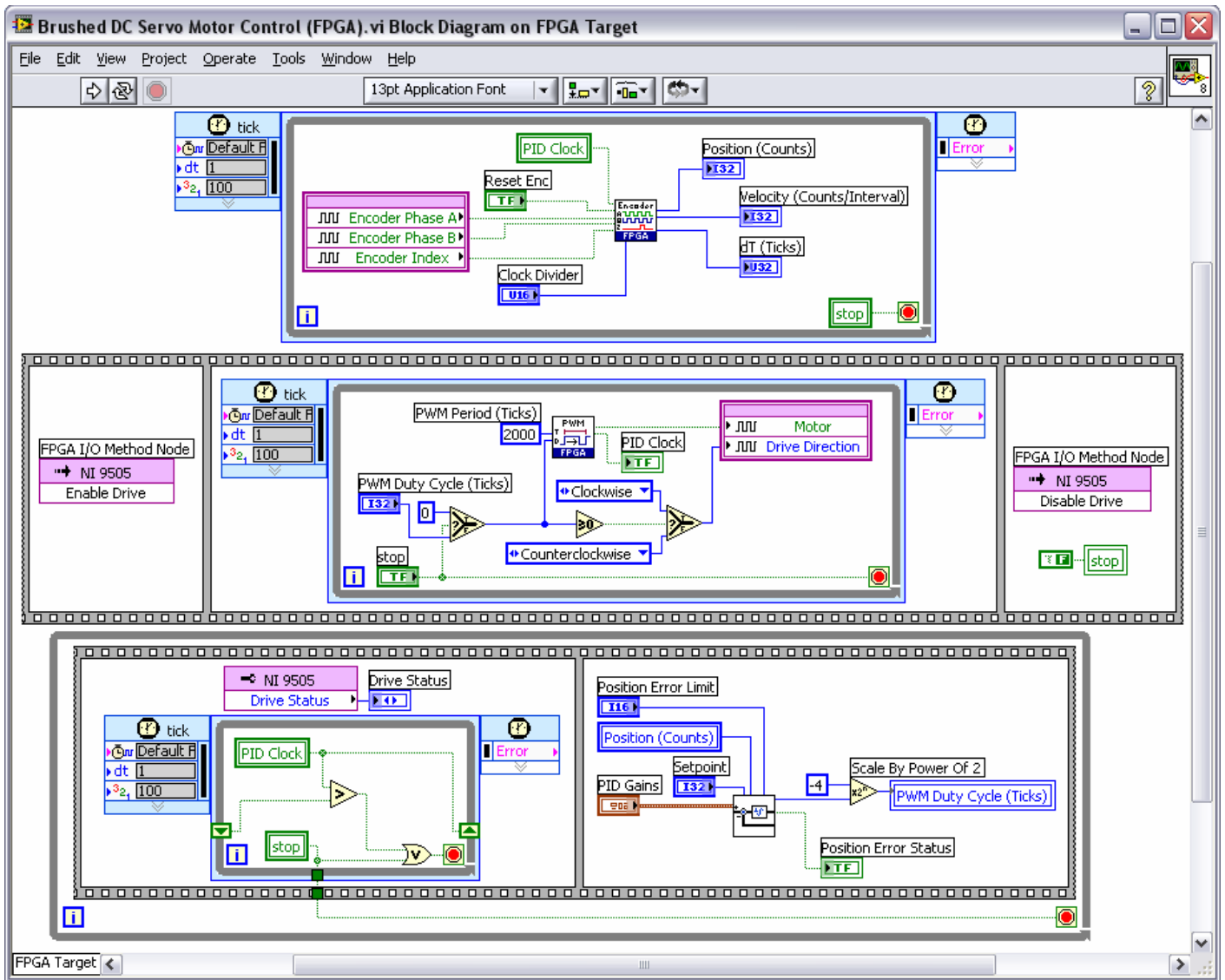
## Developing the LabVIEW FPGA Application

In this section you will:

- Create an application for an FPGA-based motor position control
- Use free encoder interface and PWM intellectual property (IP) blocks
- Read digital signals from a Quadrature encoder sensor and calculate position
- Generate a pulse-width-modulation (PWM) signal to accurately control the amount of power delivered to the motor
- Perform closed loop PID control at 20 kHz loop rates using a 32-bit PID function included with the NI SoftMotion Development Module
- Use single-cycle timed loops (SCTLs) to execute multiple parallel operations at 25 nanosecond (40 MHz) loop rates

Here is how the front panel and block diagram of your completed FPGA application will look:



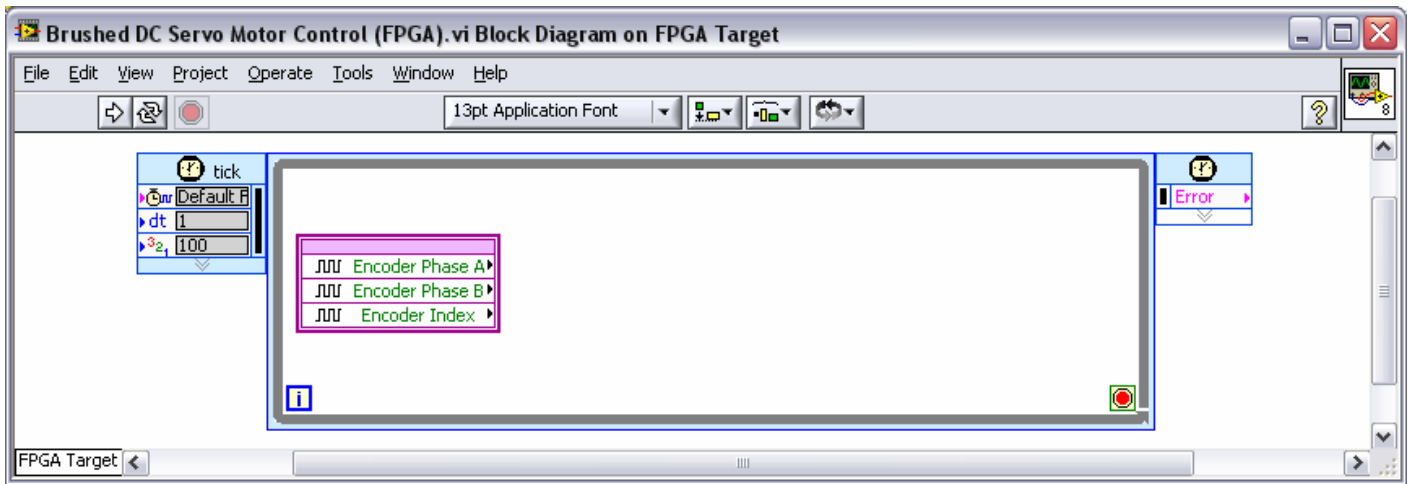


1. In the LabVIEW **Project Explorer**, right-click on the **FPGA Target** and select **New>>VI** to start a new LabVIEW FPGA application. (LabVIEW programs are called Virtual Instruments or VIs). When the VI opens, navigate to **File>>Save**. Then browse to the folder for your **CompactRIO Motor Control Basics** project and save the application as "**Brushed DC Servo Motor Control (FPGA)**".

2. Navigate to the block diagram window for your application. Right-click in the white area of the block diagram to display the **Functions** palette. Click on the thumb tack icon (📌) in the top left corner of the **Functions** palette to tack it down. Then navigate to the **Help** menu on your VI and select **Show context help**.

3. You will begin building the application by reading the digital lines from the Quadrature encoder sensor at a 25 nanosecond loop rate. First, place a **Timed Loop** from the **Functions>>Structures>>Timed Structures** palette on the block diagram.

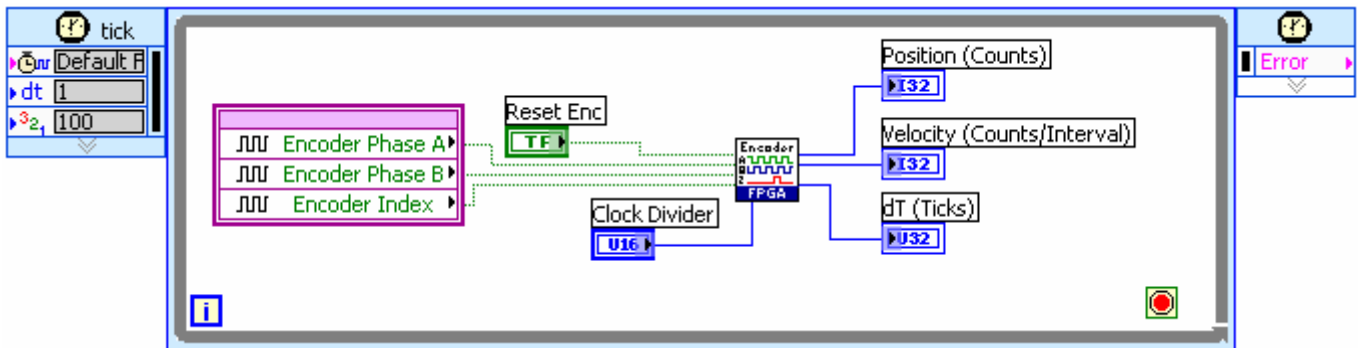
4. From the **Functions>>FPGA I/O** palette, place a **FPGA I/O Node** function inside the **Timed Loop**. Left-click on the **I/O Name** terminal and select **Digital Line Input>>NI 9505>>Encoder Phase A**. Click on the bottom of the I/O node and drag down to create terminals for **Encoder Phase B**, and **Encoder Index**.



