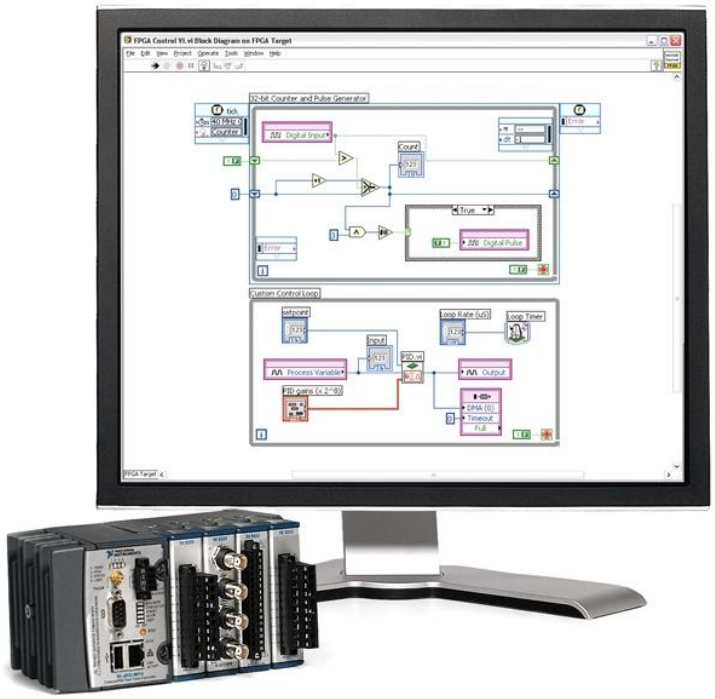
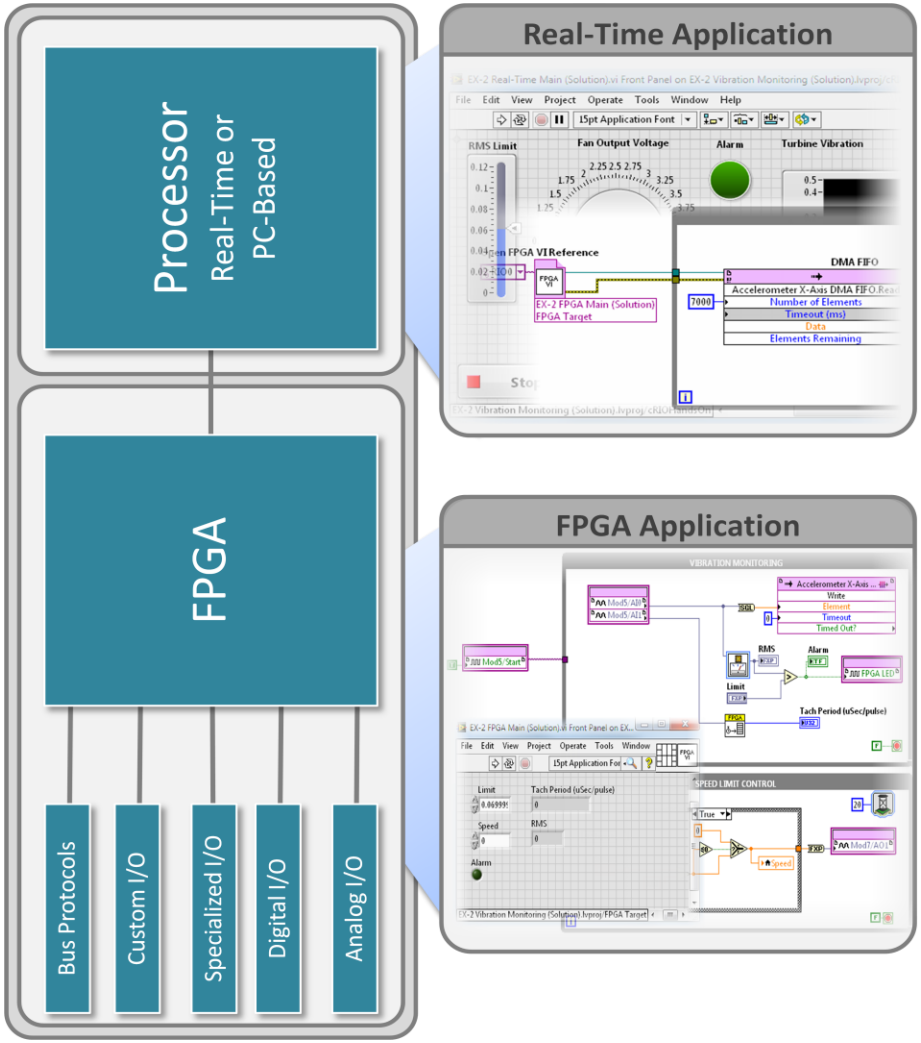
Hands-on Embedded Systems

Developing Monitoring and Control Systems with LabVIEW and CompactRIO



Contents

THE LABVIEW RIO ARCHITECTURE



- Real-time OS
- Application software
- Networking and peripheral I/O drives
- DMA, interrupt, and bus control drivers

- Application IP
- Control IP
- DSP IP
- Specialized I/O drivers and interface
- DMA controller

The CompactRIO System

In this Hands-On Seminar, you will be working with one of the following systems:

The **NI cRIO-9022** embedded real-time controller is part of the high-performance CompactRIO programmable automation controller platform with an integrated 533 Mhz processor.
The **NI cRIO-9113** four-slot, reconfigurable embedded chassis features a user-programmable Xilinx Virtex-5 FPGA and has four slots for NI C Series I/O modules.

RT: 533 MHz processor
2 GB nonvolatile storage
256 MB DDR2 memory

FPGA: Xilinx Virtex-5 LX50
Flip-flops: 28.800
6-input LUTs: 28.800
Multipliers: 48
Block RAM: 1728 kbits
DMA channels: 3

The **NI cRIO-9076** integrated system combines a real-time processor and a reconfigurable field-programmable gate array (FPGA) within the same chassis for embedded machine control and monitoring applications. It integrates a 400 MHz industrial real-time processor with an LX45 FPGA and has four slots for NI C Series I/O modules.

RT: 400 MHz processor
512 GB nonvolatile storage
256 MB DDR2 memory

FPGA: Xilinx Spartan-6 LX45
Flip-flops: 54.576
6-input LUTs: 27.288
Multipliers: 58
Block RAM: 2088 kbits
DMA channels: 5

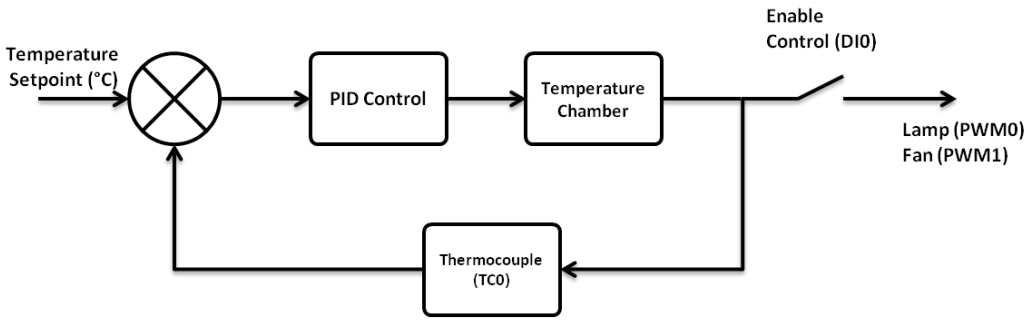
EXERCISE 1: TEMPERATURE CONTROL

Goals

- Use the ***LabVIEW*** system design software and the ***CompactRIO*** system configured in ***Scan Mode*** to interface with sensors and actuators located in the ***temperature chamber*** of the ***CompactRIO Hands-On box***.
- Implement a ***closed-loop PID algorithm*** with conditional output to control the temperature within the ***temperature chamber***.

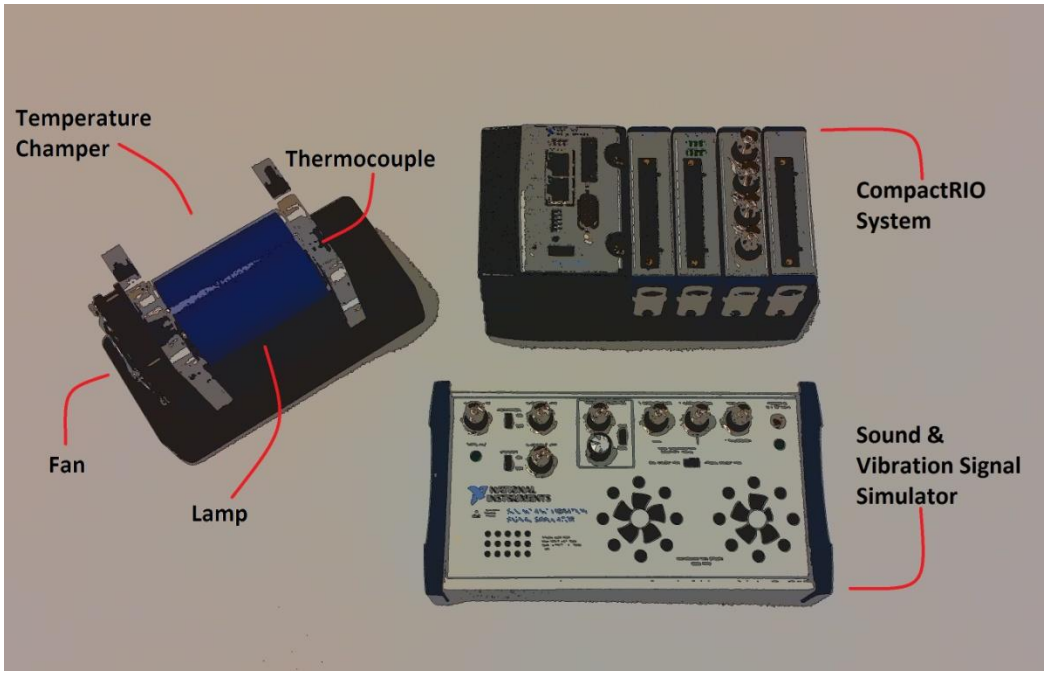
Part A—APPLICATION DESCRIPTION

Temperature Control With Conditional Output of a Temperature Chamber



In this exercise, use ***LabVIEW*** and the ***CompactRIO system*** configured in ***Scan Mode*** to implement a ***closed-loop PID control algorithm*** with conditional output to control the temperature within the ***temperature chamber***.

CompactRIO Hands-On Kit

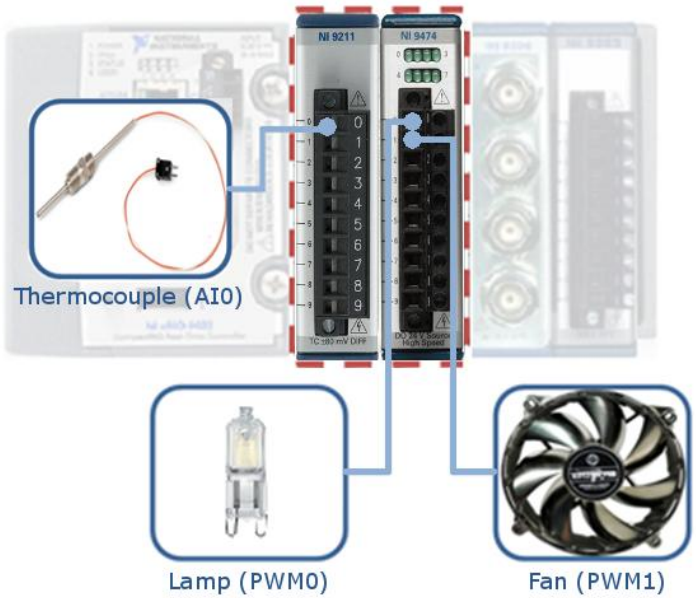


The ***CompactRIO System*** consists of a RT controller and a FPGA on the backplane. In addition, we are using different IO modules.

In this exercise, we will be using the ***temperature chamber.***

Locate the ***lamp***, ***fan***, and ***thermocouple*** that you will use during the exercise.

CompactRIO System



NI Scan Mode

This feature gives you easy access to signals wired to measurement modules connected to the ***field-programmable gate array (FPGA)*** included in the ***CompactRIO system***.

In this exercise, use the measurement modules inserted in ***slots 1*** *and* ***2*** of the ***CompactRIO system***.

Inputs

Slot 1: **NI 9211**—*Thermocouple Input Module*
Analog Input 0 (AI0)→Thermocouple

Outputs

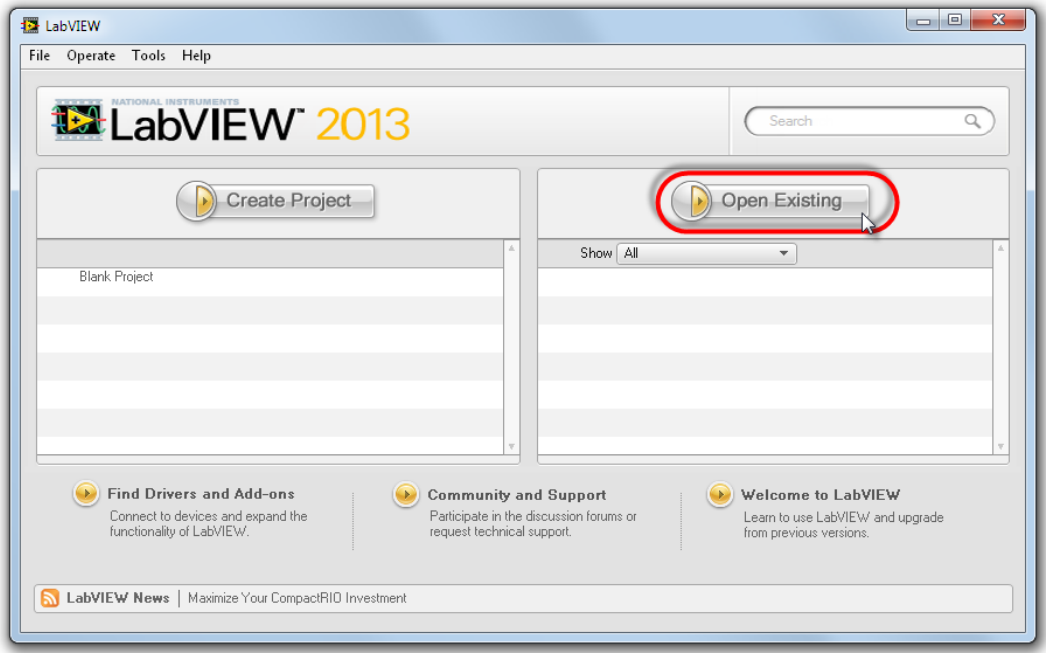
Slot 2: NI 9474—*Sourcing Digital Output Module (Configured as PWM)*
Digital Output 0 (PWM0)→Lamp
Digital Output 1 (PWM1)→Fan

Part B—CODE IMPLEMENTATION

1. Open Temperature Control.lvproj.

Launch LabVIEW and open the existing LabVIEW project Temperature Control.lvproj.

C:\CompactRIO HO\Exercise 1\Temperature Control.lvproj



DETAILED INSTRUCTIONS

In this exercise, use an existing ***LabVIEW project*** as the starting point of the application.

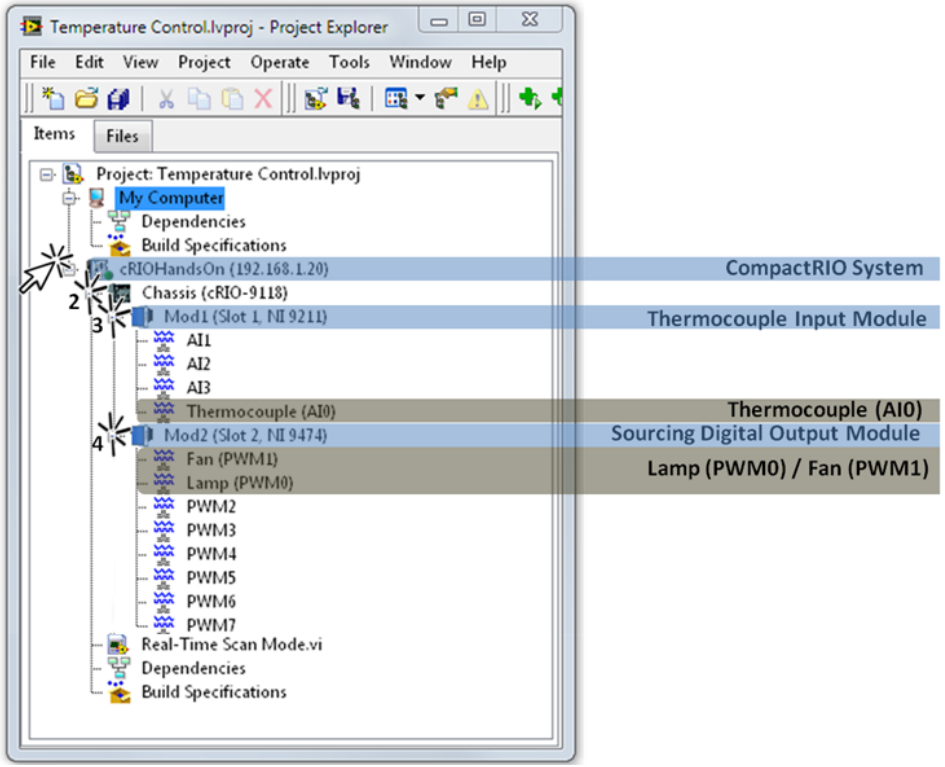
- ***Launch LabVIEW***
  Launch LabVIEW by navigating to ***Start » All Programs » National Instruments » LabVIEW 2013 » LabVIEW***.

- ***Open an Existing LabVIEW Project***
  Click on the ***Open Existing*** button on the right side of the ***LabVIEW*** main screen and open the ***LabVIEW Project*** file named ***Temperature Control.lvproj*** located at:

  C:\CompactRIO HO\Exercise 1\Temperature Control.lvproj

2. Explore the project and reveal its components.

Expand each hierarchy on the project to reveal the measurement modules and I/O channels that you will use during the exercise.



DETAILED INSTRUCTIONS

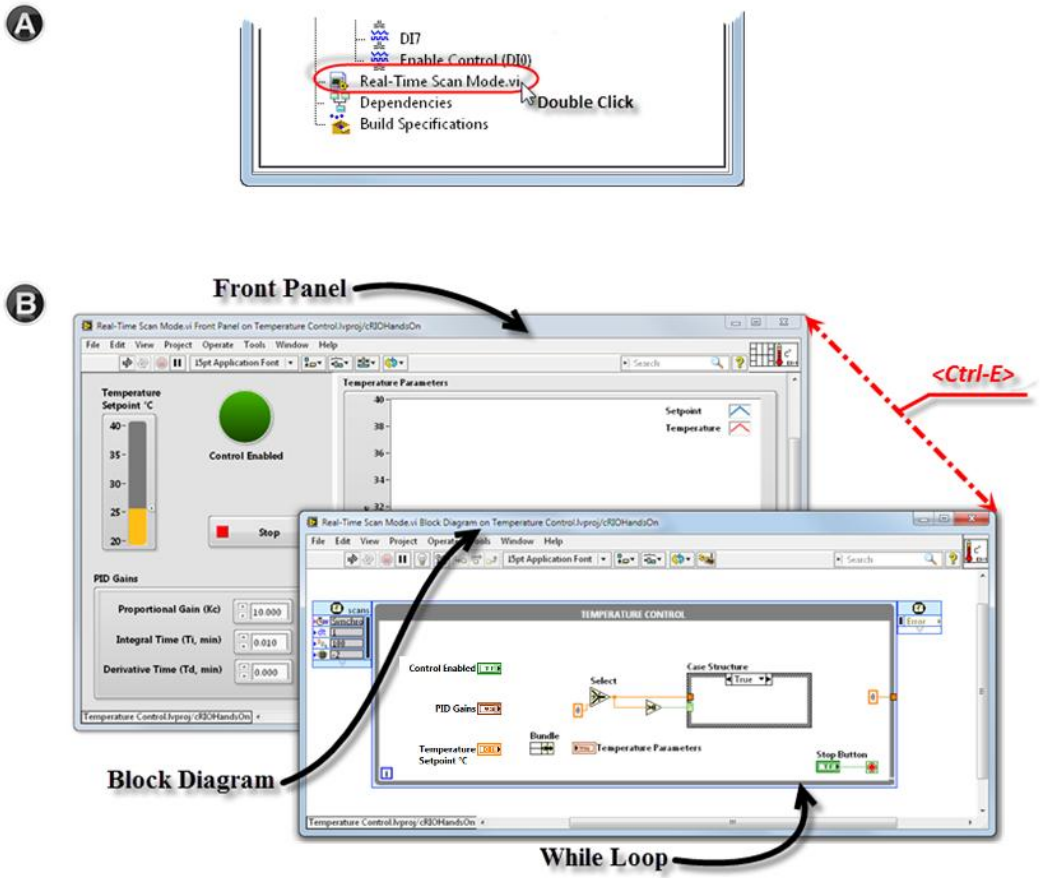
- ***Expand the Hierarchies on the Temperature Control.lvproj LabVIEW Project***
  This project already contains the ***CompactRIO system*** and the ***measurement modules*** that you are going to use during this exercise.