5. On the **Functions Palette**, navigate to **Select a VI**, browse to the **H:\VirtualLab\CompactRIO Motor Control Basics Tutorial\cRIO-9104 Chassis (8-slot, 3M gates)\Exercises\Part 1\IP Cores\LabVIEW\_FPGA** folder and select **Quadrature Encoder dX Method (FPGA, Use in SCTL).vi**. Place the IP core function block inside the timed loop to the right of the I/O node. Navigate to **Help>>Show Context Help** and move your mouse over the function to see its terminal assignments and learn more about the function.

6. Wire the signals from the I/O node to the **A**, **B** and **Z** terminals of the encoder interface IP core. Then right-click on the **Reset Enc** terminal and select **Create>>Control**. Right-click on the **Clock Divider** input and select **Create>>Control**.

7. Right-click on the **Position (Counts)**, **Velocity (Counts/Interval)**, and **dT (Ticks)** outputs and select **Create>>Indicator**.

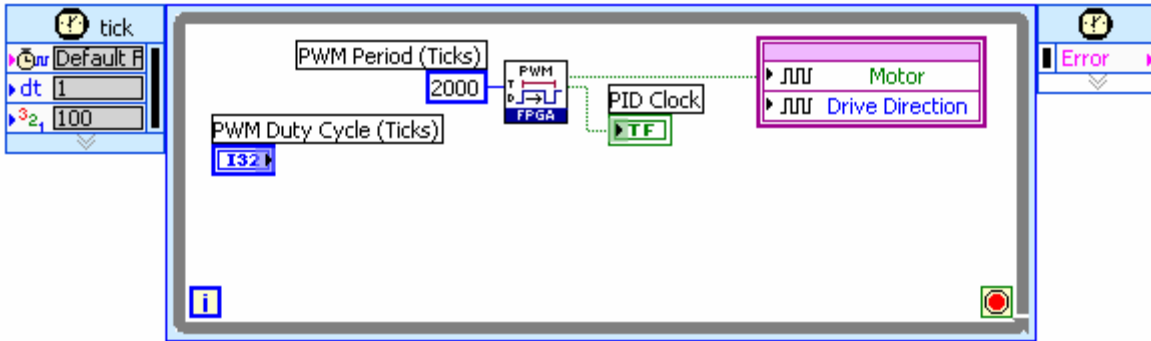


8. Next you will add a parallel 40 MHz loop to generate a pulse-width-modulation (PWM) signal that precisely controls the amount of power delivered to the motor. Place another **Timed Loop** from the **Functions>>Structures>>Timed Structures** palette on the block diagram below the encoder loop.

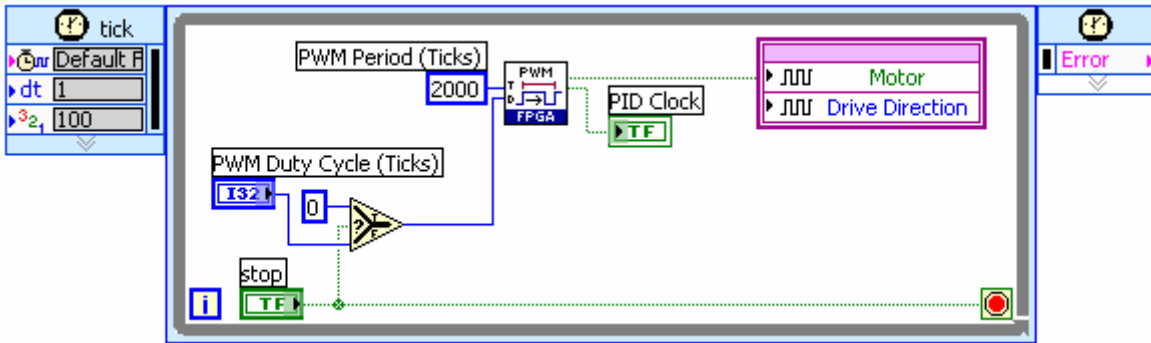
9. On the **Functions Palette**, navigate to **Select a VI**, browse to the **H:\VirtualLab\CompactRIO Motor Control Basics Tutorial\cRIO-9104 Chassis (8-slot, 3M gates)\Exercises\Part 1\IP Cores\LabVIEW\_FPGA** folder and select **Pulse Width Modulation (FPGA, Use in SCTL).vi**. Place the function block inside the timed loop.

10. Right-click on the **PWM Period (Ticks)** input and select **Create>>Constant**. Right-click on the constant and select **Visible Items>>Label**. Right-click on the **PID Clock** output and select **Create>>Indicator**.

11. From the **Functions>>FPGA I/O** palette, place a **FPGA I/O Node** function inside the **Timed Loop**. Left-click on the **I/O Name** terminal and select **Digital Line Output>>NI 9505>>Motor**. Click on the bottom of the I/O node and drag down to create another terminal for **Drive Direction**. Wire the **PWM DO** output from the PWM function to the **Motor** terminal on the I/O node. Right-click on the **PWM Duty Cycle (Ticks)** input and select **Create>>Control**. Click on the wire and delete it.

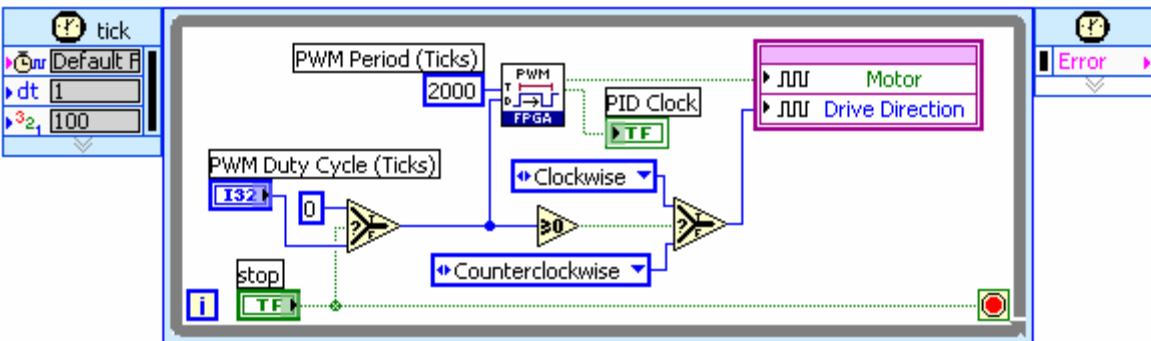


12. Navigate to the **Functions>>Comparison** palette and drop down a **Select** function. Drop it down to the right of the **PWM Duty Cycle (Ticks)** control. Right-click on the conditional terminal ( ) of the while loop and select **Create>>Control**. Move it to the left of the **Select** function and wire it to the **s** terminal. Wire the **PWM Duty Cycle (Ticks)** signal to the **f** terminal. Right-click on the **t** terminal and select **Create>>Constant**. Wire the **s? t:f** output of the **Select** function to the **PWM Duty Cycle (Ticks)** input of the **PWM** function block.



13. Navigate to the **Functions>>Comparison** palette and drop down a **Greater Or Equal To 0?** function. Wire the duty cycle signal to the **x** terminal. Drop down another **Select** function to the right, and wire the **x >= 0?** output of the **Greater Or Equal To 0?** function to the **s** terminal of the **Select** function.

14. On the **Functions Palette**, navigate to **Select a VI**, browse to the **H:\VirtualLab\CompactRIO Motor Control Basics Tutorial\cRIO-9104 Chassis (8-slot, 3M gates)\Exercises\Part 1\IP Cores\LabVIEW\_FPGA** folder and select **\_nicrio\_cRIO-9505\_DriveDirection.ctl**. Place the constant inside the timed loop and wire it to the **t** input on the **Select** function. Copy the constant, left-click to change the value to **Counterclockwise** and wire it to the **f** input on the **Select** function. Wire the output of the **Select** function to the **Drive Direction** terminal on the I/O node.



15. Next we add logic to enable the drive module before the PWM loop starts running and to disable the drive module after the PWM loop stops running. On the palette, navigate to **Functions>>Structures>>Flat Sequence Structure** and drag the structure around the outside of the PWM loop. Right-click on the left border of the sequence structure and select **Add Frame Before**. Right-click on the right border of the sequence structure and select **Add Frame After**.

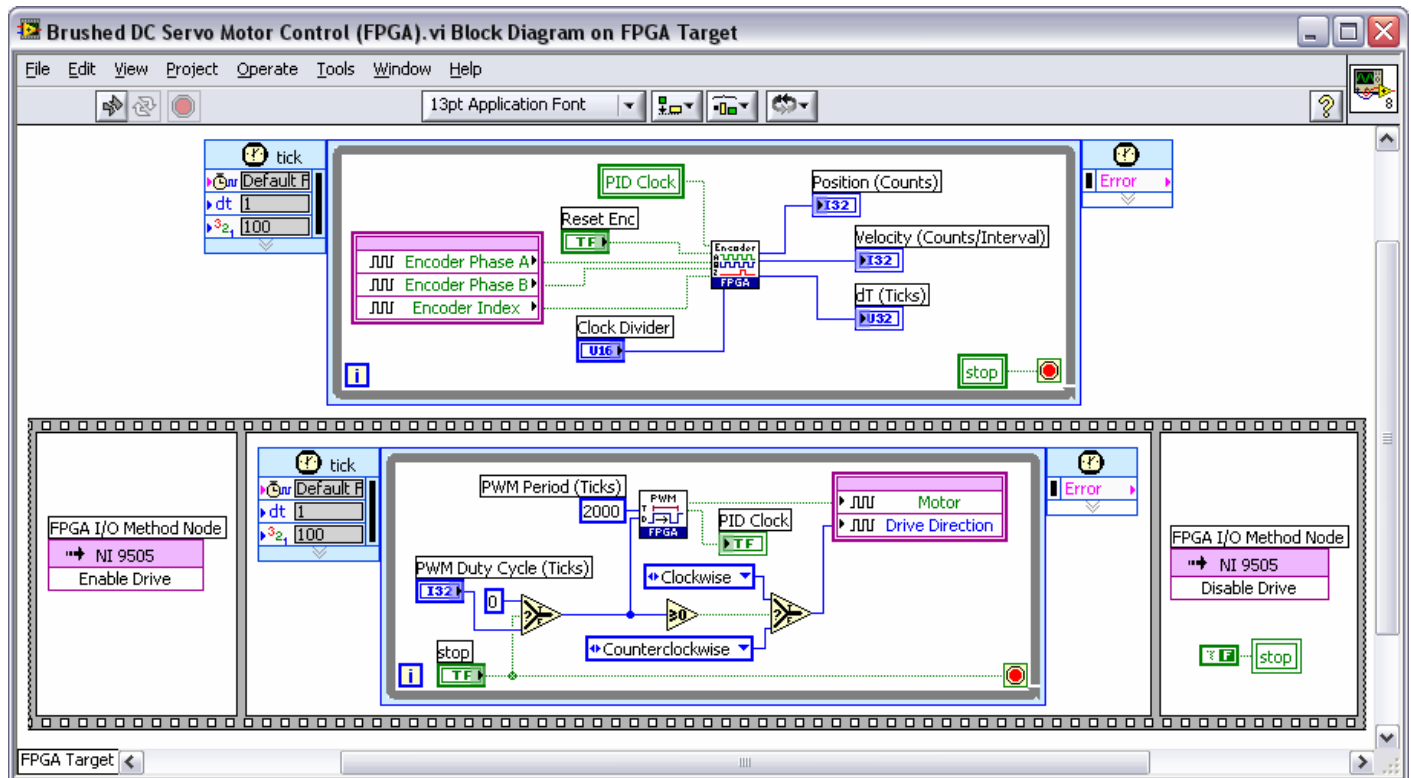
16. On the palette navigate to **Functions>>FPGA I/O>> FPGA I/O Method Node** and drop it in the left frame of the sequence structure. Right-click and navigate to **Select Item>>9505**. Right-click and navigate to **Select Method>>Enable Drive**. Copy the


method node and drag the copy into the right frame of the sequence structure. Right-click and navigate to **Select Method>>Disable Drive**.

17. Right-click on the **stop** control and select **Create>>Local Variable**. Drop the local variable into the right frame of the sequence structure. Right-click on the **stop** local variable and select **Create>>Constant**.

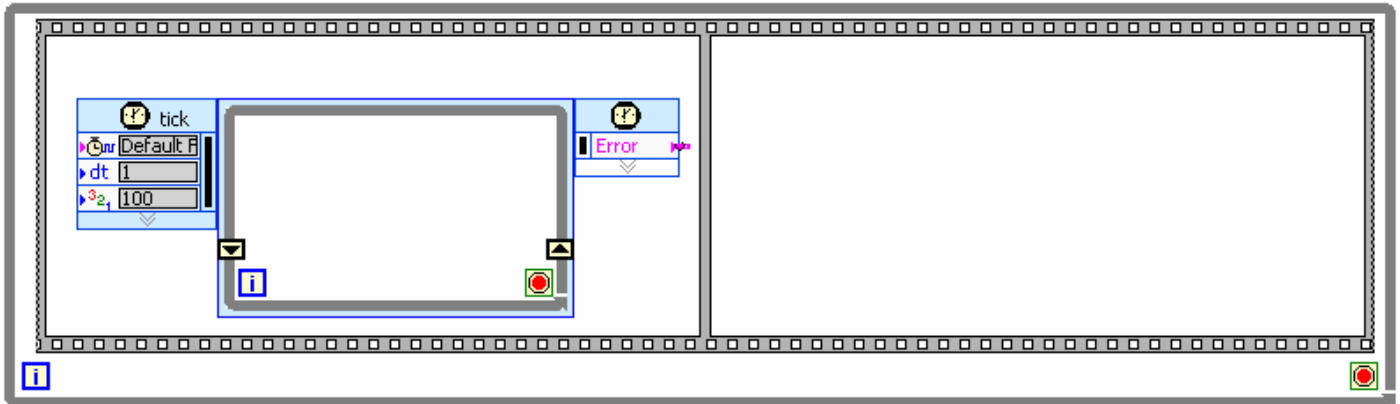
18. Right-click on the **stop** control again and select **Create>>Local Variable**. Drop the local variable into the encoder loop. Right-click on the local variable and select **Change to Read**. Then wire the local variable to the conditional terminal on the encoder loop.

19. Right-click on the **PID Clock** indicator and select **Create>>Local Variable**. Drop the local variable into the encoder loop. Right-click on the local variable and select **Change to Read**. Then wire the local variable to the **Clock** input on the encoder IP core. Your application should look similar to that shown below.




20. Navigate to the front panel and right-click on the **stop** control. Navigate to **Mechanical Action** and select **Switch When Pressed**. The broken run arrow on your application should change to a working run arrow (  ). Navigate to **File>>Save**.

21. Next we will add a PID control loop and triggering logic to synchronize the loop execution with the rising edge of the **PID Clock** signal. First, place a **While Loop** from the **Functions>>Structures** palette on the block diagram below the PWM loop. Inside the while loop, place a **Flat Sequence Structure** from the **Functions>>Structures** palette. Right-click on the right border of the sequence structure and select **Add Frame After**. Place a **Timed Loop** from the **Functions>>Structures>Timed Structures** palette inside the left frame of the sequence structure. Left-click on the lower-left border of the timed loop and select **Add Shift Register**. Your application should look similar to that shown below.

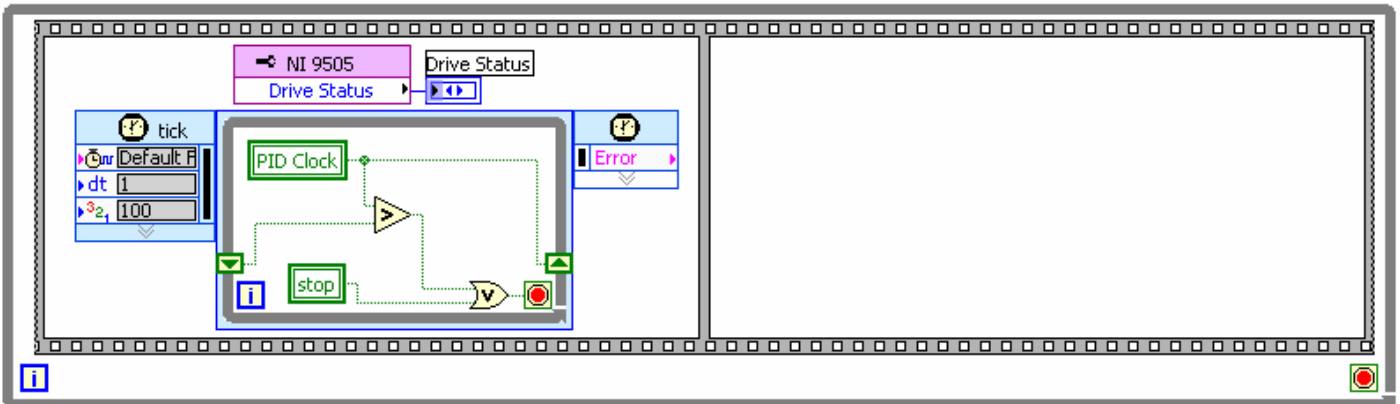


22. Right-click on the **PID Clock** indicator and select **Create>>Local Variable**. Drop the local variable into the timed loop. Right-click on the local variable and select **Change to Read**. Then wire the local variable to the shift register terminal on the right on the timed loop. Every 25 nanoseconds, this shift register will pass the value of the **PID Clock** signal from one iteration of the loop to the next.