  To reveal the components of the project, expand the hierarchies in the project tree by clicking on the "**+**" boxes.

  Notice that the inputs and outputs of your application are already listed under the corresponding measurement modules.

3. Open the Real-Time Scan Mode.vi file.

This file already contains a front panel with a user interface and a block diagram with code that you will use to complete the temperature control application.



DETAILED INSTRUCTIONS

- ***Open the Real-Time Scan Mode.vi File***
  Double-click on the file named ***Real-Time Scan Mode.vi*** located at the bottom of the LabVIEW Project Explorer window as shown in ***Figure A***.

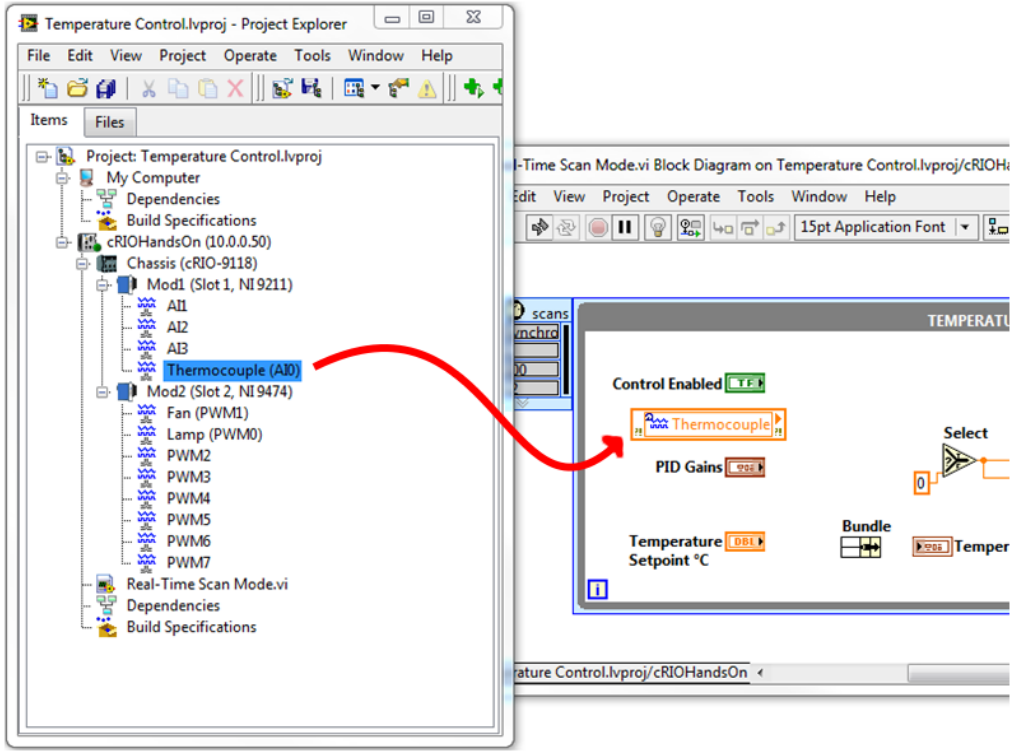
  This file consists of two windows: the ***front panel,*** which contains the user interface of the application, and the ***block diagram,*** which contains the code of the application.

- ***Show the Block Diagram***
  Press ***<Ctrl-E>*** or click on ***Window » Show Block Diagram*** to toggle between the block diagram and the front panel as shown in ***Figure B***.

  In this exercise, you will complete the code in the ***block diagram*** by adding the I/O measurement channels and the PID algorithm inside the ***While Loop*** named ***Temperature Control,*** which is synchronized with the ***Scan Engine*** with a scan period of ***100 ms***.

4. Add the inputs of the control algorithm to the block diagram.

Drag and drop the Enable Control and the Thermocouple I/O channels onto the block diagram.



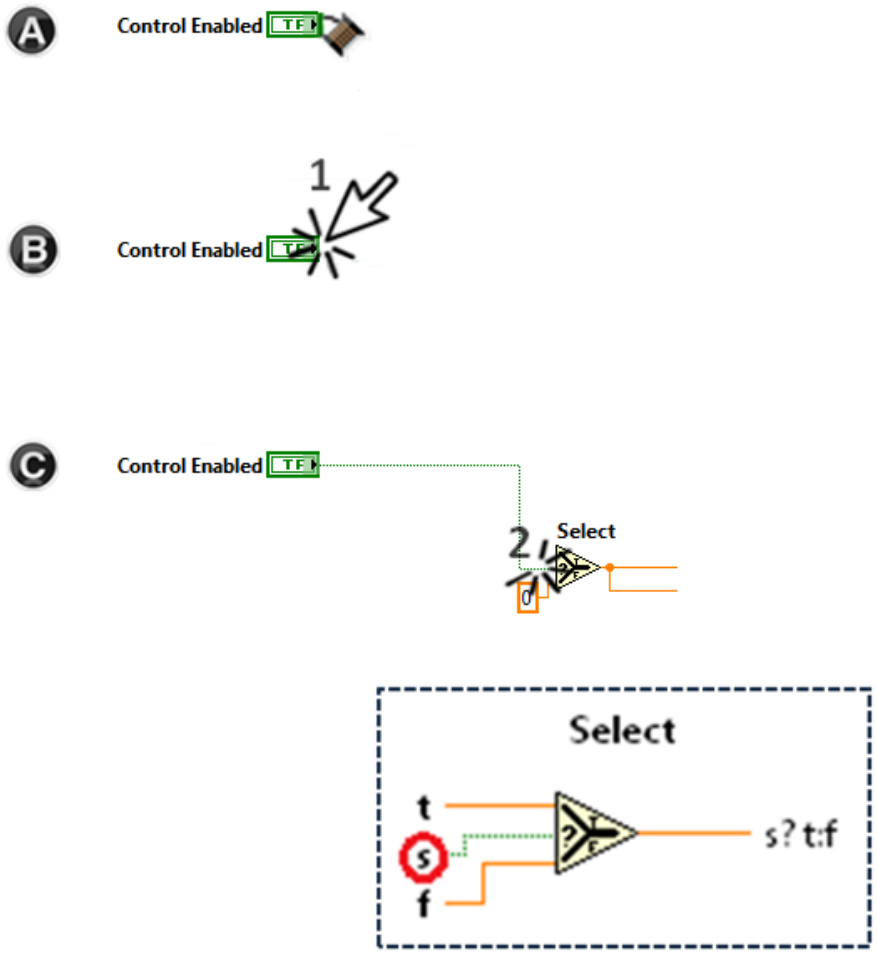
DETAILED INSTRUCTIONS

- ***Add the Thermocouple I/O Node***
  From the LabVIEW Project Explorer window, drag and drop the item labeled ***Thermocouple (AI0)*** under ***Mod1 (Slot1, NI 9211)*** onto the block diagram.

  Thanks to the ***Scan Mode,*** you can access the inputs of the control algorithm by simply dragging and dropping the required channels onto the block diagram.

5. Build the code for the conditional output of the control algorithm.

Wire the Control Enabled input terminal to the "s" terminal of the Select function.



DETAILED INSTRUCTIONS

- ***Wire the Control Enabled Terminal to the Select Function***

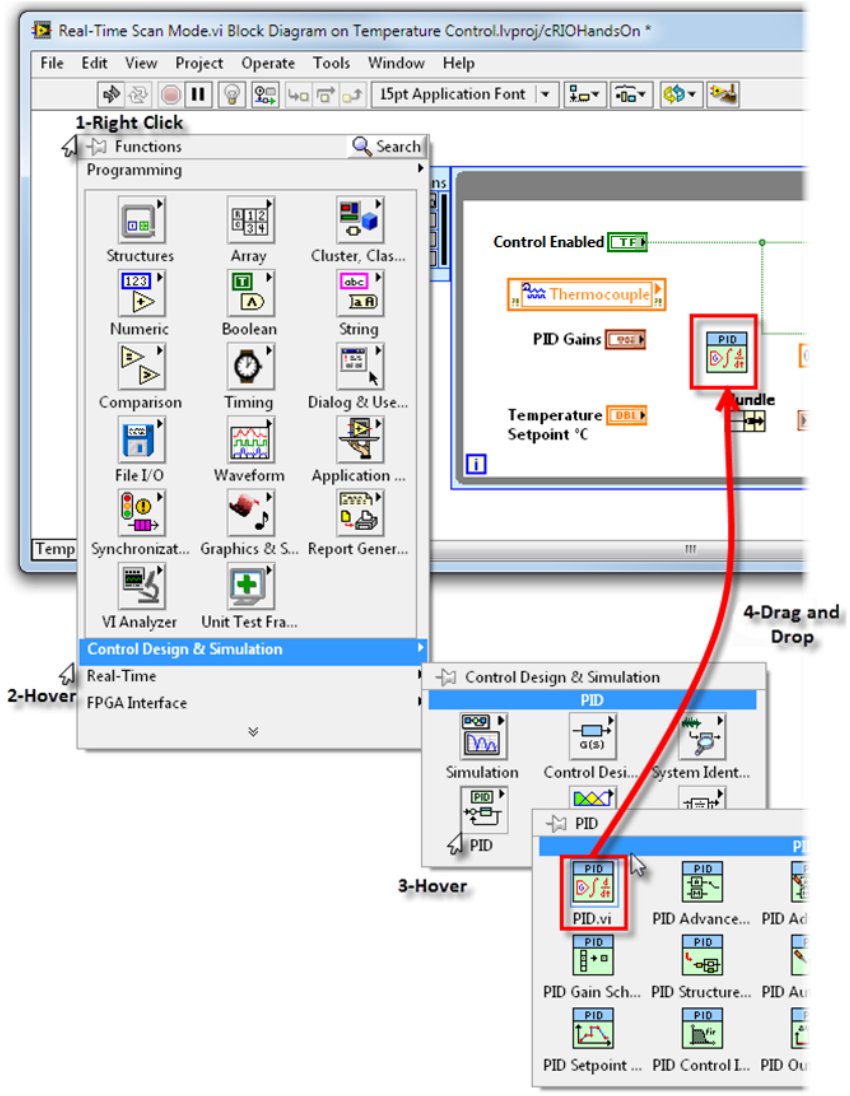
  Move the cursor near the ***Control Enabled*** terminal until it changes to the wiring tool as shown in ***Figure A***.

  With the cursor in wiring mode, make a click ***[1]*** on the Control Enabled input terminal as shown in ***Figure B***.

  Make a second click ***[2]*** on the ***"s"*** terminal of the ***Select function*** as shown in ***Figure C***. This is the selector switch that allows you to activate or deactivate the output of the control algorithm. This creates a wire between these two elements.

6. Integrate the PID algorithm into the code.

Add the PID.vi function to the block diagram.



DETAILED INSTRUCTIONS

- ***Add the PID.vi Function to the Block Diagram***

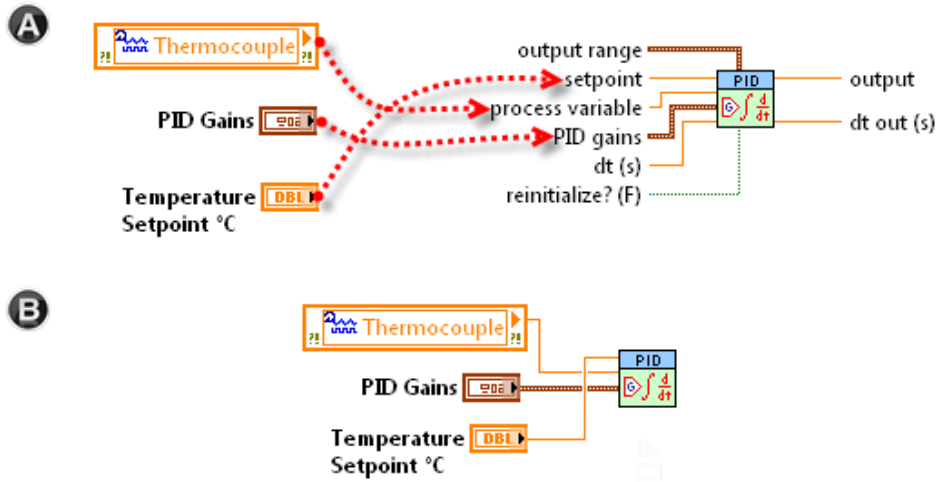
  Include the PID algorithm inside the ***While Loop*** to start building the application around it.

  Right-click on any section of the ***block diagram*** to open the ***Functions Palette***. In this palette, navigate to the following sections: ***Control Design & Simulation » PID » PID.vi***.

  Select the ***PID.vi*** function and drag and drop it inside the ***Temperature Control While Loop***. With this function, you can specify a setpoint, a process variable, and a set of gains to generate a control output for the fan and lamp in the temperature chamber.

7. Connect the inputs of the PID algorithm.

Wire the Thermocouple I/O, PID Gains, and Temperature Setpoint °C terminals to the corresponding inputs in the PID.vi as shown in the Figure A.

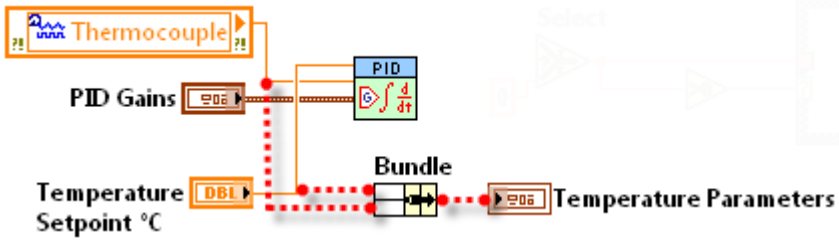


DETAILED INSTRUCTIONS

- ***Connect the Inputs of the PID Algorithm as Shown in Figure A***
  Wire the **Thermocouple I/O terminal** to the **process variable input** terminal of the PID.vi. Wire the **PID Gains control** to the **PID gains input** terminal of the PID.vi. Finally, wire the **Temperature Setpoint °C control** to the **setpoint input** terminal of the PID.vi. This is the minimum information required by the PID.vi function to perform the control of the temperature chamber. (***Figure B***)

8. Visualize the process variable and the setpoint on a graph in the user interface.

Use the Bundle function to visualize the process variable and the setpoint on the same graph.

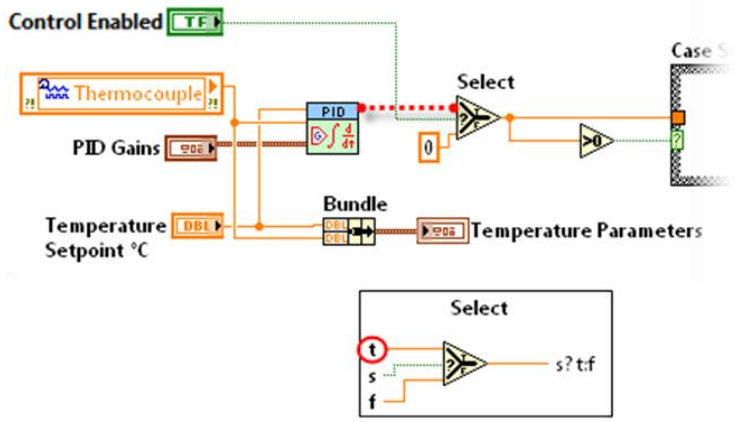


DETAILED INSTRUCTIONS

- ***Graph the Process Variable and the Setpoint***
  Wire the **Temperature Setpoint °C** output terminal to the first input terminal of the **Bundle function**. Wire the **Thermocouple I/O** terminal to the second input terminal of the **Bundle function**. Finally, wire the output terminal of the Bundle function to the ***Temperature Parameters graph***.

9. Close the control loop.

Wire the output of the PID.vi to the "t" terminal of the Select function.

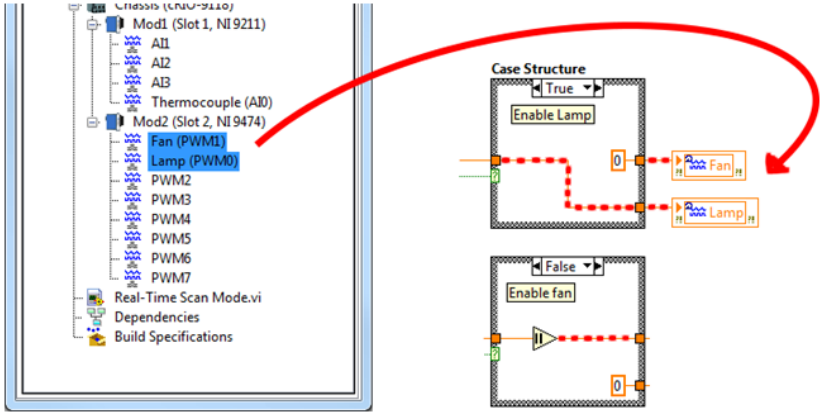


DETAILED INSTRUCTIONS

- ***Close the Control Loop***
  To complete the control loop, wire the output of the **PID.vi function** to the ***"t"*** terminal of the Select function.

10. Generate the outputs of the control algorithm through the CompactRIO system.

Drag and drop instances of the Lamp and Fan output terminals to provide outputs through the CompactRIO system.

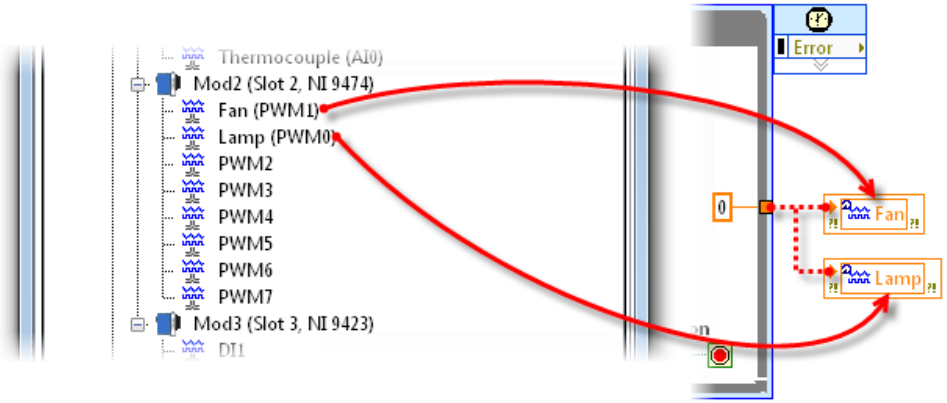


DETAILED INSTRUCTIONS

- ***Generate Outputs Through the CompactRIO System***
  Place an instance of the **Lamp** terminal and the **Fan** terminal and wire them with the outputs from the **Case Structure**.
  In the **True Case**, continue the wire coming from the Select function and wire it to the Lamp. In the **False Case** wire the output of the **Absolute Value Function** to the input of the Fan.