23. Navigate to the **Functions>>Comparison** palette and drop down a **Greater?** function. Drop it down to the right of the **PID Clock** local variable. Wire the **PID Clock** signal to the **x** terminal, and wire the left shift register terminal to the **y** terminal. Navigate to the **Functions>>Boolean** palette and drop down an **Or** function. Wire the **x > y?** output of the **Greater?** Function to the **x** terminal of the **Or** function. Wire the **x .or. y?** output of the **Or** function to the conditional terminal (  ) of the timed loop.

24. Right-click on the **stop** control again and select **Create>>Local Variable**. Drop the local variable into the timed loop. Right-click on the local variable and select **Change to Read**. Then wire the local variable to the **y** terminal of the **Or** function.

25. Navigate to **Functions>>FPGA I/O>> FPGA I/O Property Node** and drop it in the left frame of the sequence structure above the timed loop. Right-click and navigate to **Select Item>>9505**. Left-click on the terminal and select the **Drive Status** property from the list. Then right-click on the terminal and select **Create>>Indicator**. Your application should look similar to that shown below.



*By placing the **PID Clock** local variable into the timed loop and comparing the current value to the previous value using a shift register, the application is able to detect the rising edge of the signal when it transitions from a 0 (False) state to a 1 (True) state. The timed loop will keep executing at a 40 MHz rate until a rising edge is detected or the stop condition occurs.*

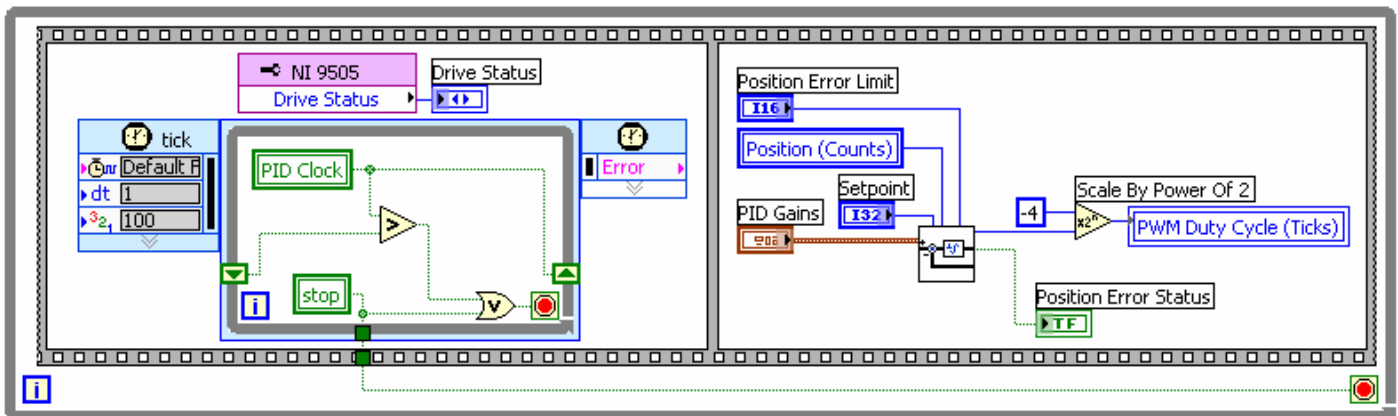
26. Next you will add a 32-bit PID control algorithm from the NI SoftMotion Development Module for closed loop position control. On the **Functions Palette**, navigate to **Select a VI**, browse to the **H:\VirtualLab\CompactRIO Motor Control Basics Tutorial\cRIO-9104 Chassis (8-slot, 3M gates)\Exercises\Part 1\IP Cores\SoftMotion\ControlLoop** folder, double-click on the **ControlLoop PID(FixedPoint).ilb** file and select **nimcDMControlLoop\_PID(FixedPoint).vi**. Place the function block inside right frame of the sequence structure.

27. Right-click on the **PID Gains** terminal of the SoftMotion PID function and select **Create>>Control**. Do the same for the **Setpoint** and the **Position Error Limit** terminals. Right-click on the **Position Error Status** terminal and select **Create>>Indicator**.

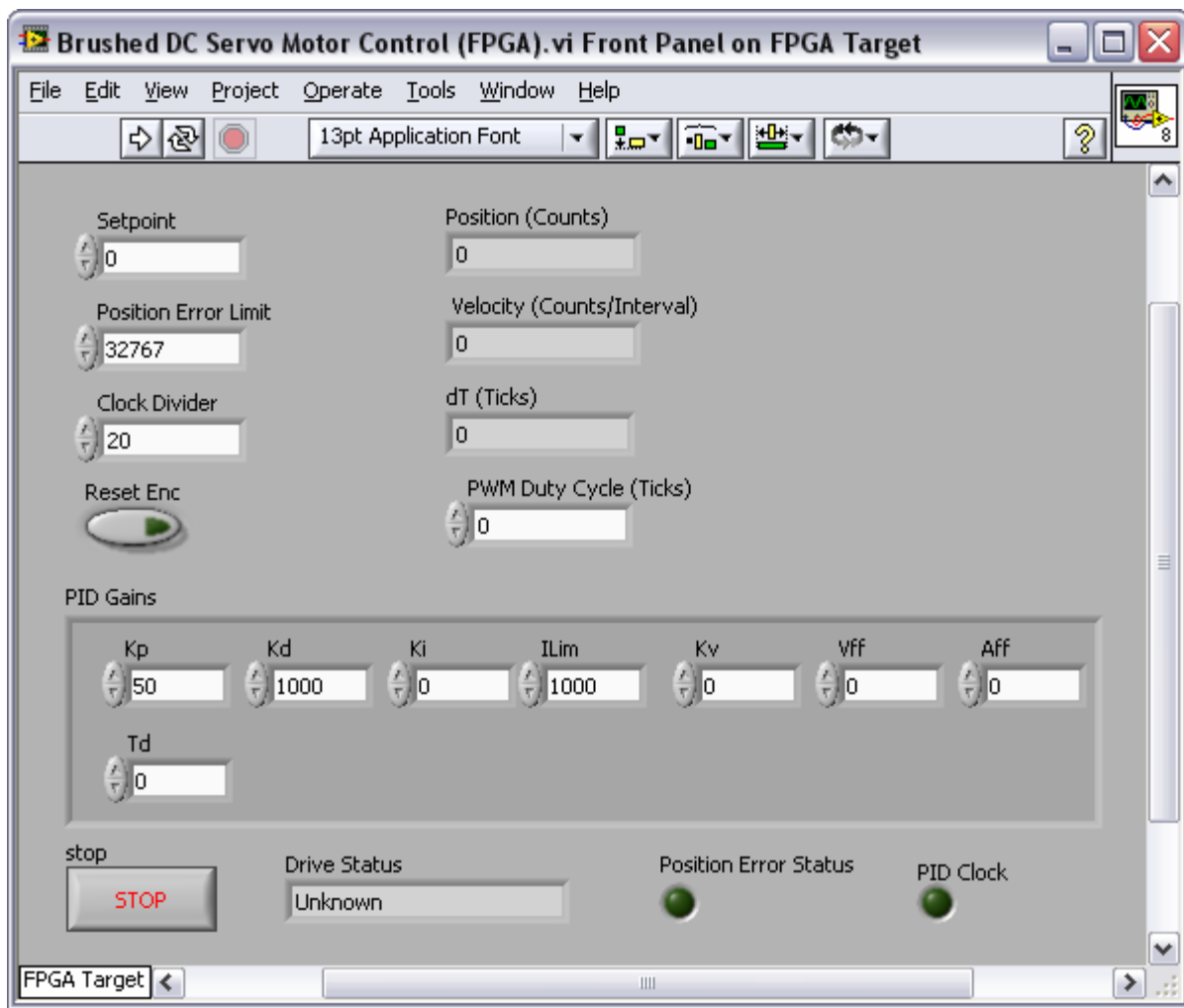
28. Right-click on the **Position (Counts)** indicator from the encoder loop and select **Create>>Local Variable**. Drop the local variable to the left of the PID function. Right-click on the local variable and select **Change to Read**. Then wire the local variable to the **Position Feedback** terminal of the PID function.

29. The output range of the SoftMotion PID function is the full scale range of a 16-bit integer (-32,768 to +32,767). In this case, we want to scale the Output to a range that is compatible with our PWM function (-2,000 to +2,000). To do this, we'll use a **Scale By Power Of 2** function to divide the Output value by 16, resulting in a signal with a range from (-2,048 to +2,047). To do this, navigate to the **Functions>>Numeric** palette and drop down a **Scale By Power Of 2** to the right of the PID block. Wire the Output signal to the **x** terminal of the **Scale By Power Of 2** function. Then right-click on the **n** terminal and select **Create>>Constant** and set the value to **-4**. Then right-click on the constant, navigate to **Representation** and select **I8**. By changing the representation of the scaling constant to an 8-bit signed integer (rather than the default 32-bit signed integer), we reduce the FPGA gate consumption for the divide operation significantly.

30. Finally, right-click on the **PWM Duty Cycle (Ticks)** control from the PWM loop and select **Create>>Local Variable**. Drop the local variable to the right, and wire the **x\*2^n** output of the **Scale By Power Of 2** function. Then wire the stop signal from the left frame of the sequence structure to the conditional terminal of the while loop. Your completed PID loop should look similar to that shown below.



31. Navigate to the front panel and arrange it similar to that shown below. Set the value for **Position Error Limit** to **32767**. Set the value for **Clock Divider** to **20**. Then navigate to **Edit>>Make Current Values Default**. and save the application. Finally, navigate to **File>>Save All** to save all open applications and the project.



**See Also:**


[Learn more about the NI SoftMotion Development Module for LabVIEW](#)

## Running the Application

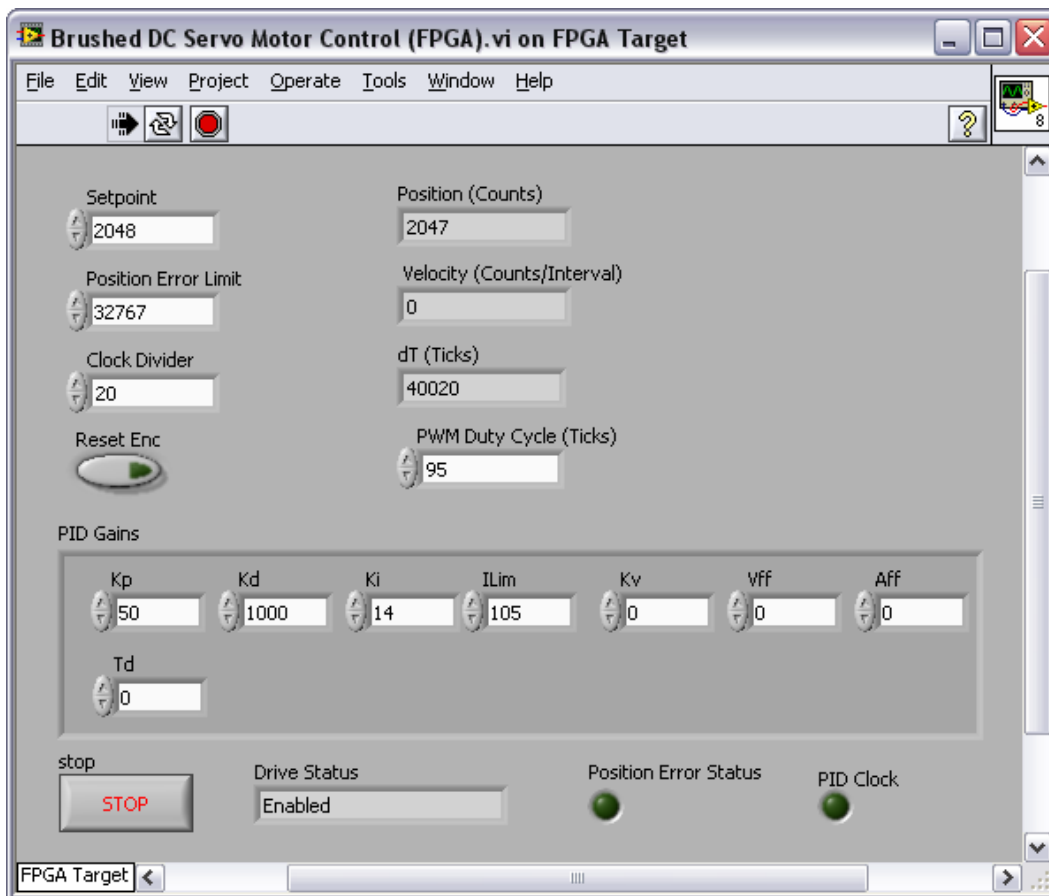
In this section you will:


- Open a precompiled version of the LabVIEW FPGA application
- Run the LabVIEW FPGA application and verify its operation
- Run the LabVIEW Real-Time application and verify its operation

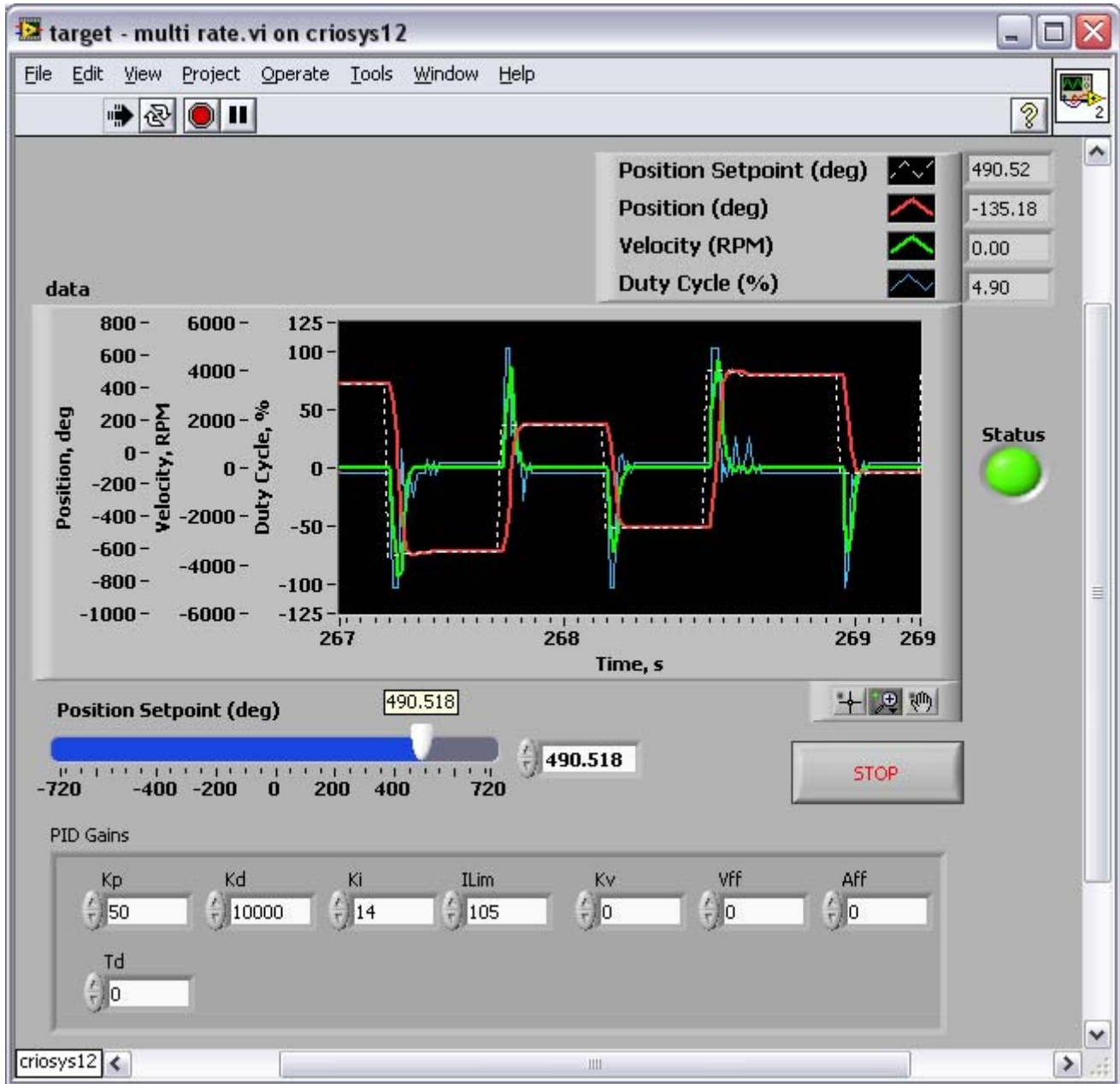
1. Close any open LabVIEW applications and close your project. Then go to **File>>Open Project** and navigate to the **H:\VirtuaLab\CompactRIO Motor Control Basics Tutorial\cRIO-9104 Chassis (8-slot, 3M gates)\Exercises\Part 2** directory. Open the **CompactRIO Motor Control Basics.lvproj** project. Expand the **FPGA Target** in the project tree and double-click to open the **Brushed DC Servo Motor Control (FPGA).vi** file.

2. Click the **Run** button (  ) to run the precompiled FPGA application. If prompted to configure the compile settings, click the **OK** button to accept the default settings. Change the **Setpoint** to **2048** and observe the response via the **Position (Counts)** indicator. For the [MicroMo Electronics 3242 DC micromotor](#) with IE2-512 encoder, one full rotation is equivalent to 2048 counts. Try testing different position setpoints and observe the response.