11. Provide shutdown states to the fan and lamp.

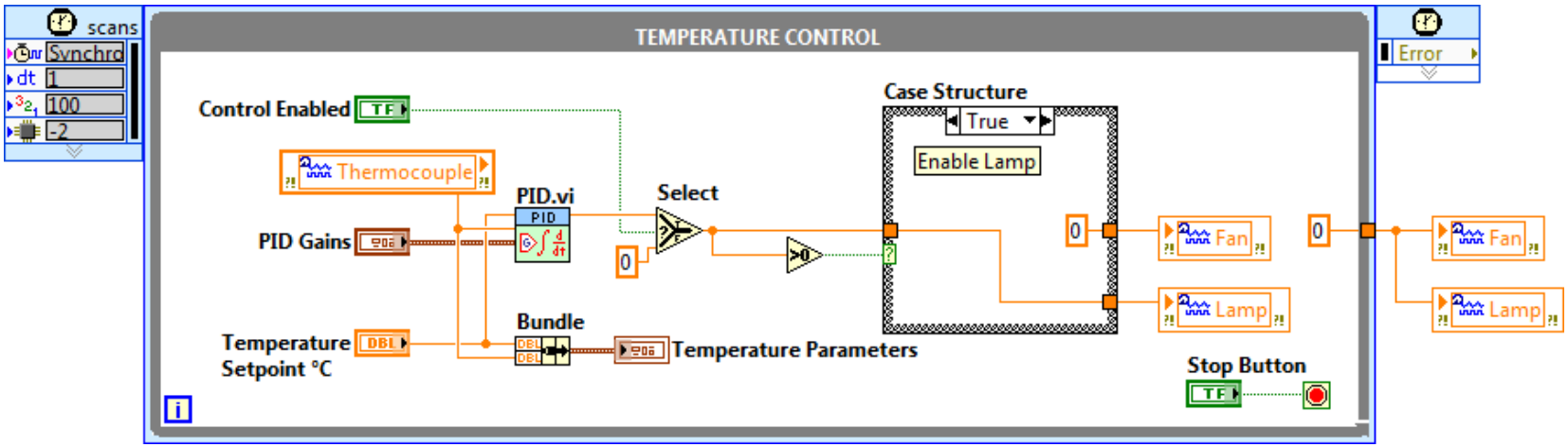
Define shutdown states for the control system by placing instances of the Fan and Lamp terminals outside the While Loop and wiring a constant with value 0 to these terminals.



DETAILED INSTRUCTIONS

- ***Provide a Shutdown State to the Control System***
  Place instances of the **Fan** and **Lamp** terminals outside the **While Loop** and continue the wire coming from the DBL constant with ***value 0*** to provide a shutdown state to these elements.

12. The completed block diagram should look like the image below.

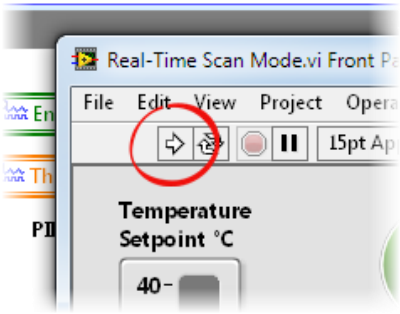


Part C—RUN THE APPLICATION

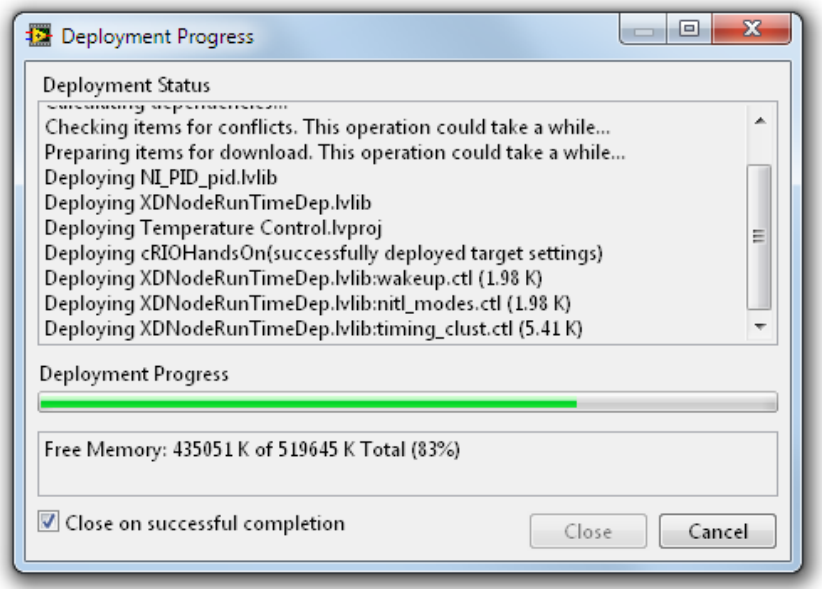
13. Run the application.

Run the Real-Time Scan Mode.vi by clicking on the execution arrow to deploy the code to the CompactRIO system.

A



B



DETAILED INSTRUCTIONS

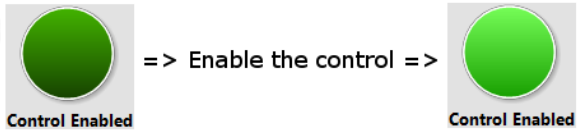
- ***Run the VI***
  Switch back to the front panel **<Ctrl-E>** and run the program by clicking on the **arrow button** at the top of the front panel as shown in ***Figure A***. Save the VI when prompted.

- ***Deploy the Code***
  Running the VI deploys the code through the ***Ethernet cable*** to the real-time controller of the ***CompactRIO system*** as shown in ***Figure B***, and initializes an interactive execution of the code in which the code is executed on the CompactRIO system while you monitor and control inputs and outputs from the user interface on the development machine.

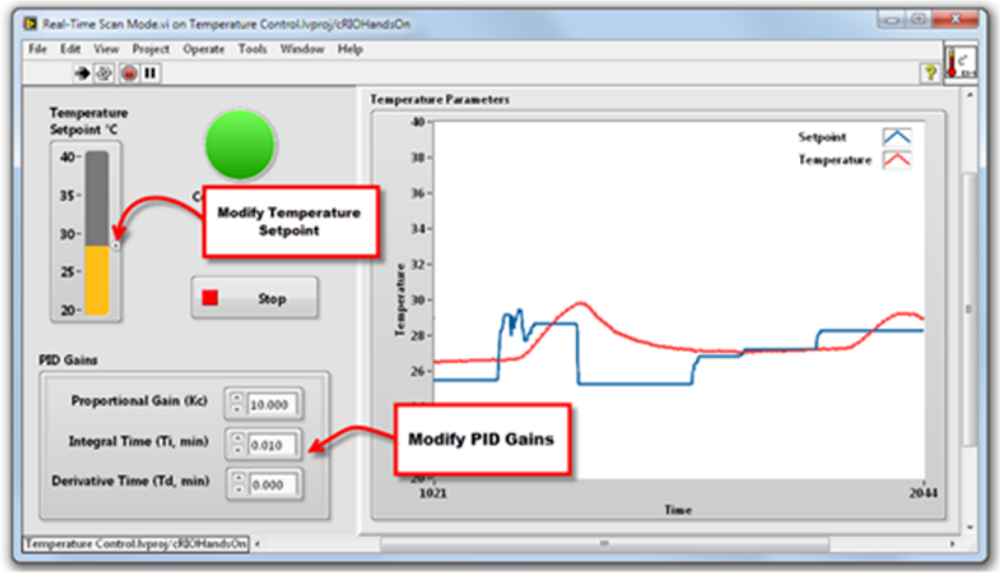
14. Interact with the system by modifying the setpoint and PID gains.

Press the **Control Enabled** switch on the front panel. Interact with the controls on the front panel to modify the temperature setpoint and the PID gains.

A



B



DETAILED INSTRUCTIONS

- ***Interact With the System to Test the Application***
  To enable the control output, press the **Control Enabled** button on the front panel as shown in ***Figure A***.

  Modify the value of the **Temperature Setpoint °C** control to change the setpoint of the system as shown in ***Figure B***. The PID algorithm then attempts to reach and hold the setpoint by powering the **Lamp** and the **Fan**.

  Experiment with different **PID Gains** to adjust the behavior of the control system.

- **Tip:** The Control Enabled button is used to decide, when to use the output from the PID controller. This could be replaced with a Digital Module, coming from a DI module like the NI-9421 or NI-9423.

15. Stop the application.

Stop the execution of the code by pressing the Stop button on the front panel.

Part D—CHALLENGE

16. Modify the visualization of the temperature parameters graph.

Modify the ***Temperature Parameters graph*** so that the ***Temperature line*** changes its color to red if it is above the ***Setpoint line*** and to green if it is below the ***Setpoint line***.

<END OF EXERCISE 1—TEMPERATURE CONTROL>

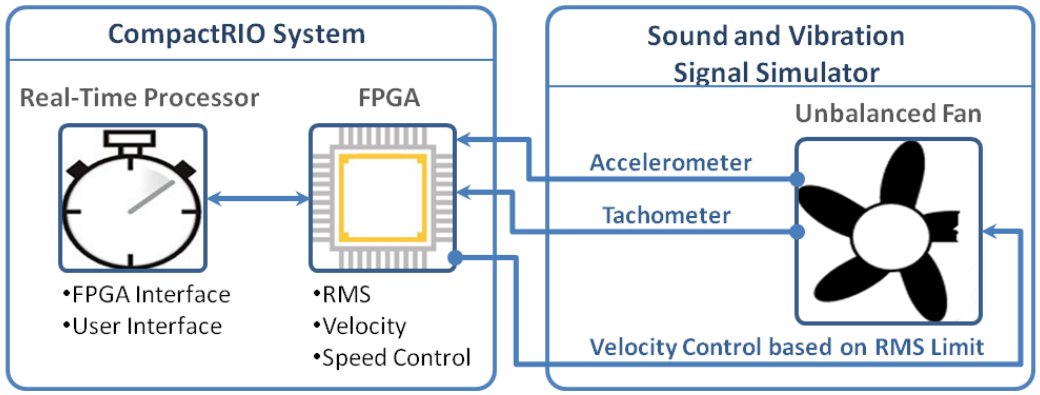
EXERCISE 2: VIBRATION MONITORING

Goals

- Use the ***FPGA*** portion of the ***CompactRIO system*** to implement a speed limit control algorithm for an unbalanced ***fan*** by monitoring signals from an ***accelerometer*** and a ***tachometer*** attached to it.
- ***Process*** the acquired signals on the ***FPGA*** to get the ***RMS*** value of the vibrations of the system and the ***velocity of the fan***.
- Use the ***real-time processor*** of the ***CompactRIO system*** to interface with the ***FPGA*** and generate a ***user interface*** to control and monitor the parameters of the system.

Part A—APPLICATION DESCRIPTION

Vibration Monitoring of an Unbalanced Fan

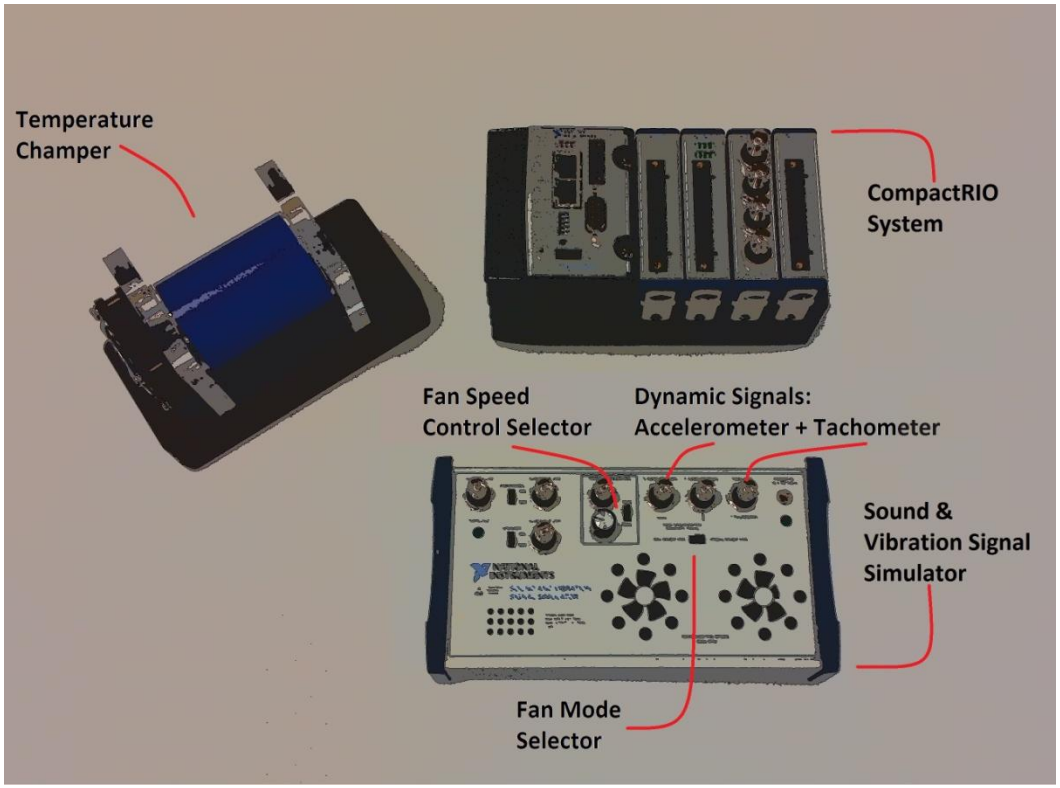


In this exercise, use ***LabVIEW*** to configure the ***FPGA*** chip in the ***CompactRIO system*** to create a ***speed limit control system.***

As shown in the diagram on the left, the FPGA portion of the architecture will be used to acquire signals from the ***Sound and Vibration Signal Simulator*** and control the speed of the fan in it.

Finally, the ***real-time processor*** will be used to implement a ***user interface*** to display the information produced by the FPGA.

CompactRIO Hands-On Kit



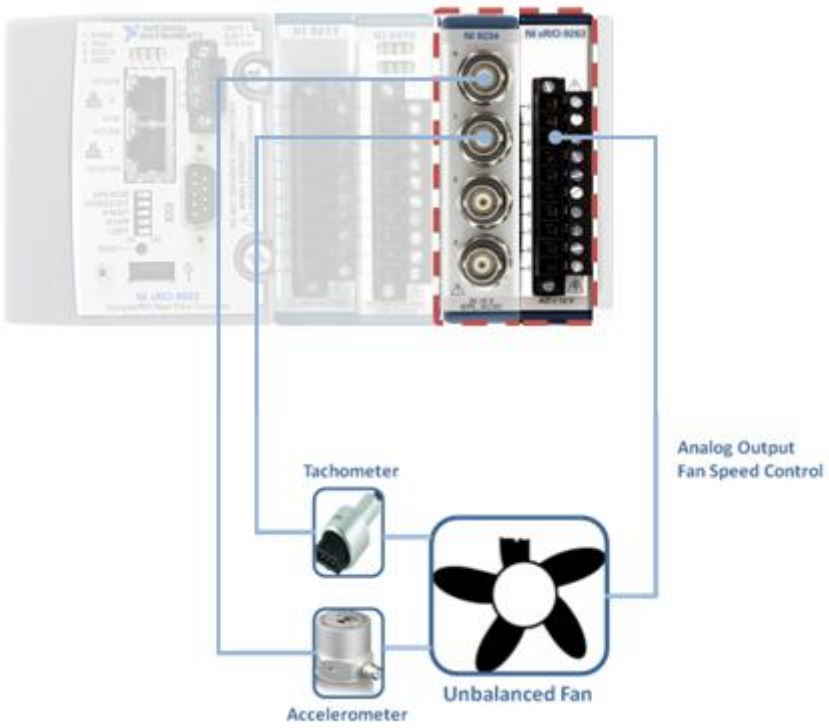
Locate the ***CompactRIO system*** and the ***Sound and Vibration Signal Simulator***.

The ***unbalanced fan*** is the rightmost fan of the Signal Simulator.

The ***Sound and Vibration Signal Simulator*** provides an ***unbalanced fan*** with the top speed of ***6000 rpm,*** an ***accelerometer*** with sensitivity of ***175 mV/g***, and a ***tachometer*** with a resolution of ***2 pulses/rev*.**

For this exercise, make sure that the ***Fan Speed Control Selector*** in the Sound and Vibration Signal Simulator is on ***BNC mode***, and that the ***Fan Mode Selector*** is on ***Unbalanced mode***.

CompactRIO System



Field-Programmable Gate Array (FPGA)

An FPGA is a reprogrammable silicon chip. In contrast to programming the processors in your PC, programming an FPGA "physically" rewires the chip itself to implement your functionality rather than run a software application.

Real-Time Operating System (RTOS)

An RTOS is specially design to manage hardware resources and run applications with very precise timing and a high degree of reliability.

In this exercise, use measurement modules in ***slots 3 and 4*** of the **CompactRIO *system***.

Inputs

Slot 3: NI 9234—*Dynamic Signal Acquisition Module*
AI0→Accelerometer
AI1→Tachometer
Sampling Period: 51.2 kS/s
Resolution: 24-bit

Outputs

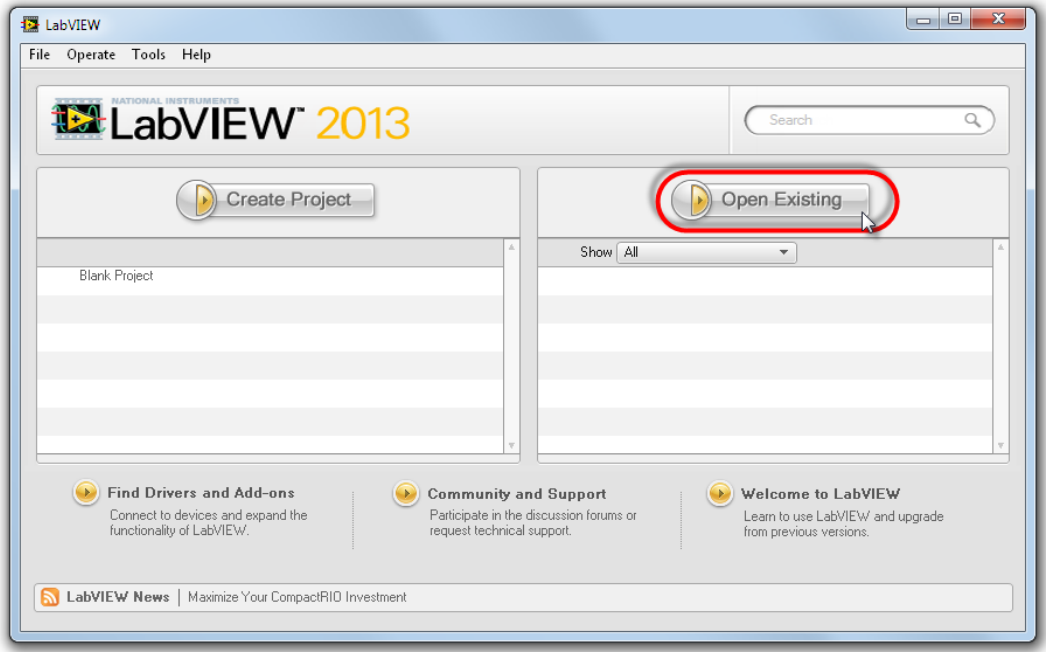
Slot 4: NI 9263—*Analog Output Module*
AO0→Fan Speed Control
Sampling Period: 100 kS/s
Resolution: 16-bit

Part B—FPGA CODE IMPLEMENTATION

1. Open the Vibration Monitoring.lvproj.

Launch LabVIEW and open the existing LabVIEW project Vibration Monitoring.lvproj.

C:\CompactRIO HO\Exercise 2\Vibration Monitoring.lvproj



DETAILED INSTRUCTIONS

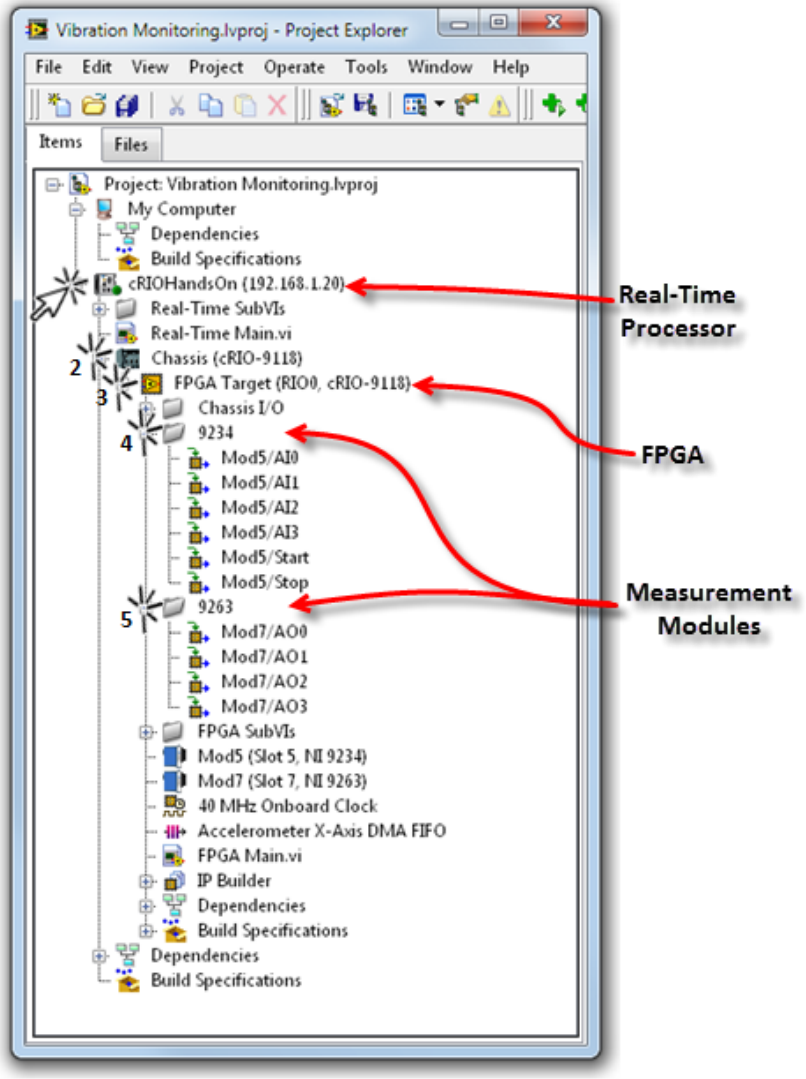
In this exercise, use an existing **LabVIEW project** as the starting point of the application.

- ***Launch LabVIEW***
  Launch LabVIEW by navigating to ***Start » All Programs » National Instruments » LabVIEW 2013 » LabVIEW***.

- ***Open an Existing LabVIEW Project***
  Click on the ***Open Existing*** button on the right side of the ***LabVIEW*** main screen and open the ***LabVIEW Project*** file named ***Vibration Monitoring.lvproj*** located at:
  C:\CompactRIO HO\Exercise 2\Vibration Monitoring.lvproj

2. Explore the project and reveal its components.

Expand each hierarchy on the project to reveal the real-time processor and the FPGA of the CompactRIO system.



DETAILED INSTRUCTIONS

- ***Expand the Hierarchies on the Vibration Monitoring .lvproj LabVIEW Project.***
  This project already contains the **CompactRIO** system exposing the **FPGA** and the measurement modules that you are going to use.

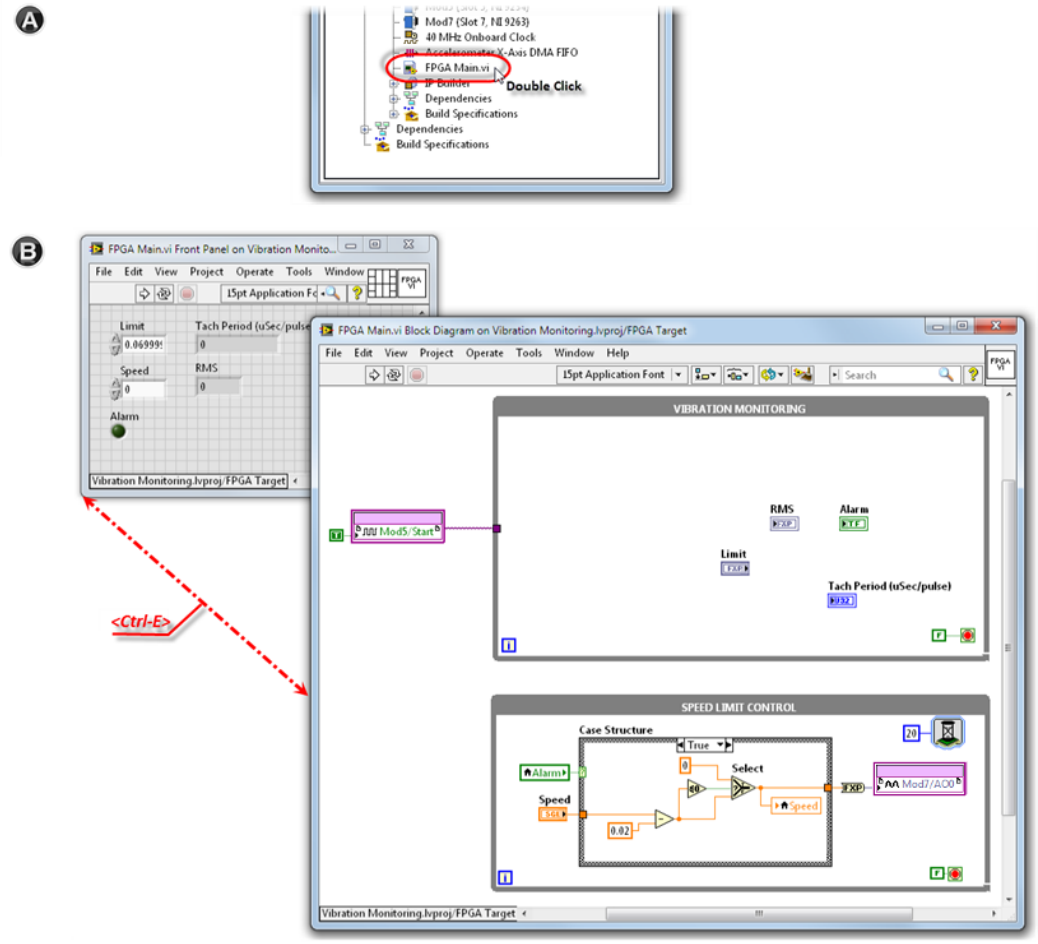
  Notice that the measurement modules are located under the FPGA hierarchy.

  To reveal the components of the project, expand the hierarchies in the project tree by clicking on the "**+**" boxes.

The slot number might not be the same in your exercise. Please refer to the setup page of your compactRIO system (p. 22). In this exercise, we will be using two modules, and you can always tell them apart by looking at the **model number** (i.e. 9234 or 9263) or by the fact that the signal is either **Analog Input** or **Analog Output**.

3. Open the FPGA Main.vi and show its block diagram.

This FPGA Main.vi is located under the FPGA hierarchy and it already contains code to use for the vibration monitoring application.



DETAILED INSTRUCTIONS

- ***Open the FPGA Main.vi***
  Double-click on the file named **FPGA Main.vi** located under the FPGA hierarchy as shown in ***Figure A***.

- ***Show the Block Diagram***
  Press ***<Ctrl-E>*** or click on ***Window » Show Block Diagram*** to toggle between the block diagram and the front panel as shown in ***Figure B***.

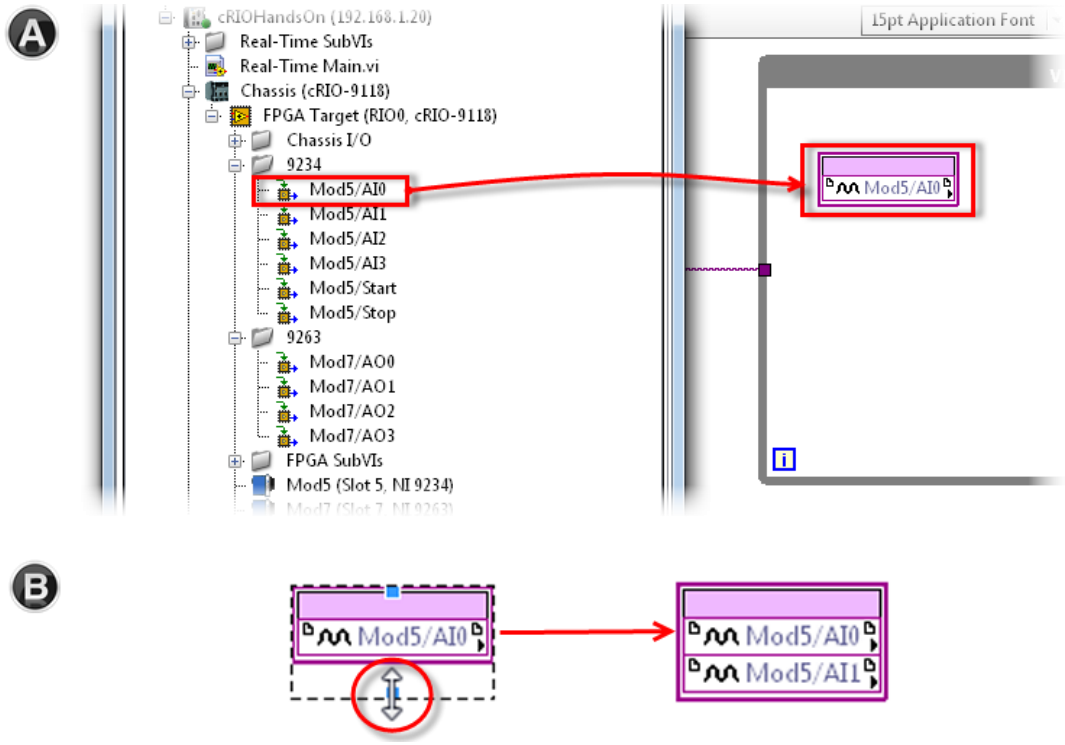
This file contains two parallel loops:

- ***Vibration Monitoring Loop***
  Add code to this loop to acquire the accelerometer and tachometer signals. Also include the ***RMS*** and ***Velocity*** analysis in this loop.

- ***Speed Limit Control Loop***
  This already completed loop controls the speed of the ***Fan*** based on an ***RMS Alarm Level***.

4. Add the accelerometer and tachometer channels to the block diagram.

Drag and drop the AI0 I/O node of the NI 9234 C Series measurement module into the Vibration Monitoring Loop. Expand the I/O node to show the AI1 terminal as well.



The slot number might not be the same in your exercise. Please refer to the setup page of your compactRIO system (p. 22).
In this exercise, we will be using two modules, and you can always tell them apart by looking at the **model number** (i.e. 9234 or 9263) or by the fact that the signal is either **Analog Input** or **Analog Output**.

DETAILED INSTRUCTIONS

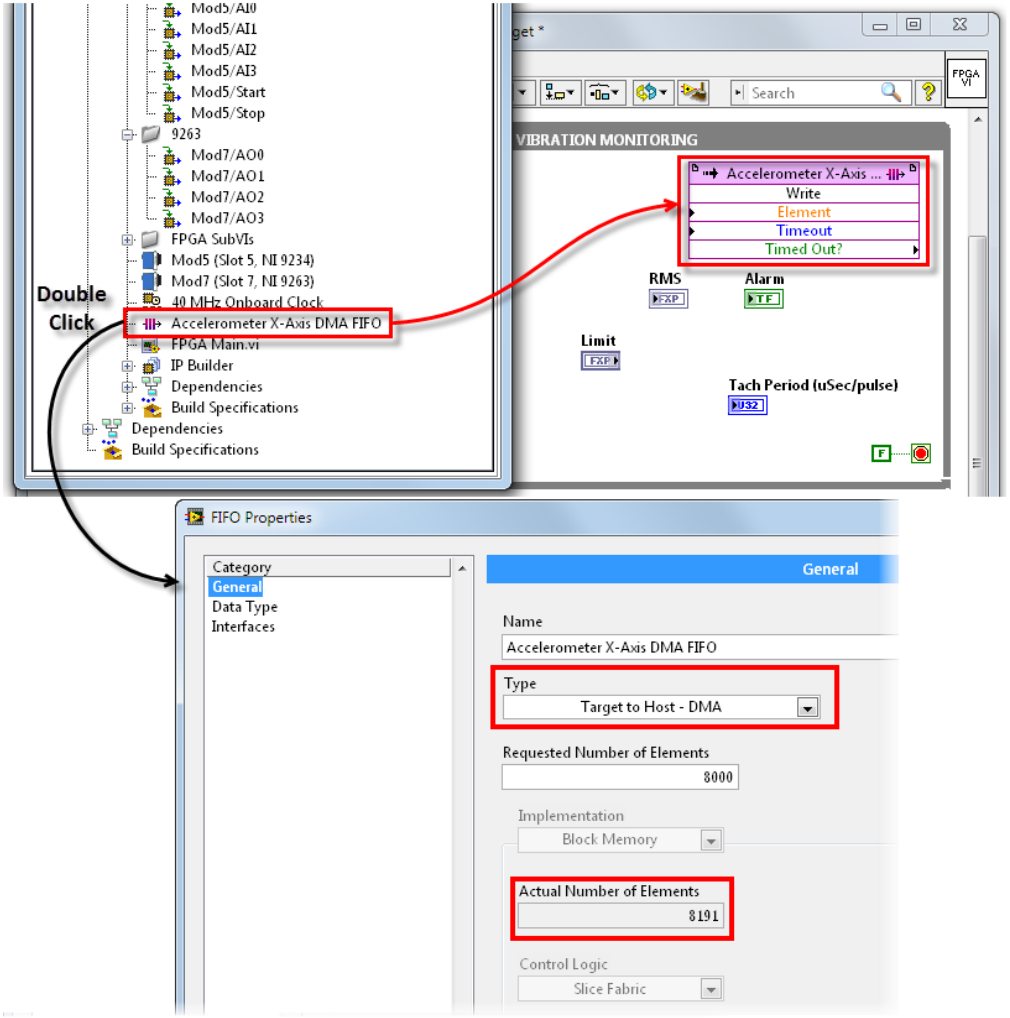
- ***Drag and Drop the AI0 Into the Vibration Monitoring Loop***
  Click on the ***AI0*** terminal located in the LabVIEW project under the NI 9234 measurement module of the FPGA hierarchy as shown in ***Figure A***. Drag and drop the terminal onto the block diagram and place it inside the ***Vibration Monitoring Loop***.

- ***Expand the I/O Node to Show the AI1 Terminal***
  Move the cursor over the I/O node to show the blue points from which you can expand the node. Expand the node to show the additional terminal ***AI1*** as shown in ***Figure B***.

  The two analog inputs are used for:
  AI0→Accelerometer
  AI1→Tachometer

5. Add the communication channel between the FPGA and the real-time processor.

Drag and drop the DMA FIFO named Accelerometer X-Axis DMA FIFO onto the Vibration Monitoring Loop.



DETAILED INSTRUCTIONS

- ***Drag and Drop the Accelerometer X-Axis DMA FIFO Onto the Block Diagram***
  Click on the ***Accelerometer X-Axis DMA FIFO*** node located in the LabVIEW project under the FPGA hierarchy. Drag and drop the node onto the block diagram and place it inside the ***Vibration Monitoring Loop***.

  With this function, you can transfer data from the ***FPGA*** to the ***real-time processor***.

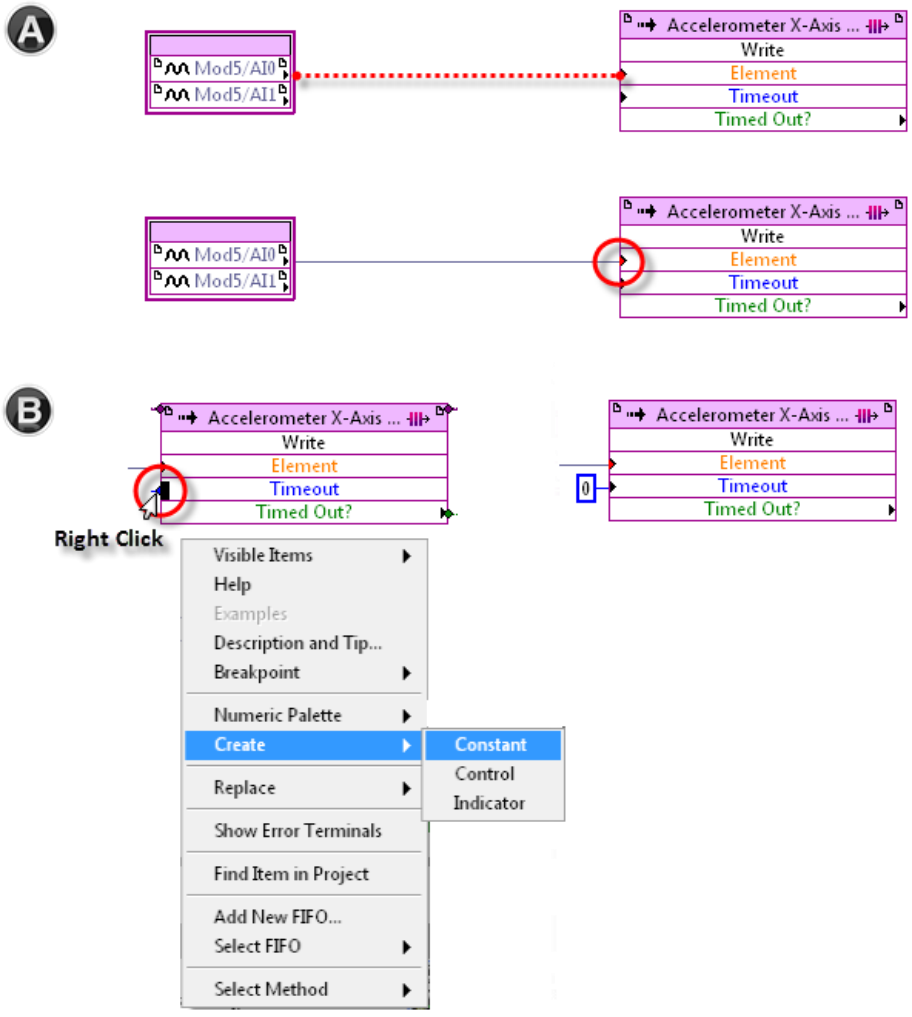
  The ***Accelerometer X-Axis DMA FIFO*** has been preconfigured as follows:

  - Target to Host-DMA
  - 8191 Elements
  - Single precision floating-point

  Double click on the ***Accelerometer X-Axis DMA FIFO*** to open its configuration window.

6. Transfer the accelerometer data to the real-time processor.

Wire the AI0 terminal to the Element terminal of the Accelerometer X-Axis DMA FIFO. Wire a constant with value 0 to the Timeout terminal of the Accelerometer X-Axis DMA FIFO.