3. Try experimenting with the **Ki** (integral gain) and **Ilim** (integration limit) values in the **PID Gains** cluster to minimize the difference between the **Setpoint** and the **Position (Counts)**. Then change the **Setpoint** to different values to test your PID tuning. Click the **stop** button when you are finished testing the application. Close the FPGA application.

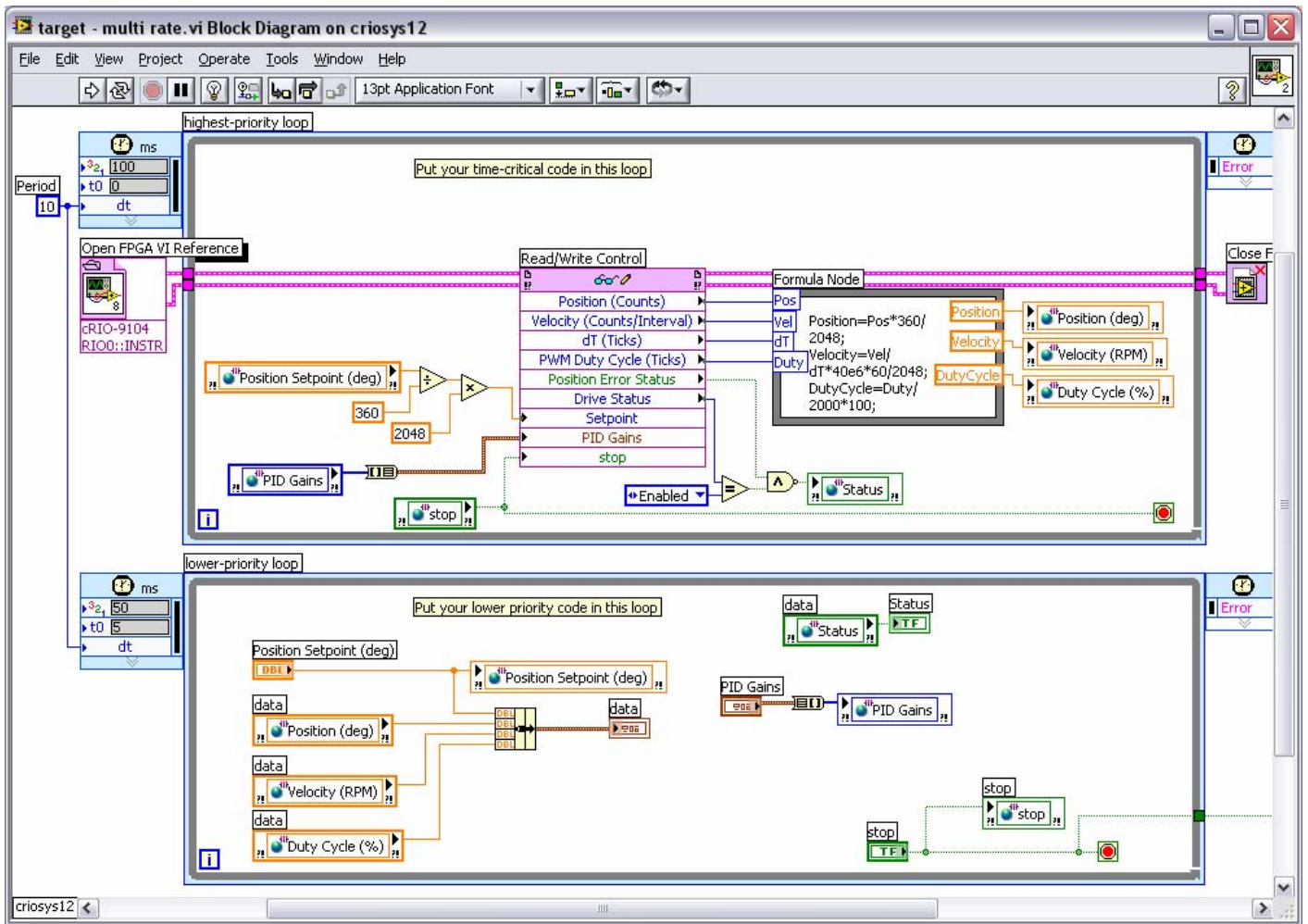


4. In the project tree, double-click to open the **target - multi rate.vi** for the real-time processor. Click the **Run** button (  ) to run the prewritten LabVIEW Real-Time application. While the application is running, try changing the **Position Setpoint (deg)** control and observe the response of the motor.



*360 degrees is equal to one full revolution of the motor.*

5. Open the block diagram of the application and familiarize yourself with the code.

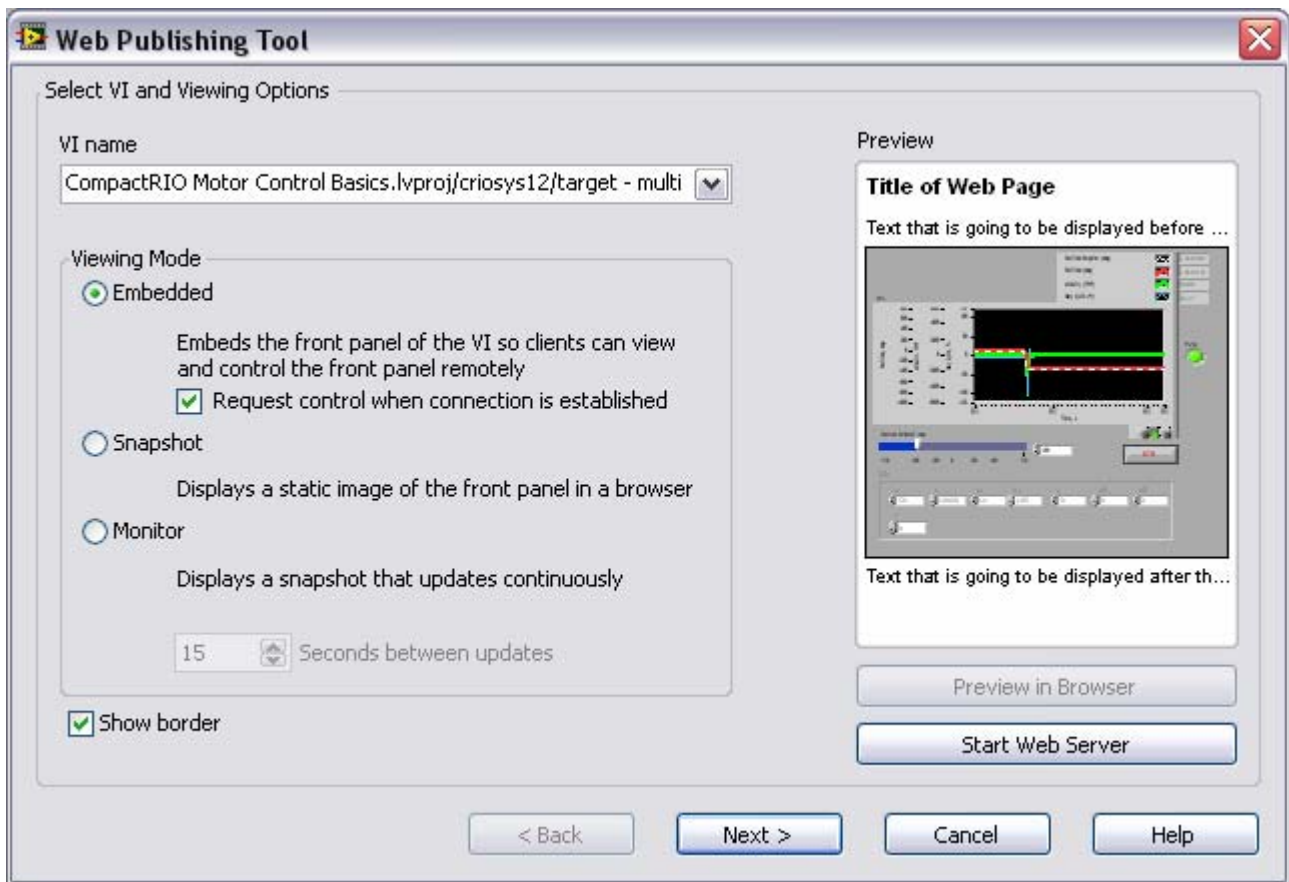


In this multi-threaded application for the CompactRIO real-time processor, two timed loops are used to separate the lower-priority user-interface tasks (lower loop) from the higher priority task of interfacing with the FPGA application (upper loop). Notice that all of the user-interface controls and indicators are located in the lower loop, which is set to a priority of 50 with an offset of 5 milliseconds. The upper loop is set to a priority of 100 with 0 offset. In the upper loop, we use a **Read/Write Control** from the **FPGA Interface** palette to communicate with the FPGA and a **Formula Node** to convert the integer data from the FPGA to scaled floating-point engineering units.