DETAILED INSTRUCTIONS

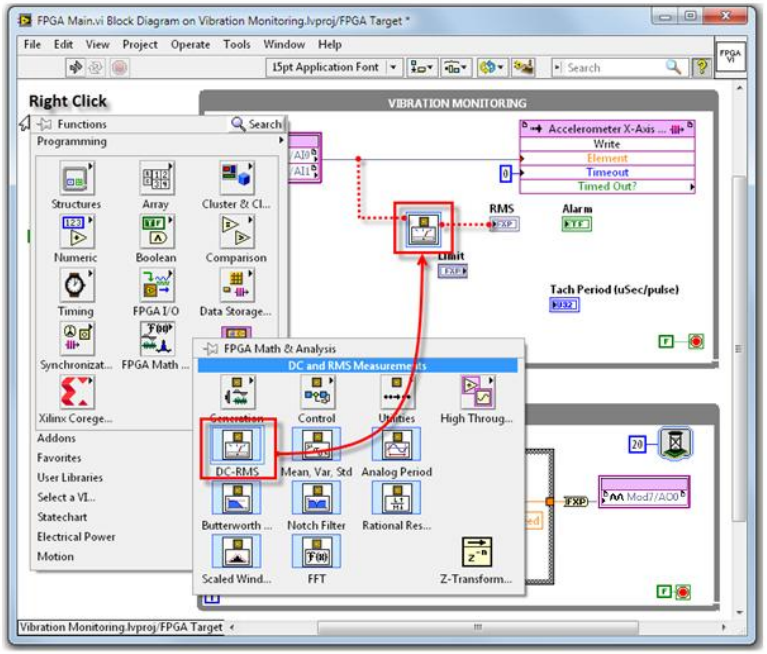
- ***Wire the AI0 Terminal to the Element Terminal of the Accelerometer X-Axis DMA FIFO as Shown in Figure A***
  When this wire is created, LabVIEW automatically creates a ***coercing point*** (red dot) to convert from a fixed-point data type to a single-precision floating-point data type.

- ***Wire a Constant With Value 0 to the Timeout Terminal of the Accelerometer X-Axis DMA FIFO***
  Right-click on the ***Timeout terminal*** of the ***Accelerometer X-Axis DMA FIFO*** and select ***create » constant*** from the pull-down menu as shown in ***Figure B***. A constant with ***value 0*** indicates that the method does not wait for available space in the FIFO to execute.

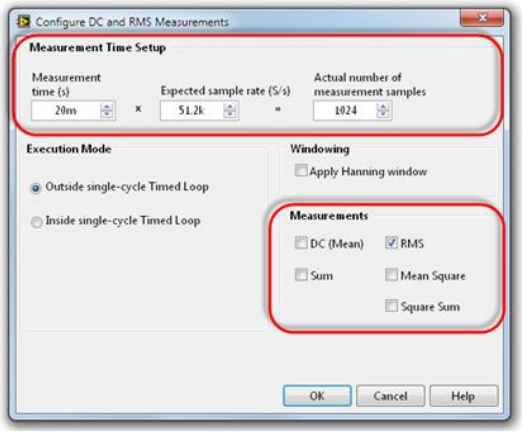
7. Obtain the RMS value of the accelerometer signal.

Add the DC-RMS function to the block diagram. Wire the AI0 terminal and the RMS indicator to the DC-RMS function.

A



B



DETAILED INSTRUCTIONS

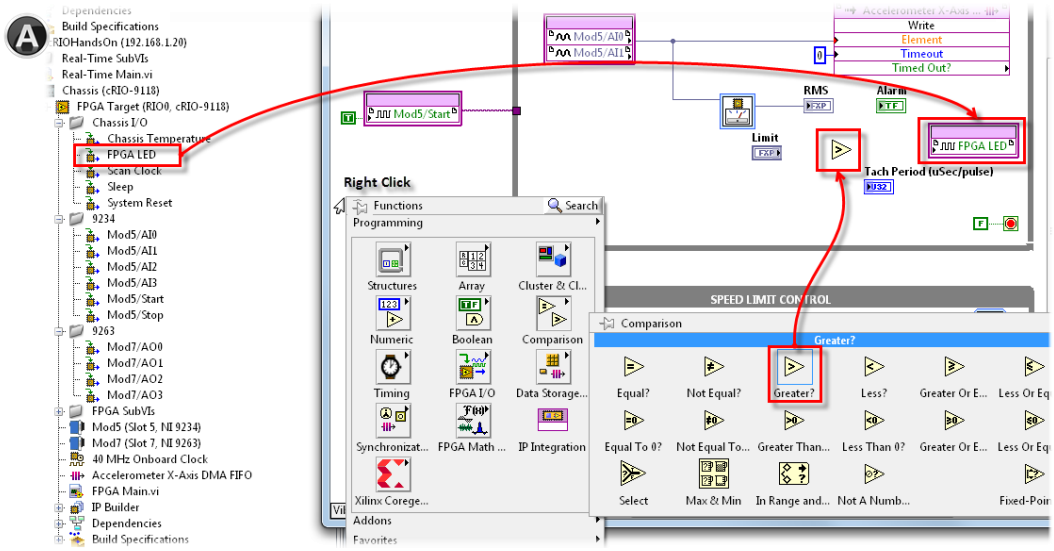
- ***Add the DC-RMS Function to the Block Diagram***
  Right-click on the block diagram to open the ***Functions palette***. Navigate to ***Programming » FPGA Math & Analysis » DC-RMS***. Drag and drop the ***DC-RMS function*** onto the ***block diagram*** as shown in ***Figure A***. Configure the function as shown in ***Figure B*** using the following parameters:

  Measurement time (s): 20 m
  Expected sample rate (S/s): 51.2 k
  Number of measurement samples: 1024
  Measurements: RMS

- ***Wire the AI0 Terminal and the RMS Indicator to the DC-RMS Function as Shown in Figure A***
  Create a new branch from the wire created in the previous step to connect the ***AI0*** terminal to the ***input data terminal*** of the DC-RMS function. Additionally, wire the ***RMS output terminal*** of this function to the ***RMS indicator***.

8. Generate the alarm based on the RMS limit.

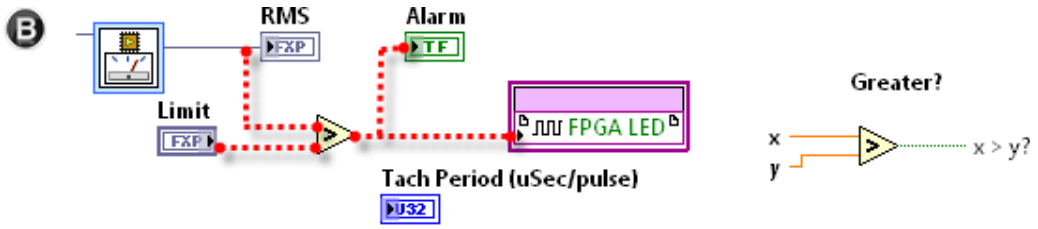
Add a Greater? function and the FPGA LED node to complete the code for the alarm based on the RMS value. Wire the diagram as shown in the picture below.

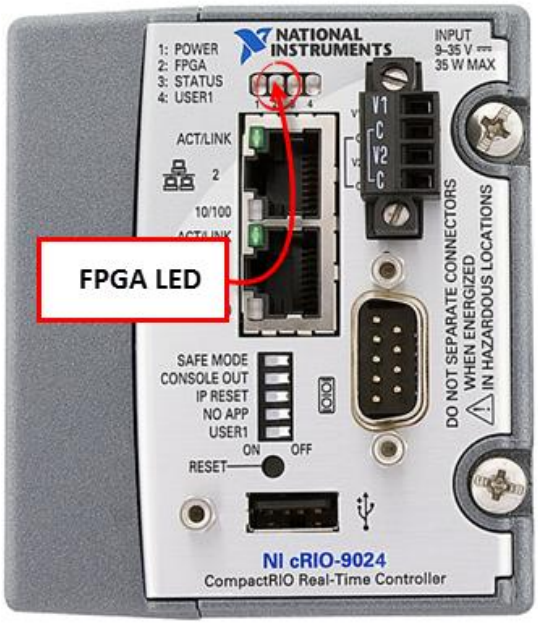


DETAILED INSTRUCTIONS

- ***Add the Greater? Function***
  Right-click on the ***block diagram*** to open the ***Functions palette***. Navigate to ***Programming » Comparison » Greater?*** Drag and drop the ***Greater? function*** onto the block diagram as shown in ***Figure A***.

- ***Add the FPGA LED Node***
  From the project tree, drag and drop the ***FPGA LED node*** located under the ***Chassis I/O folder*** of the ***FPGA*** hierarchy as shown in ***Figure A***.



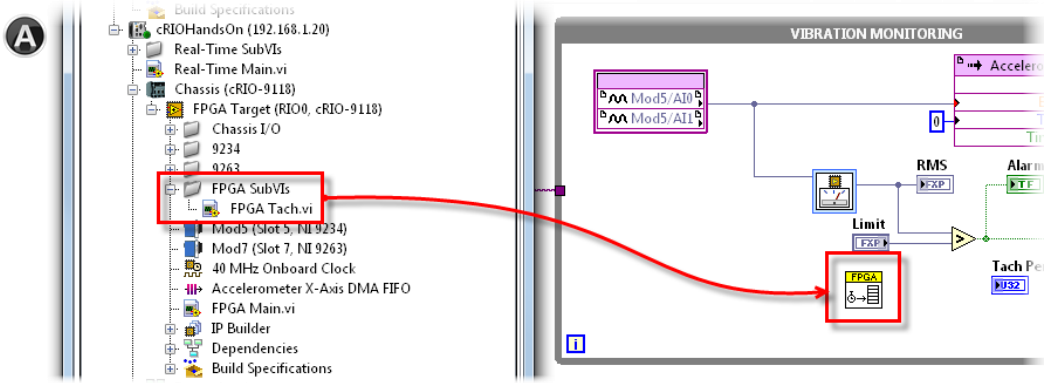


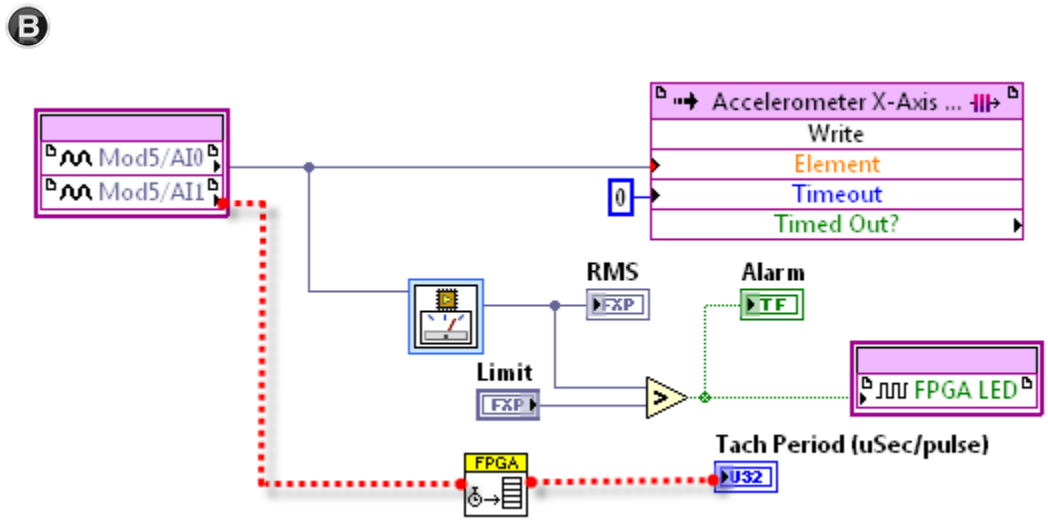
- ***Wire the Diagram as Shown in Figure B***
  Wire the ***RMS value*** to the "***x***" input of the ***Greater? function***. Wire the ***Limit indicator*** to the "***y***" input of the ***Greater? function***. Finally, wire the ***output of the Greater? function*** to the ***Alarm indicator*** and the ***FPGA LED node***.

  This shows the alarm in both the front panel and the ***FPGA LED*** in the controller of the ***CompactRIO system***.

9. Get the tachometer period.

Add the FPGA Tach.vi from the FPGA SubVIs folder and wire the diagram as indicated in the image below.





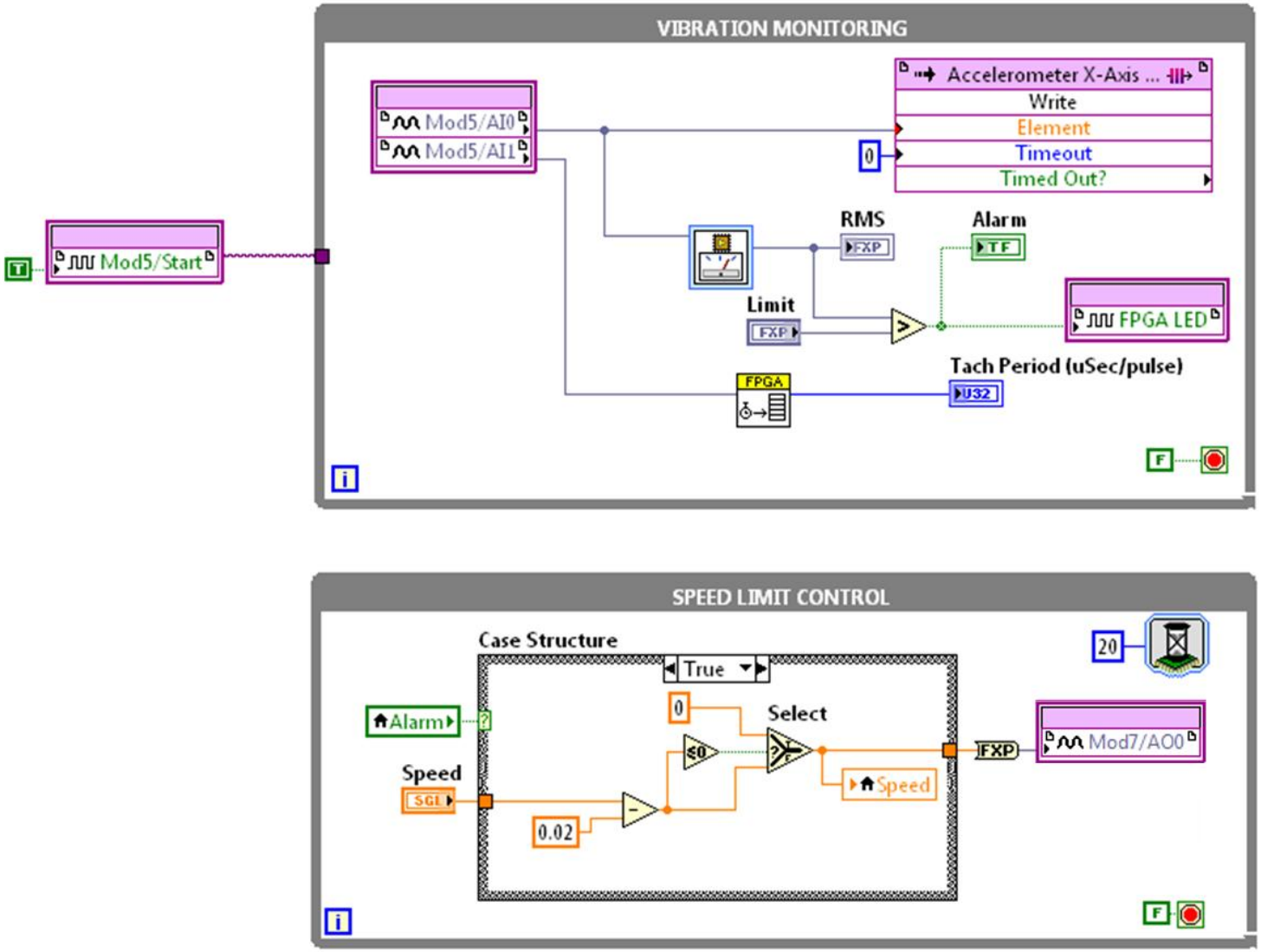
DETAILED INSTRUCTIONS

- ***Add the FPGA Tach.vi***
  From the LabVIEW project, navigate to the folder named ***FPGA SubVIs*** located under the FPGA hierarchy. Drag and drop the ***FPGA Tach.vi*** onto the block diagram as shown in ***Figure A***.

  This file is a prebuilt function that you will reuse to get the period of the tachometer signal.

- ***Wire the Inputs and Outputs of the FPGA Tach.vi as Shown in Figure B***
  Wire the ***AI1*** to the ***Analog In (Tach) terminal*** of the ***FPGA Tach.vi***. Wire the output of this VI to the ***Tach Period (uSec/Pulse) indicator***.

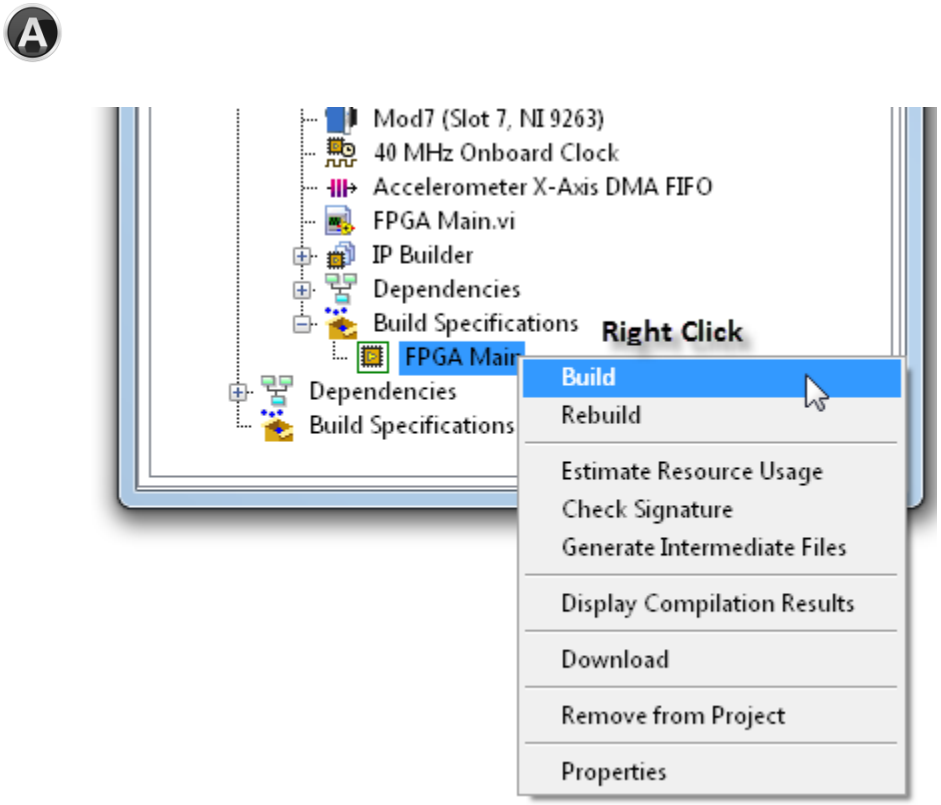
10. The completed block diagram for the FPGA Main.vi should look like the image below.



Part C—FPGA COMPILATION PROCESS

11. Save and compile the FPGA Main.vi.

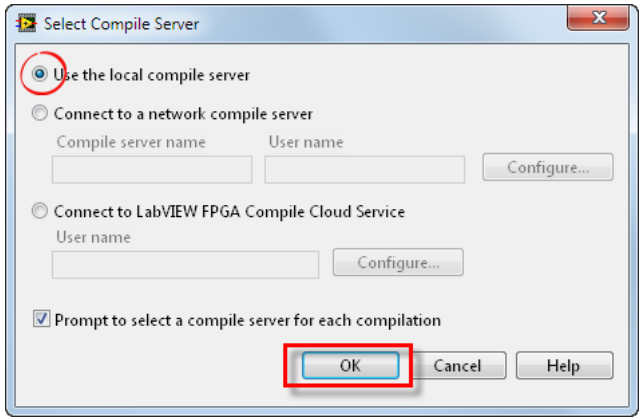
The compilation process of the FPGA Main.vi takes about 15 minutes. While this is done, continue working on the real-time portion of the application in **Part D**.

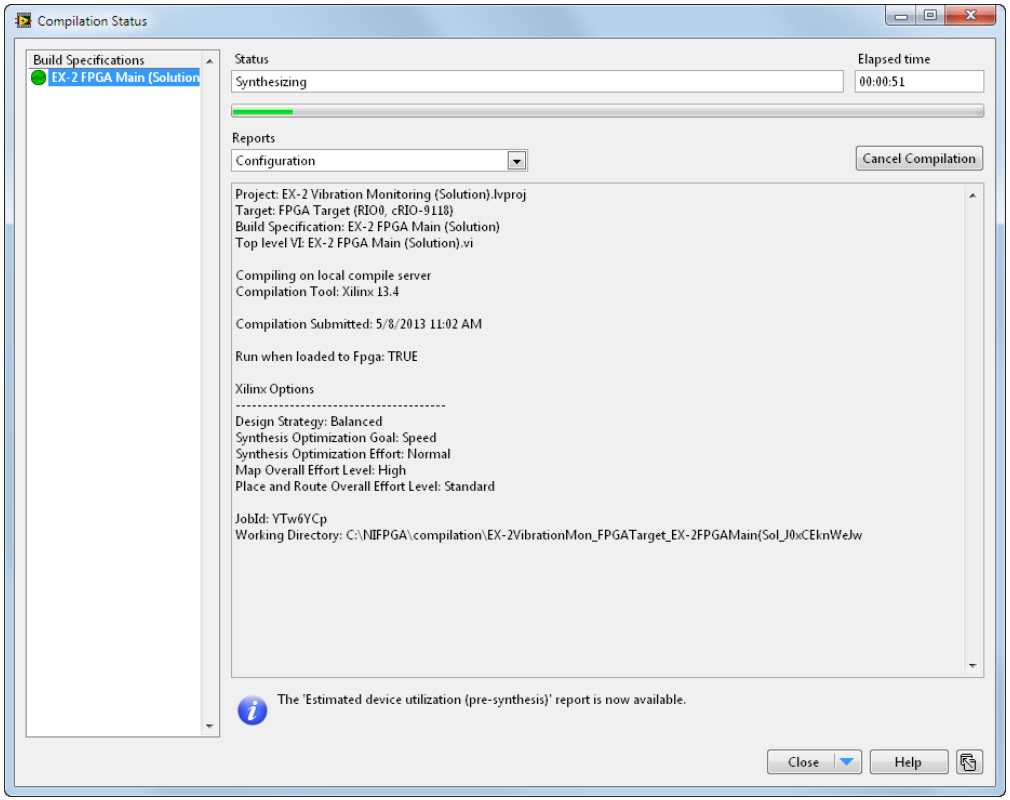


DETAILED INSTRUCTIONS

- ***Save the FPGA Main.vi***
  Press ***<Ctrl-S>*** or go to ***File » Save.***

- ***Compile the FPGA Main.vi***
  Navigate to the ***Build Specifications*** category under the FPGA hierarchy. Expand this category and right-click on the existing ***FPGA Main specification*** and select ***Build*** as shown in ***Figure A***.





LabVIEW will prompt you to select the ***compile server***. Select the ***local compile server*** and click ***OK*** as shown in ***Figure B***. This will start the compilation process.

During the compilation process, ***LabVIEW*** generates intermediate HDL files that are later processed by the ***Xilinx Compiler,*** which outputs a bitfile containing the placing and routing information of the FPGA design.

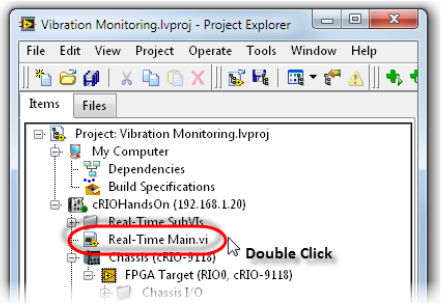
The compilation process for this VI takes **about 15 minutes**. Once you have reached the Compilation Status window, ***PROCEED TO PART D*** to start developing the real-time portion of the application.

Part D—REAL-TIME CODE IMPLEMENTATION

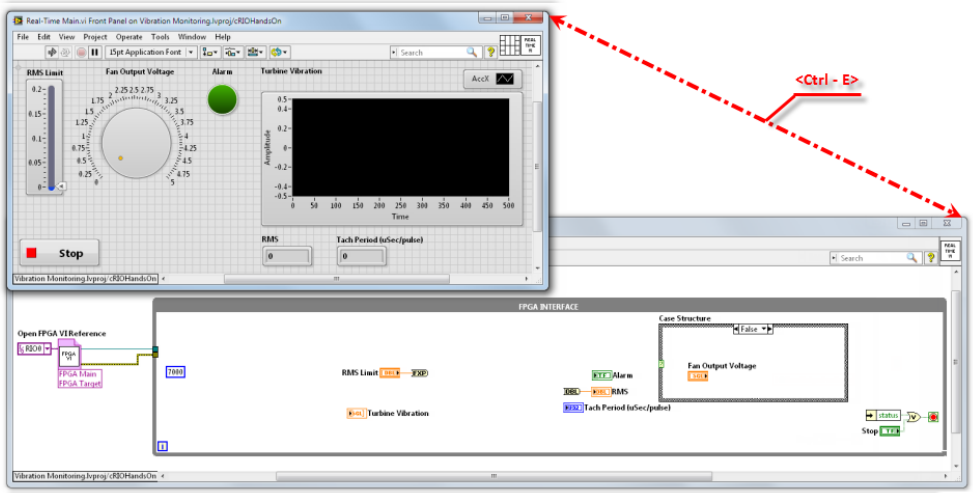
12. Open the Real-Time.vi and show its block diagram.

The Real-Time.vi already contains code to interface with the FPGA portion of the application.

A



B



DETAILED INSTRUCTIONS

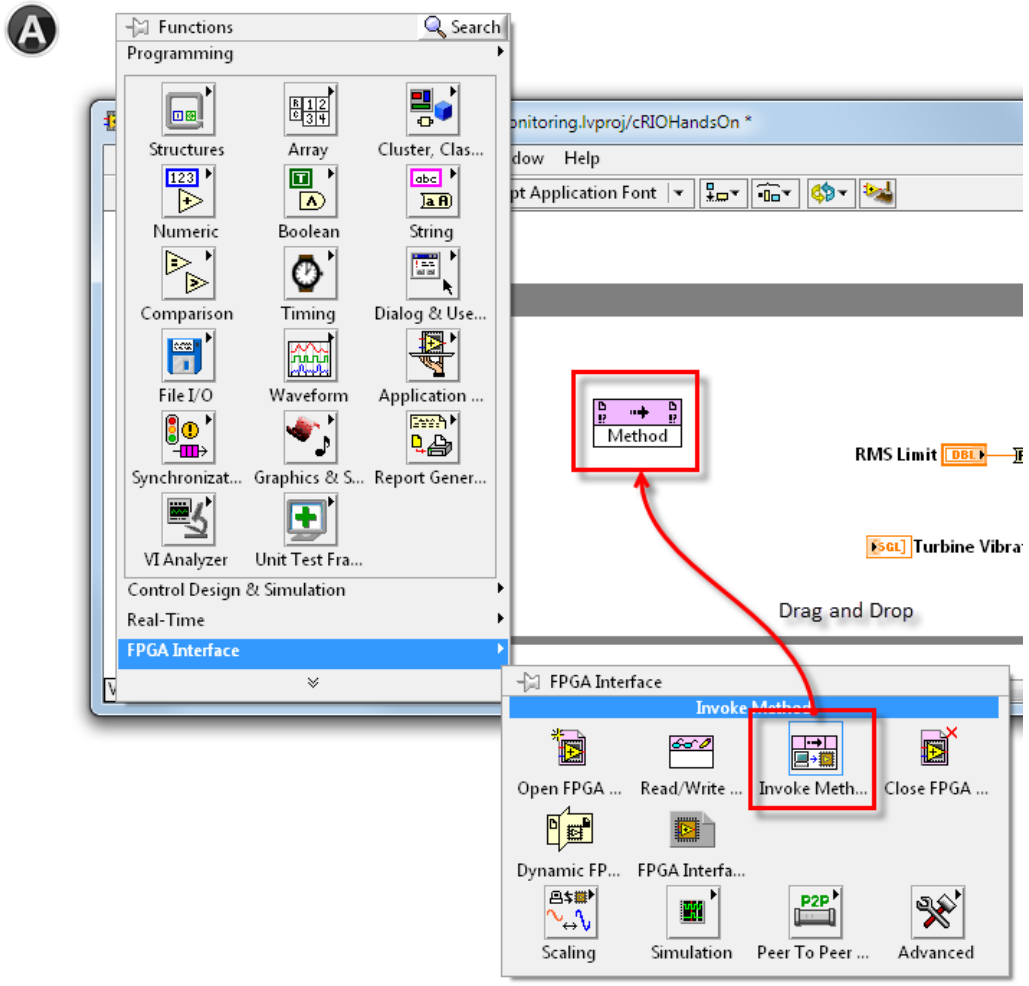
- ***Open the Real-Time Main.vi***
  Double-click on the file named **Real-Time Main.vi** located under the cRIOHandsOn hierarchy as shown in ***Figure A***.

- ***Show the Block Diagram***
  Press ***<Ctrl-E>*** or click on ***Window » Show Block Diagram*** to toggle between the block diagram and the front panel as shown in ***Figure B***.

This file already contains code to implement an interface to the FPGA code and provide visualization for the user.

13. Get the accelerometer data and display it on a graph.

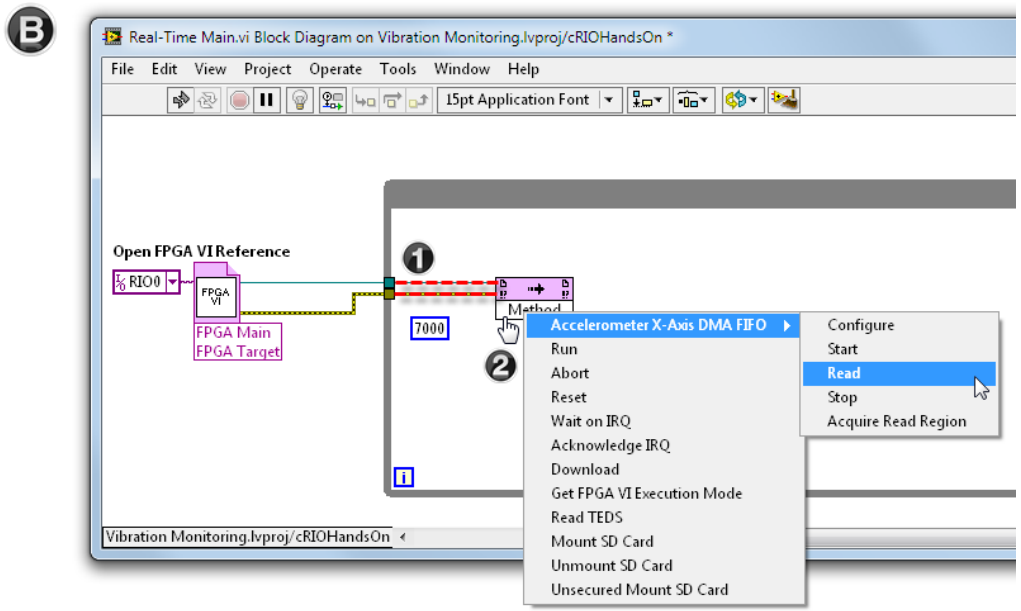
Place an Invoke Method from the FPGA Interface palette and configure it to read data from the Accelerometer X-Axis DMA FIFO. Wire the diagram as shown in Figure C.

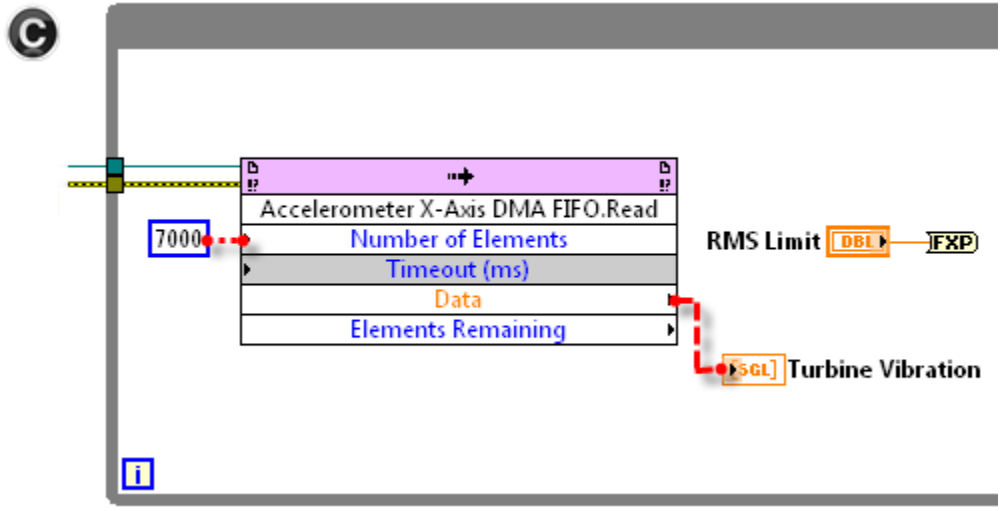


DETAILED INSTRUCTIONS

- ***Add the Invoke Method***
  Right-click on the ***block diagram*** to open the ***Functions palette***. Navigate to ***FPGA Interface » Invoke Method***. Drag and drop the ***Invoke Method node*** onto the ***block diagram*** as shown in ***Figure A***.

  This node invokes a method or action from a host VI on an FPGA VI. In this exercise, use it to access the ***DMA FIFO*** containing the accelerometer data.





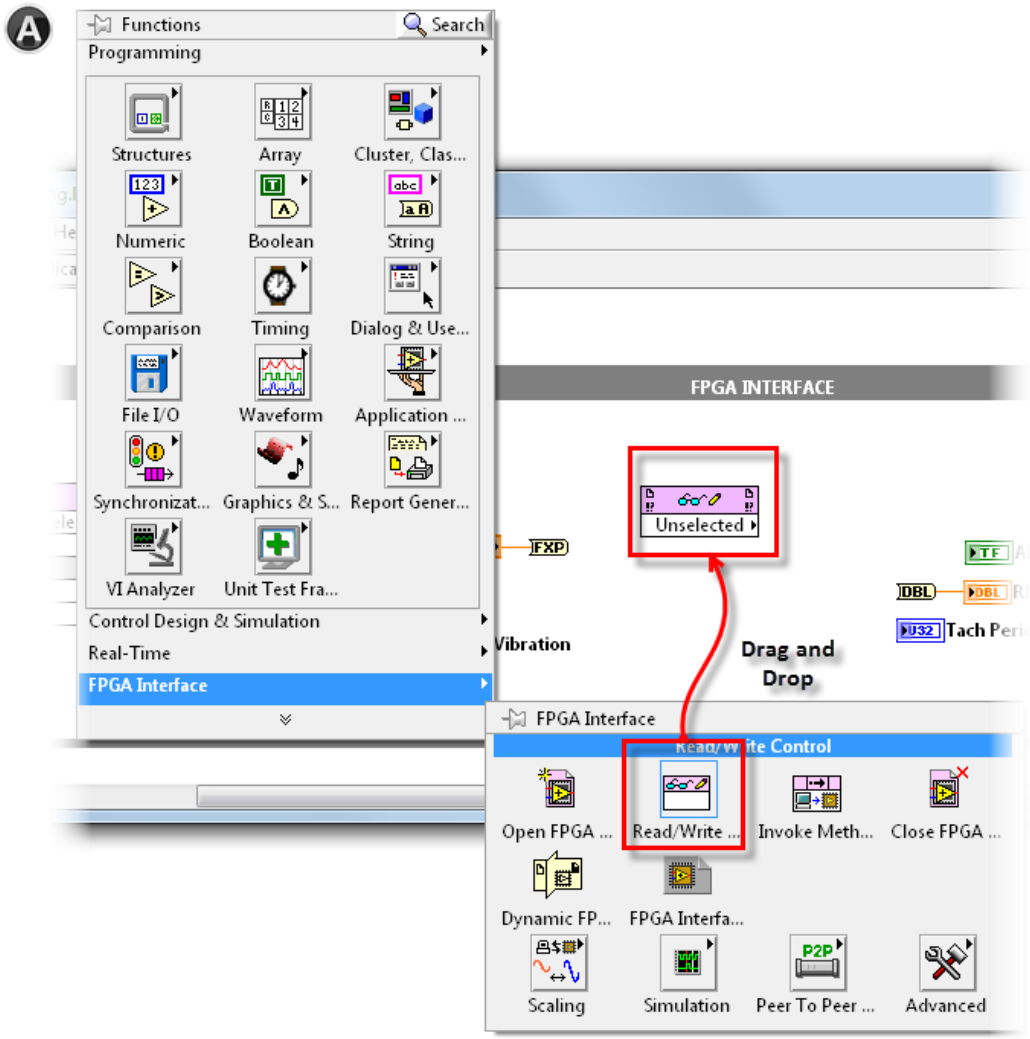
- ***Configure the Invoke Method***
  Continue the ***Reference*** (aqua) and ***Error*** (yellow) wires coming from the ***Open FPGA VI Reference function*** and connect them to the ***Invoke Method node [1]*** as shown in ***Figure B***. This populates the node with the information from the FPGA portion of the design described in FPGA Main.vi.

  Click on the ***Invoke Method node*** and navigate to ***Accelerometer X-Axis DMA FIFO » Read [2]***. This gives you access to the data included in this FIFO.

- ***Show the Data of the Accelerometer X-Axis DMA FIFO on a Graph***
  Wire the numeric constant with the value ***7000*** to the Number of Elements terminal. Wire the ***Data output terminal*** to the ***Turbine Vibration graph*** as shown in ***Figure C***. With this configuration, you get 7000 points of data out of the FIFO to display them on a graph.

14. Map the controls and indicators of the FPGA Main VI to the user interface.

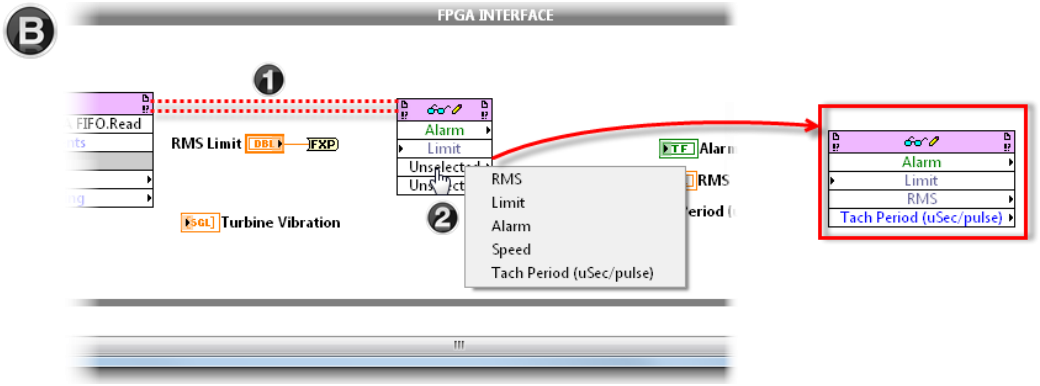
Place a Read/Write Control node to access the controls and indicators of FPGA Main.vi. Wire the diagram as shown in Figure C.