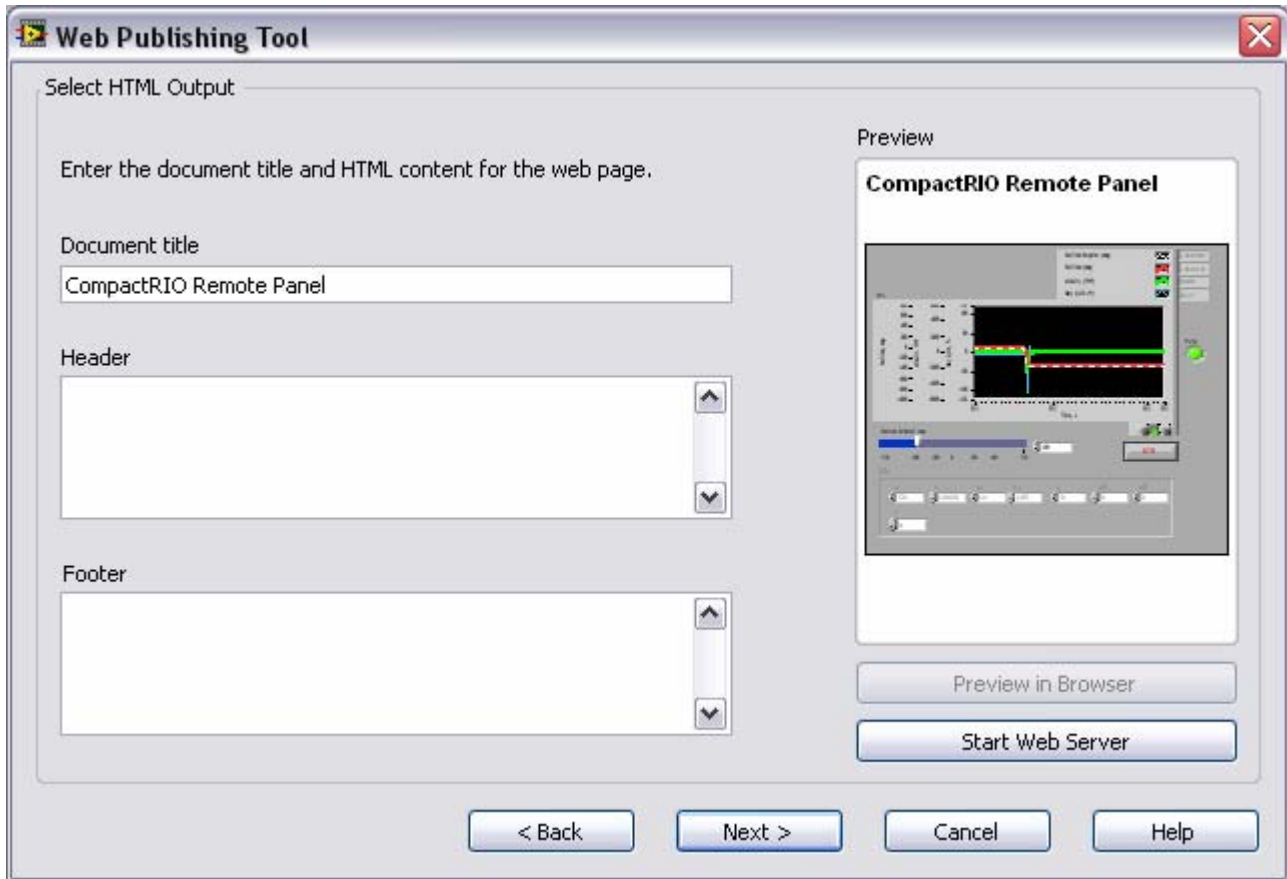
# Controlling the System through a Web Browser

In this section you will:

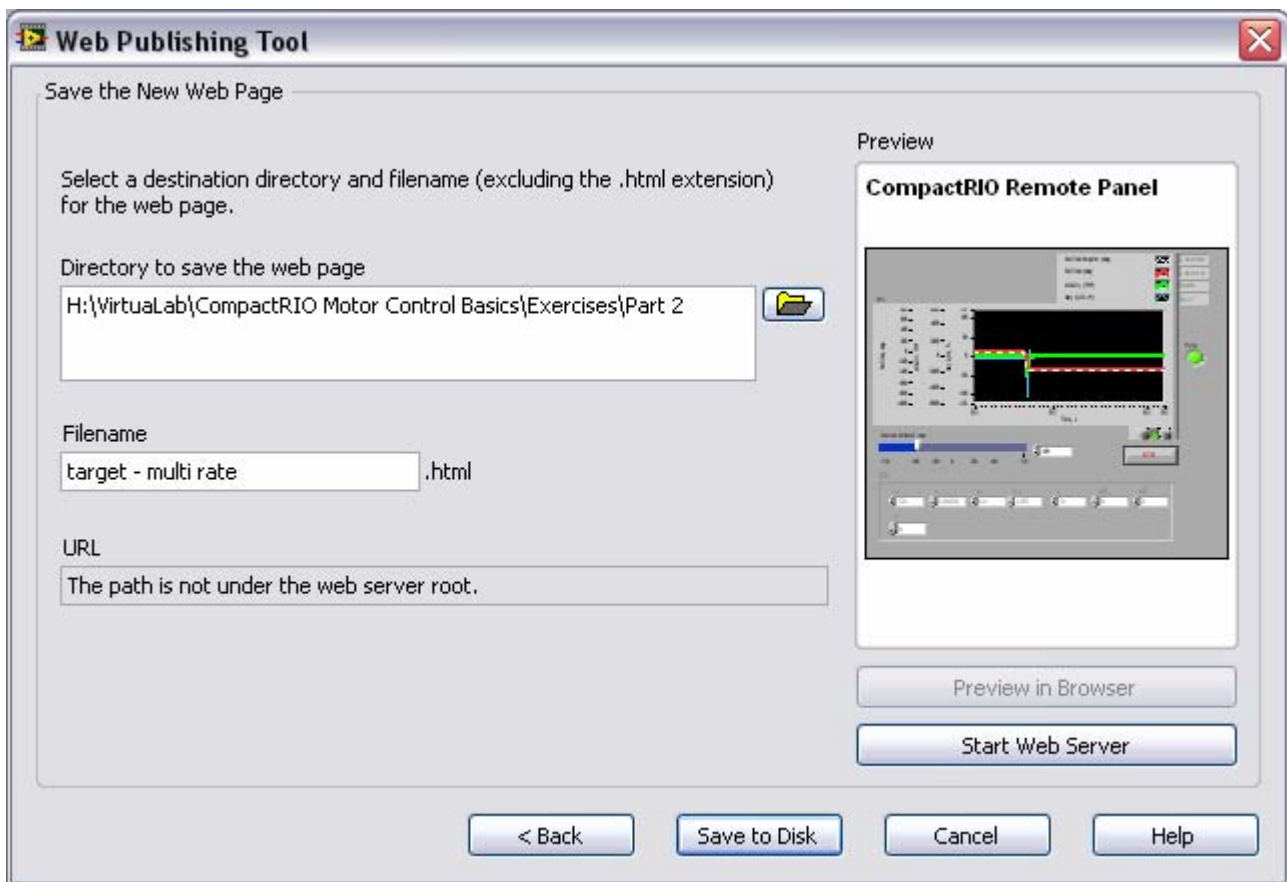
- Use the LabVIEW Web Publishing Tool to create a .HTM file for your application
  - FTP the .HTM file to the CompactRIO web server
  - Control the CompactRIO system from a web browser
1. While your CompactRIO application is running, navigate to **Tools>>Web Publishing Tool** and select **target - multi rate.vi** from the **VI name** selection box. Click the box next to **Request control when connection is established**, and then click the **Next** button.