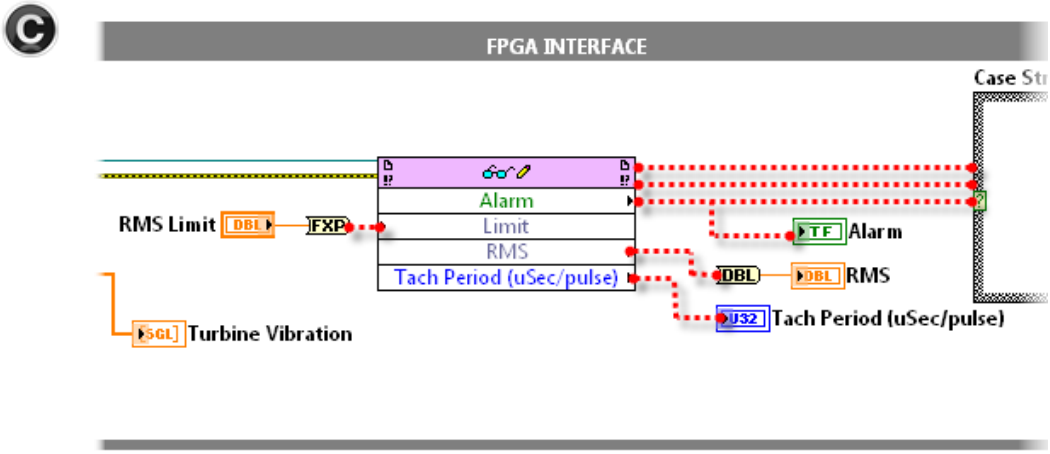


DETAILED INSTRUCTIONS

- ***Place a Read/Write Control Node***
  Right-click on the ***block diagram*** to open the ***Functions palette***. Navigate to ***FPGA Interface » Read/Write Control***. Drag and drop the ***Read/Write Control node*** onto the block diagram as shown in ***Figure A.***

  Read/Write Control nodes give you access to the controls and indicators on the front panel of the ***FPGA Main.vi***.





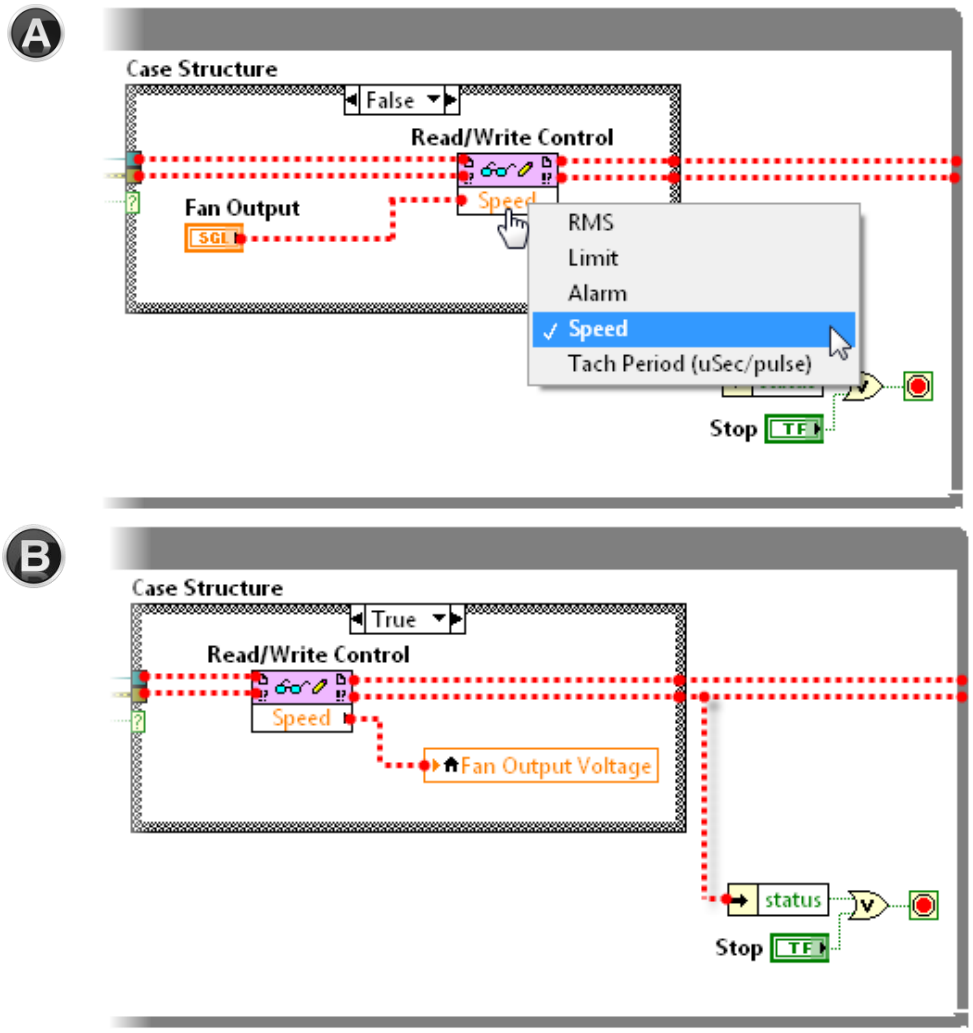
- ***Configure the Read/Write Control Node***
  Continue the ***Reference*** (aqua) and ***Error*** (yellow) wires coming from the ***Invoke Method node [1]*** and connect them to the ***Read/Write Control node*** as shown in ***Figure B***. This populates the node with the information on the controls and indicators of FPGA Main.vi.

  Expand the node to make four slots available ***[2]***. Click on each slot and select the appropriate control or indicator as shown in ***Figure B***.

- ***Wire the Read/Write Control Node to the Controls and Indicators on the User Interface***
  Wire each of the slots of the ***Read/Write Control node*** to its corresponding controls and indicators on the user interface as shown in ***Figure C***.

15. Complete the speed limit control algorithm to interact with the user interface.

Complete the Case structure with Read/Write Control nodes to write to and read from the Fan Output Voltage control on the user interface.

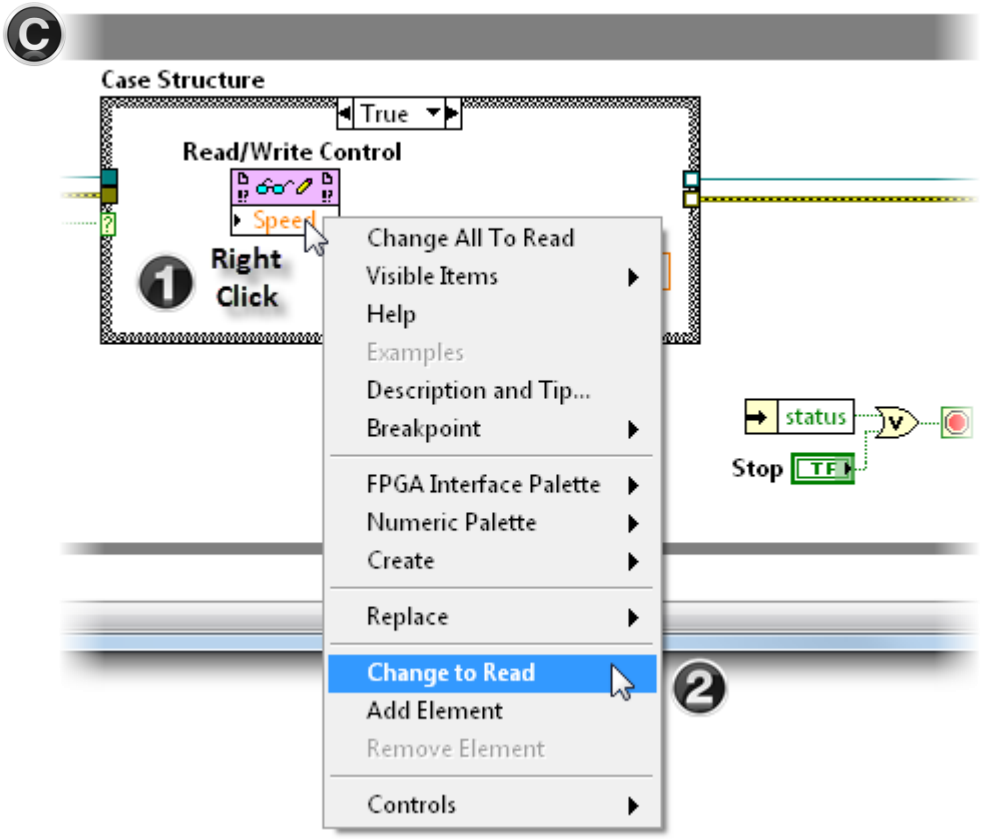


DETAILED INSTRUCTIONS

- ***Complete the Case Structure***
  The Case structure completes the logic of the ***Speed Limit Control Loop*** on the FPGA to gradually reduce the speed of the fan in case of an alarm.

- **False Case**
  If there is no alarm, operate the fan in manual mode by sending the value of the ***Fan Output Voltage control*** to the FPGA. To do this, add a ***Read/Write Control node*** as you did in point A of the previous step and continue the ***Reference*** and ***Error*** wires through it. Configure it to show the ***Speed terminal*** and wire the Fan Output Voltage terminal to it as shown in ***Figure A***.

- **True Case**
  If there is an alarm, operate the fan in auto mode to gradually reduce its speed. To do this, add a ***Read/Write Control node*** showing the ***Speed terminal.***



NOTE: To change the ***Access Method*** of the terminal, right-click on the Read/Write Control node ***[1]*** and select ***Change to Read [2]*** as shown in ***Figure C***.

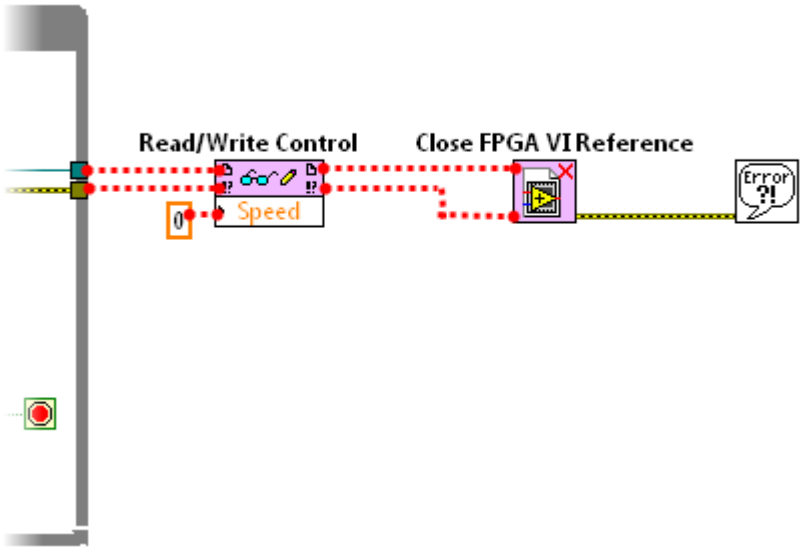
Wire the ***Read/Write Control node*** to the local variable of the ***Fan Output Voltage control*** as shown in ***Figure B***.

For both cases of the Case structure, continue the ***Reference*** and ***Error*** wires through the Case structure and out to the While Loop.

Additionally, continue the error wire to the ***status cluster***. This will stop the real-time VI in case of error.

16. Provide a shutdown state for the fan when stopping the application.

Add a Read/Write Control node outside the While Loop and wire it as shown in the image below to provide a shutdown state for the fan when stopping the application.

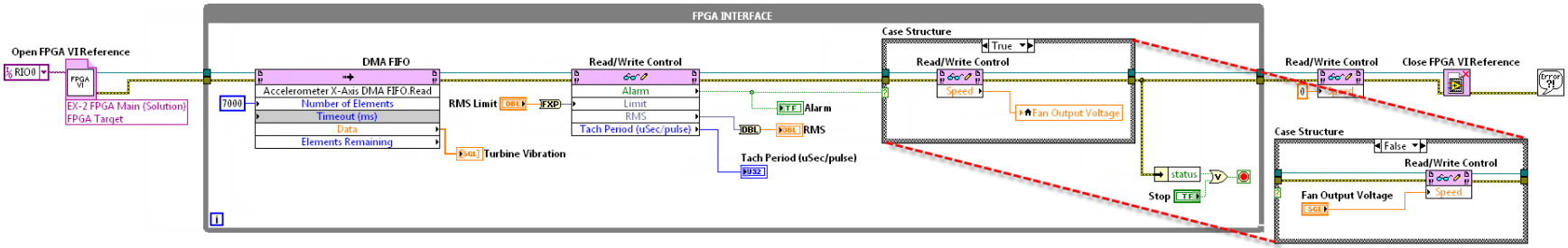


DETAILED INSTRUCTIONS

- ***Provide a Safe State for the Fan***
  Add a ***Read/Write Control node*** outside the While Loop and configure it to show the ***Speed terminal***. Wire the ***constant with value 0*** to this terminal and continue the ***Reference*** and ***Error*** wires all the way to the ***Close FPGA VI Reference*** function.

  This ensures that the fan halts when the application stops executing.

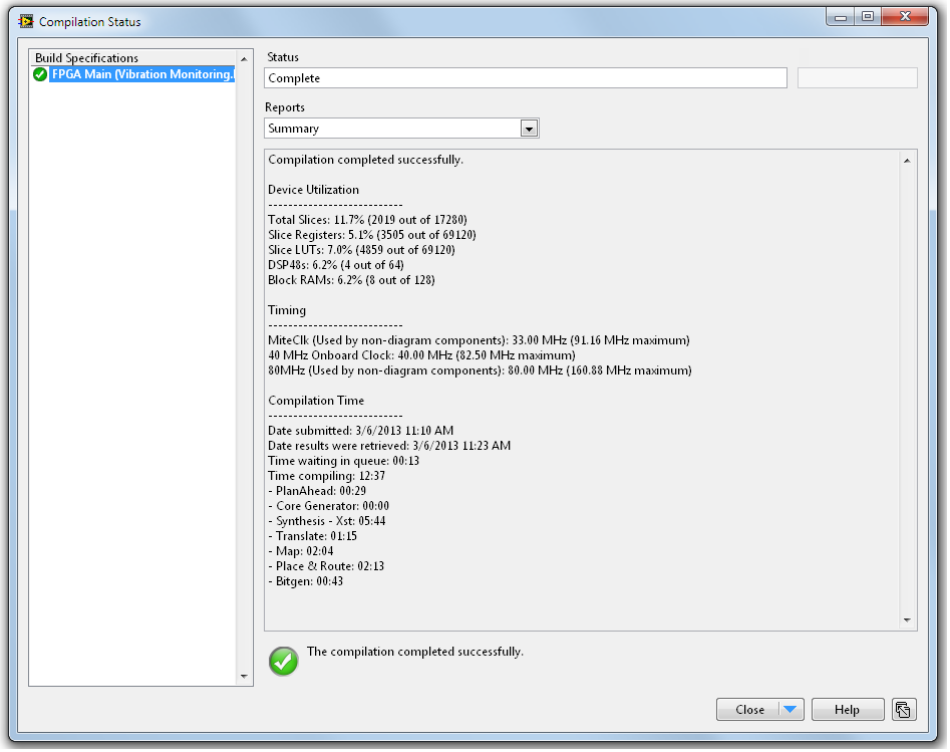
17. The completed block diagram for Real-Time Main.vi should look like the image below.



Part E—RUN THE APPLICATION

18. Run Real-Time Main.vi.

At this point, the compilation of FPGA Main.vi should be complete. Run Real-Time Main.vi and interact with the RMS limit and Fan Output Voltage controls on the user interface to verify the correct behavior of the application.



DETAILED INSTRUCTIONS

- ***Compilation Process Complete***
  Verify that the compilation process has completed successfully. The ***Compilation Status window*** shows a summary of the performance and resource utilization of the FPGA as shown in ***Figure A***.

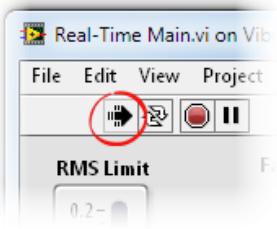
  NOTE: If you closed the ***Compilation Status window***, it can be re-opened by right clicking on the ***FPGA Main specification*** and selecting ***Display Compilation Result***.

  Device Utilization indicates the percentage of FPGA elements that the FPGA application uses.

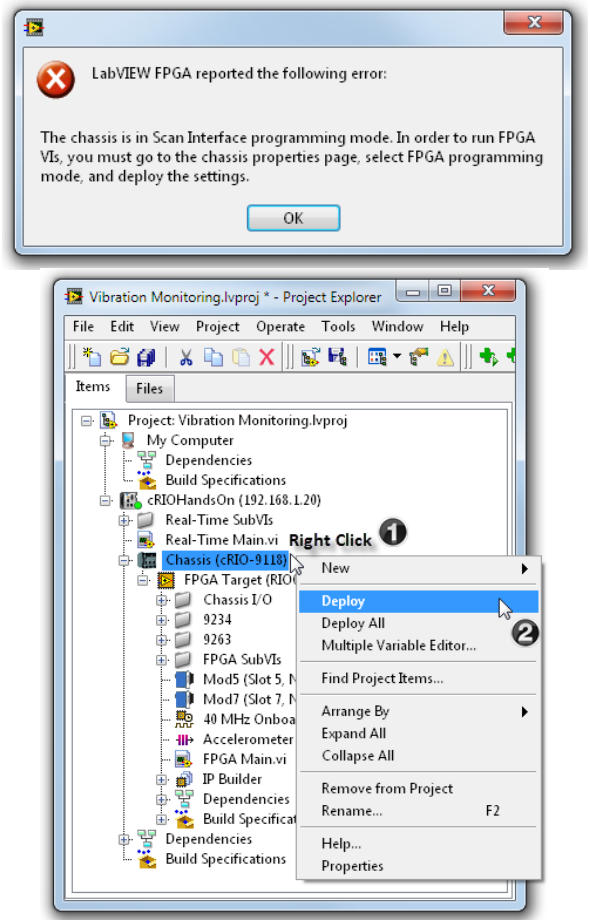
  Timing is a summary of the FPGA clocks, as estimated during the mapping of the FPGA VI.

  Compilation time depends on the size of the VI, processor speed, and amount of

B



C



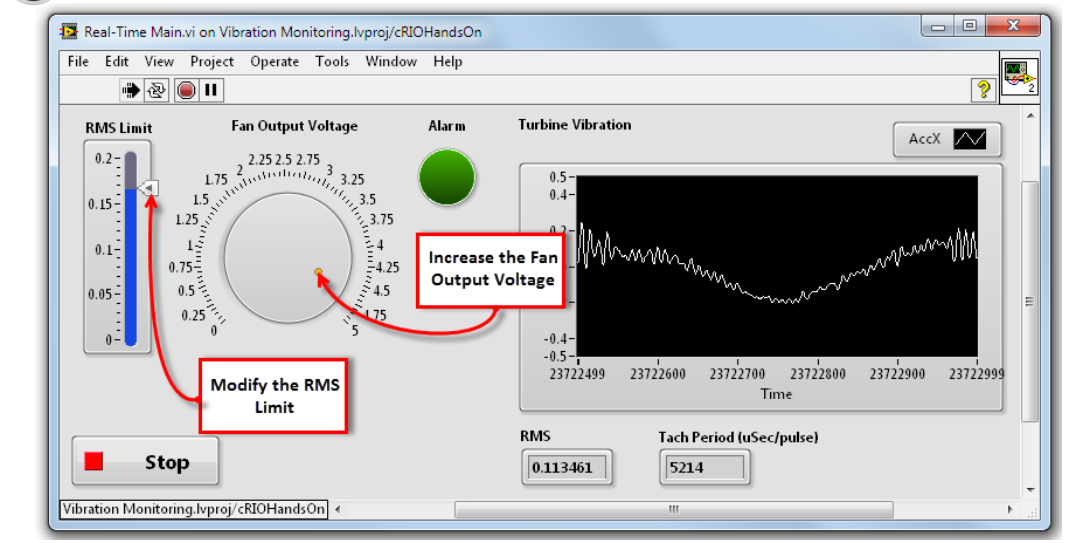
memory in the computer on which you are compiling.