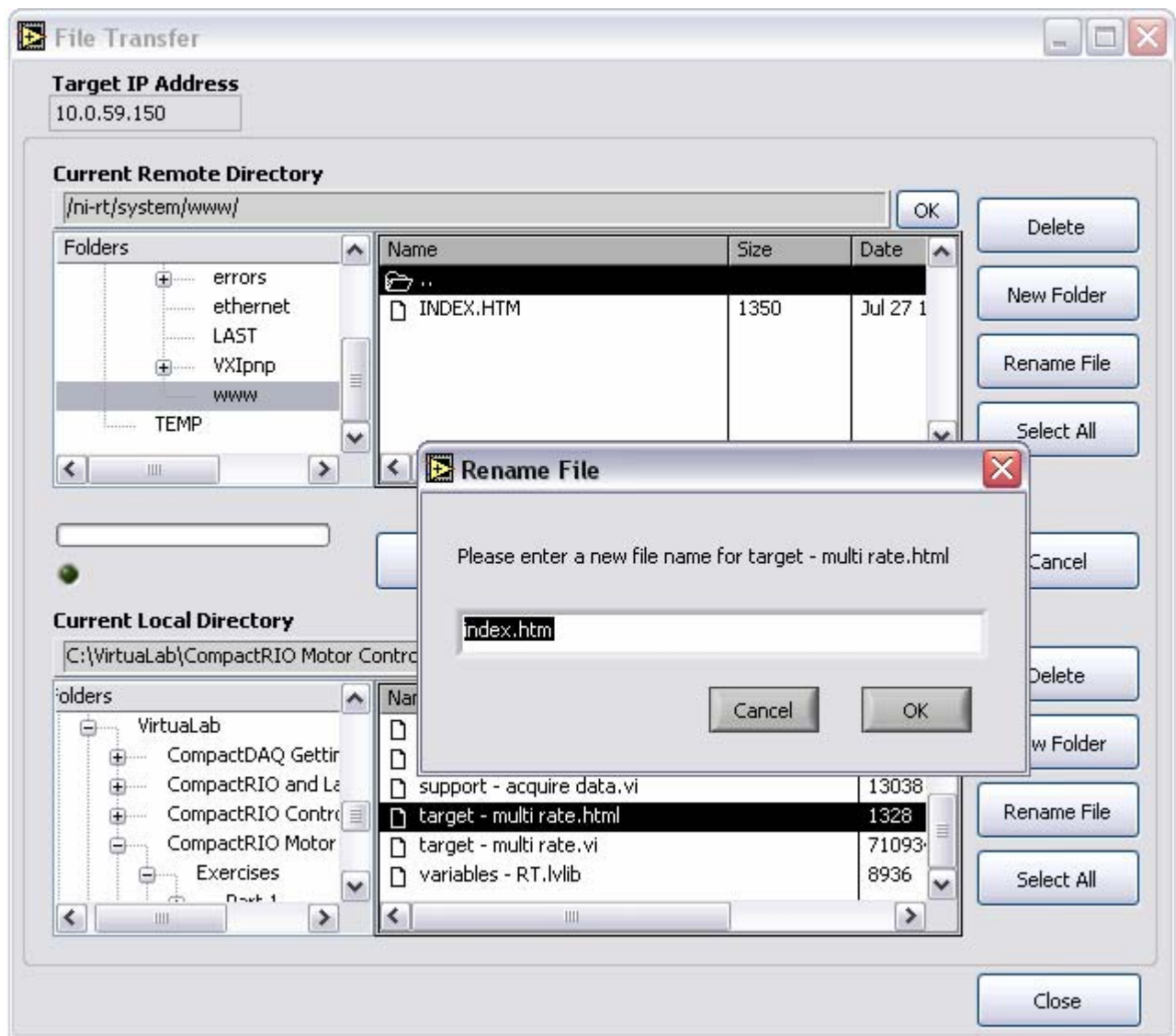
2. Change the **Document Title** to **CompactRIO Remote Panel**, delete the **Header** and **Footer** text, and click **Next**.



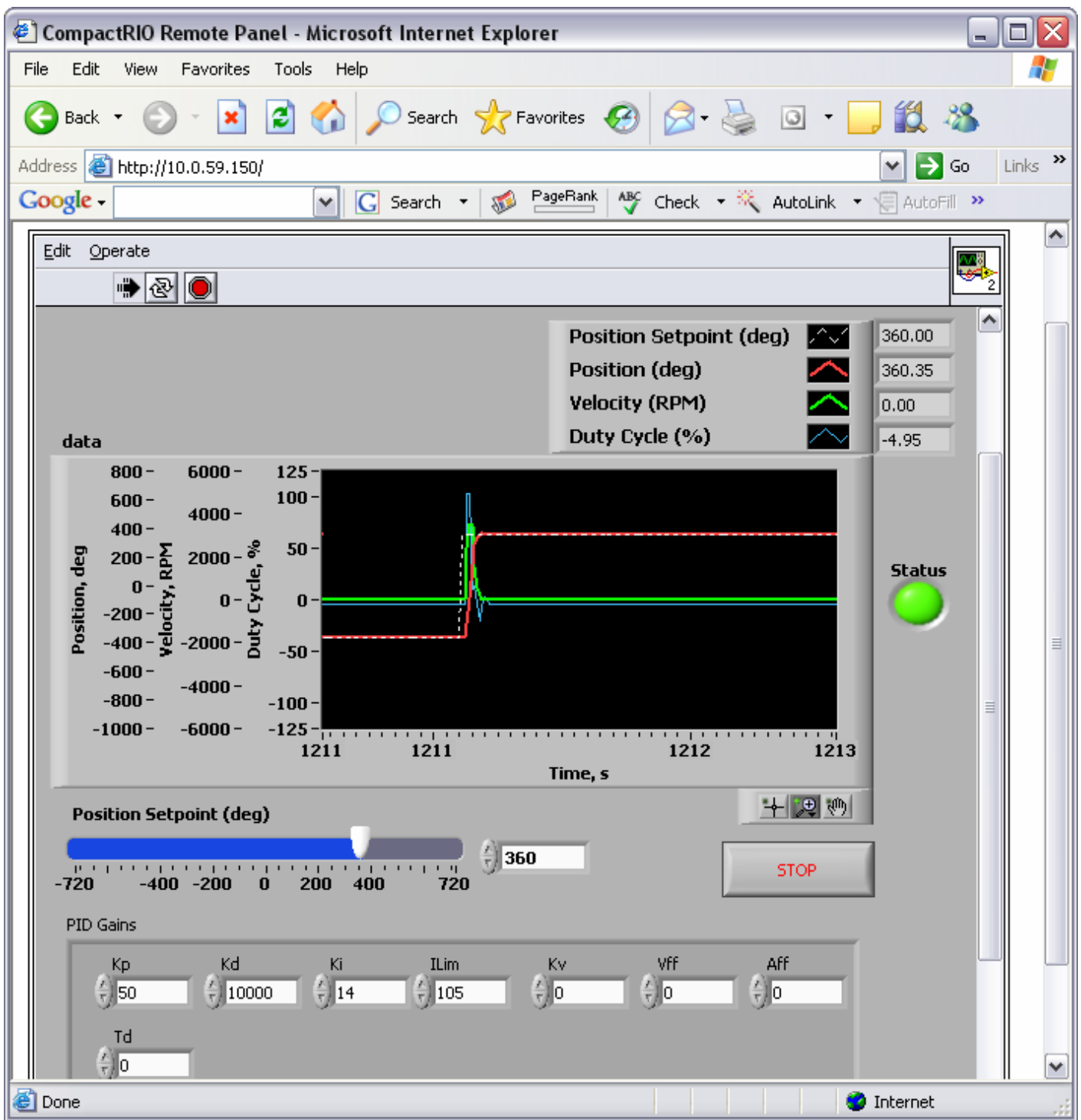
3. Change the **Directory** to **H:\VirtuaLab\CompactRIO Motor Control Basics Tutorial\cRIO-9104 Chassis (8-slot, 3M gates)\Exercises\Part 2** and click **Save to Disk**. Click **OK** when prompted to acknowledge that you are not saving to the default web server directory.



4. Launch the **Measurement & Automation Explorer** by clicking the link on your desktop or navigating to **Start>>Program Files>>National Instruments>> Measurement & Automation**. Expand the **Remote Systems** tab, right-click on your CompactRIO system and select **File Transfer**. When prompted, click the check box for **Anonymous Login** and then click **OK**.
  
5. In the upper window, navigate to the **/ni-rt/system/www/** folder on your CompactRIO system. In the lower window, navigate to **H:\VirtuaLab\CompactRIO Motor Control Basics Tutorial\cRIO-9104 Chassis (8-slot, 3M gates)\Exercises\Part 2**. Select the file named **target - multi rate.html** and click **Rename File**. Change the name to **index.htm**. (Make sure you change the file to an **.htm** ending rather than **.html**.)



- Highlight the **index.htm** file and click the **To Remote** button. If prompted, click **OK** to overwrite the existing file. Then click **Close** to close the **File Transfer** window.
- In **Measurement & Automation Explorer**, note the IP Address of your CompactRIO system. Launch a new web browser and change the address to **http://xxx.xxx.xxx.xxx**, where **xxx.xxx.xxx.xxx** is the IP address of your system. Once you are connected, experiment with controlling the CompactRIO system from your web browser.



- While leaving the application running, navigate back to LabVIEW, right-click on the CompactRIO system in the **Project Explorer** window and select **Disconnect**. Switch back to your web browser and note that you can still control the embedded CompactRIO system from your web browser even after you have disconnected from the system in the LabVIEW development environment.

## Summary

In this introduction to basic motor control with CompactRIO and the NI 9505 motor drive module, you learned the basics of programming LabVIEW FPGA for motor control using the NI SoftMotion PID function and free downloadable IP cores for PWM and Quadrature encoder interfacing. You also learned how to set up a LabVIEW Project and use the Multiple Variable Editor to configure shared variables for the project. You ran a precompiled version of the FPGA application you created and also explored a multi-threaded real-time processor application designed for hard real-time execution. Additionally you learned how to create a remote panel web browser user interface for your embedded CompactRIO application.