- ***Run Real-Time Main.vi***
  Switch back to the front panel of Real-Time Main.vi **<Ctrl-E>** and run the program by clicking on the **arrow button** at the top of the front panel, as shown in ***Figure B***. Save the VI when prompted to do so.

  NOTE: A **warning window** may appear indicating that the chassis is configured in ***Scan Mode (Exercise 1)***.

  Close the warning window, go to the project tree, right-click on the chassis, and select ***Deploy*** as shown in ***Figure C***.

  Click on the ***Apply*** button on any subsequent warnings during this process. This changes the configuration of the chassis to FPGA mode.



- ***Interact With the Front Panel of Real-Time Main.vi***
  On the front panel, set the desired ***RMS Limit*** and manually increase the speed of the fan by modifying the value of the ***Fan Output Voltage control*** as shown in ***Figure D***. The fan then increases its speed and the vibration it produces, which triggers the alarm. When an alarm goes off, the Fan Output Voltage control automatically reduces its value to control the speed of the fan.

19. Stop the application.

Stop the execution of the code by pressing the Stop button on the front panel of Real-Time Main.vi.

Part F—CHALLENGE

20. Modify the Alarm LED indicator and the Turbine Vibration graph.

Add code to make the ***Alarm LED indicator*** blink in red every time an alarm goes off. Additionally, modify the ***Turbine Vibration graph*** to show both the vibration signal and the RMS value.

Close the project when finished.

<END OF EXERCISE 2—VIBRATION MONITORING>

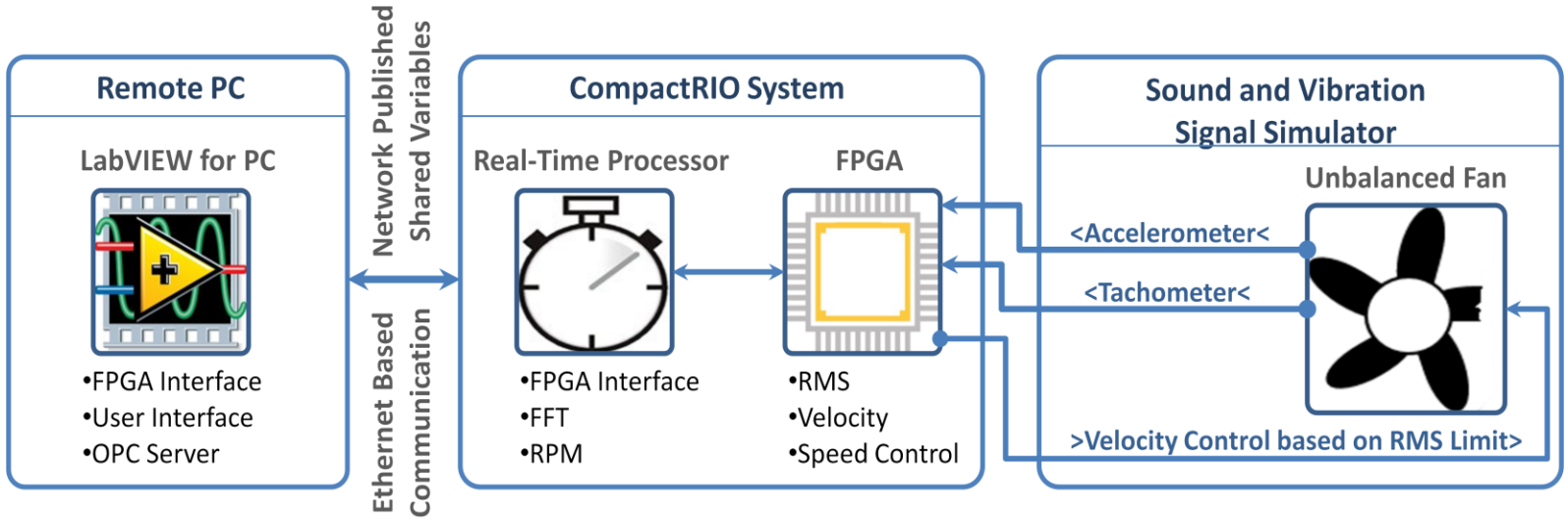
EXERCISE 3: HUMAN MACHINE INTERFACE (HMI) AND COMMUNICATIONS

Goals

- Communicate the ***CompactRIO system*** with a ***remote PC*** using ***variables.***
- Create a ***remote HMI*** to visualize the data of the ***vibration monitoring system.***
- Create an ***OPC client*** to access the data from the processes of other devices.

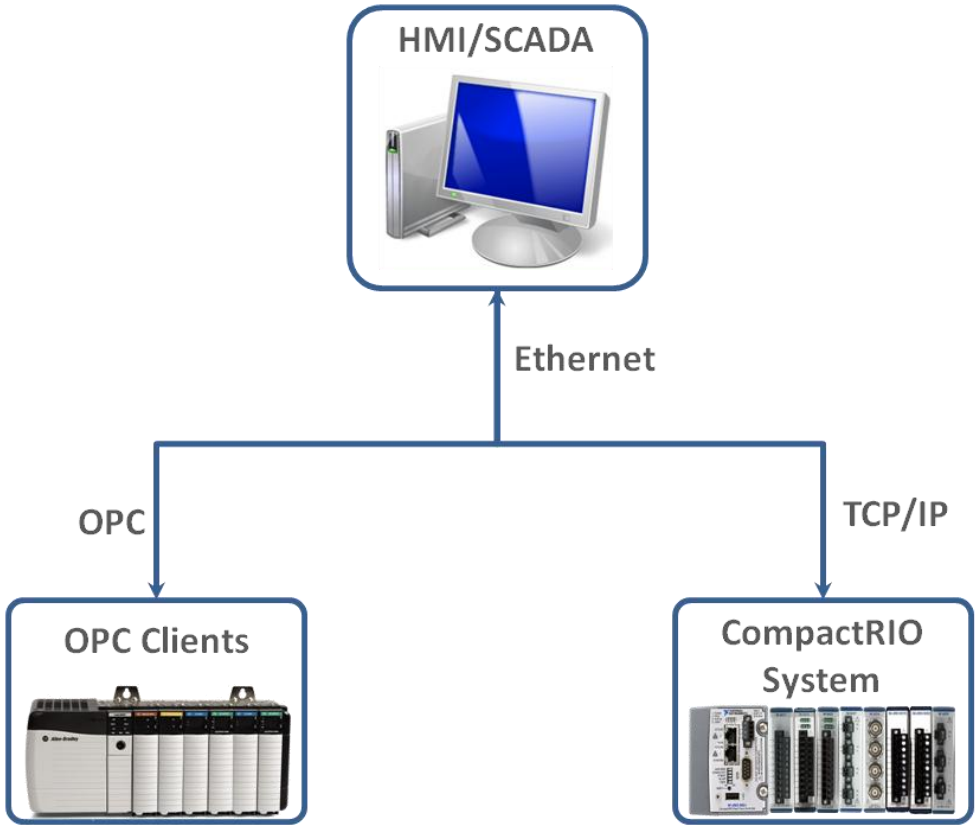
Part A—APPLICATION DESCRIPTION

Remote HMI for the Vibration Monitoring System



Interfacing With Remote Systems

In this exercise, create a remote user interface for the ***vibration monitoring application*** running on the ***CompactRIO system***. Transfer the relevant values of this process via ***Ethernet communication*** using ***network-published shared variables*** to a ***remote PC***. Additionally, use ***LabVIEW*** as an ***OPC server*** to expose the variables of the process to other devices.



Network-Published Shared Variables

This is a method to make data available over an ***Ethernet network***. In this exercise, use this method to transfer data back and forth from the ***CompactRIO system*** to the ***remote PC***.

NI OPC Servers

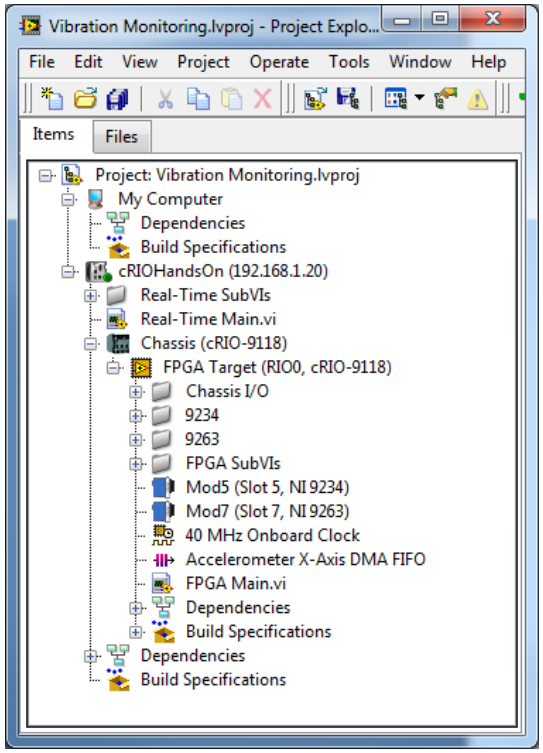
These enable ***LabVIEW*** software to communicate with many different ***programmable logic controllers (PLCs)*** and ***third-party devices*** through the ***OPC client***.

Part B—CODE IMPLEMENTATION

1. Open the Vibration Monitoring.lvproj LabVIEW project.

In this exercise, we continue building the application from Exercise 2.

C:\CompactRIO HO\Exercise 3\Vibration Monitoring.lvproj



DETAILED INSTRUCTIONS

This exercise is based on the code developed in ***Exercise 2***.

- Close any open LabVIEW projects.

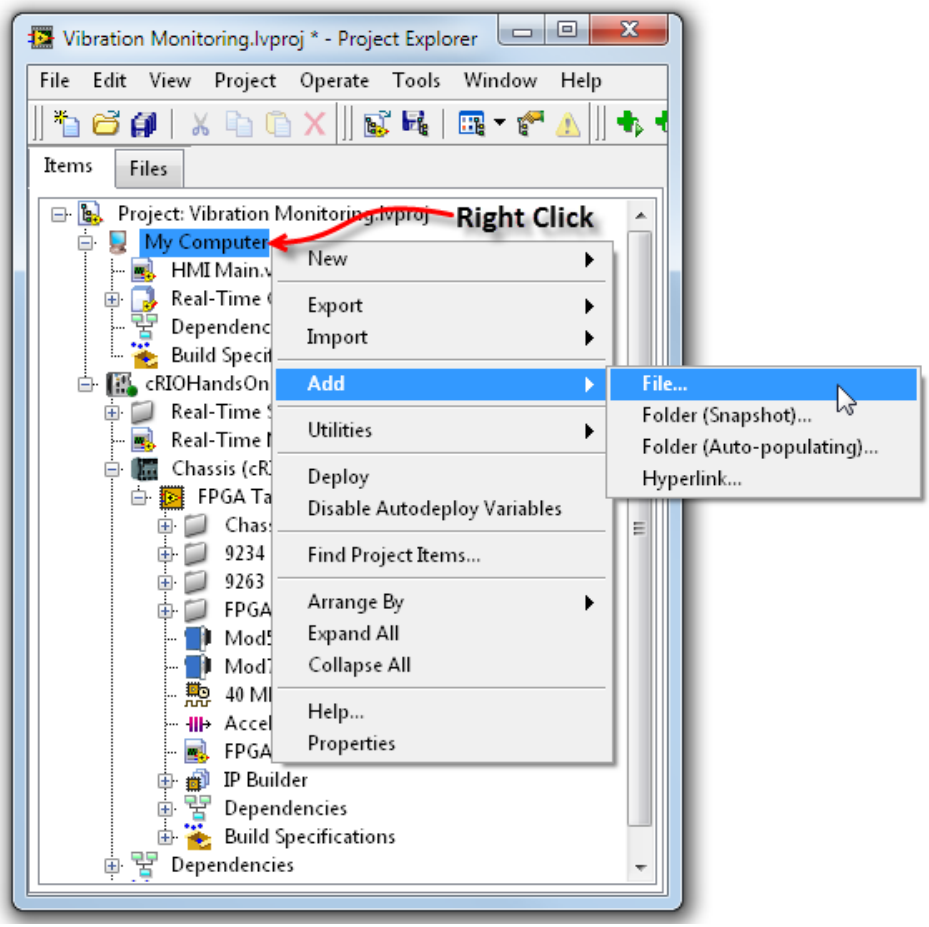
- ***Open an Existing LabVIEW Project***
  Click on the ***Open Existing*** button on the right side of the ***LabVIEW*** main screen and open the ***LabVIEW Project*** file named ***Vibration Monitoring.lvproj*** located at:
  C:\CompactRIO HO\Exercise 3\Vibration Monitoring.lvproj

Use this code to implement the communication with the ***remote HMI***.

2. Add the required files to the project to implement the remote HMI.

Use prebuilt files to implement the user interface as well as the communication mechanism with the CompactRIO system. Add HMI Main.vi and the Real-Time Communications.lvlib files to the My Computer hierarchy in the project.

C:\CompactRIO HO\Exercise 3\HMI-Communications



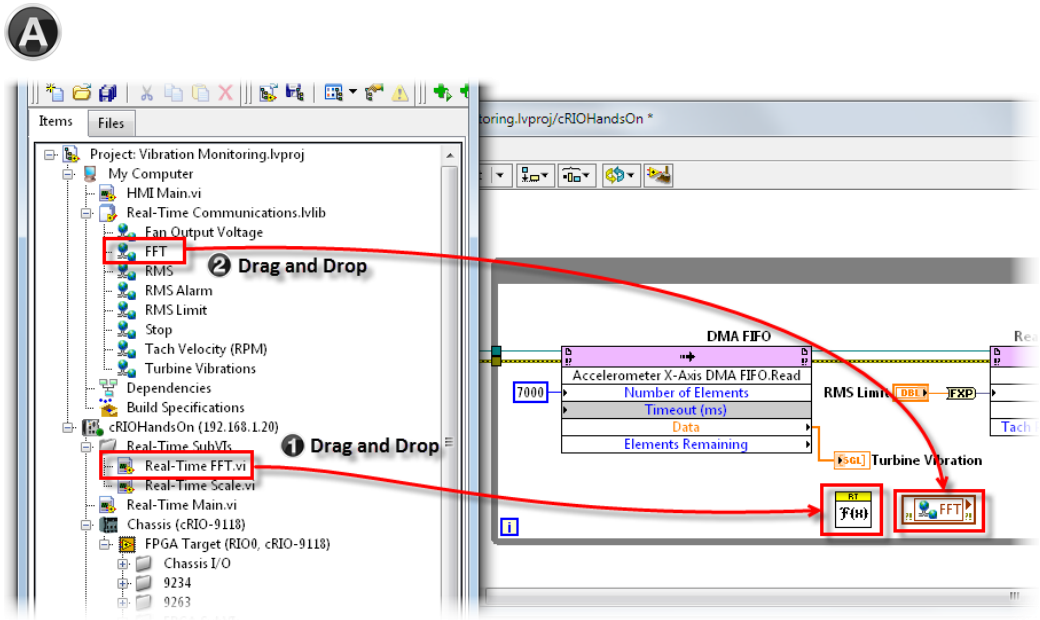
DETAILED INSTRUCTIONS

- ***Add the HMI Main.vi***
  Right-click on the ***My Computer*** hierarchy of the project and select ***Add»File…*** Navigate to the path shown in the figure on the left and select ***HMI Main.vi***. This file already contains a user interface that you will complete to be used remotely with the ***CompactRIO system***.

- ***Add the Real-Time Communications.lvlib***
  Right-click on the ***My Computer*** hierarchy of the project and select ***Add»File…*** Navigate to the path shown in the figure on the left and select the ***Real-Time Communications.lvlib***. This file is a library of ***network-published shared variables*** for helping us communicate data back and forth from the ***CompactRIO system*** to the ***remote PC***.

3. Modify Real-Time Main.vi to communicate data to the user interface.

Modify Real-Time Main.vi to add the FFT analysis of the vibration signal and send it back to the user interface. Add the file named Real-Time FFT.vi and the FFT network-published shared variable to the block diagram. Wire the diagram as shown in the figure below.

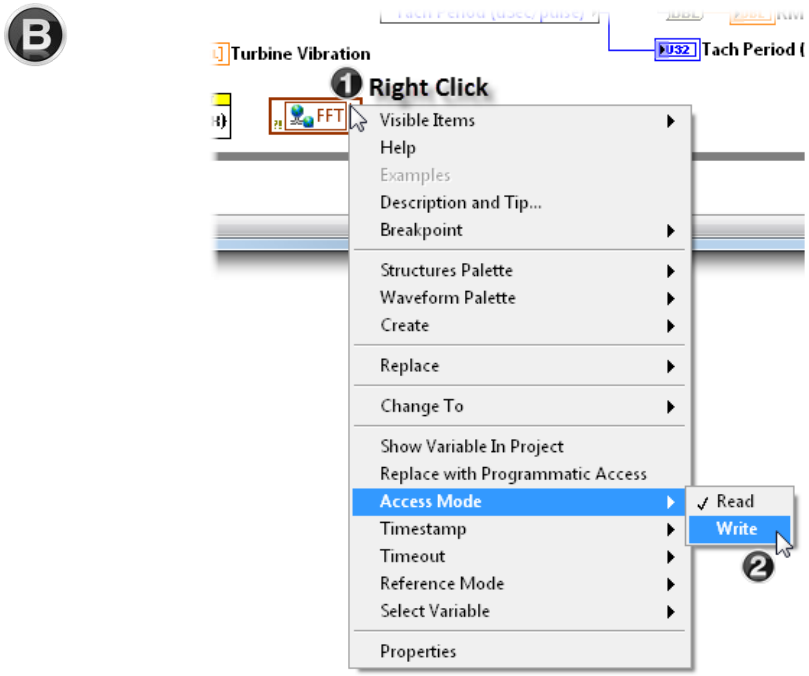


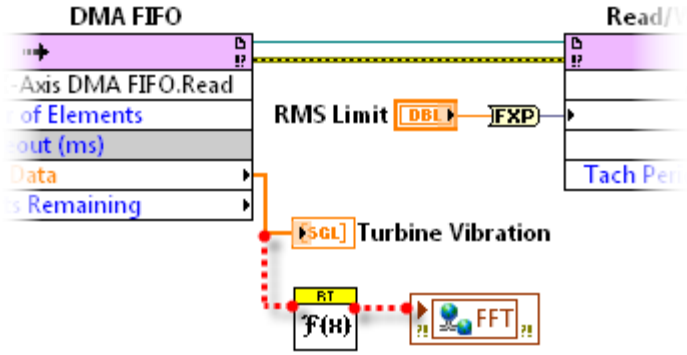
DETAILED INSTRUCTIONS

- ***Open Real-Time Main.vi***
  Double-click on the file named ***Real-Time Main.vi*** located under the ***cRIO HandsOn*** hierarchy. Press ***<Ctrl-E>*** or click on ***Window » Show Block Diagram*** to toggle between the block diagram and the front panel.

- ***Add Real-Time FFT.vi to the Block Diagram***
  Drag and drop onto the ***block diagram*** the file ***Real-Time FFT.vi*** located in the folder labeled ***Real-Time SubVIs*** under the ***cRIO HandsOn*** hierarchy as shown in ***Figure A***.

- ***Add the FFT Network-Published Shared Variable to the Block Diagram***
  Expand the ***Real-Time Communications*** variable library under the ***My Computer*** hierarchy and drag and drop the ***FFT variable*** onto the block diagram as shown in ***Figure A***.





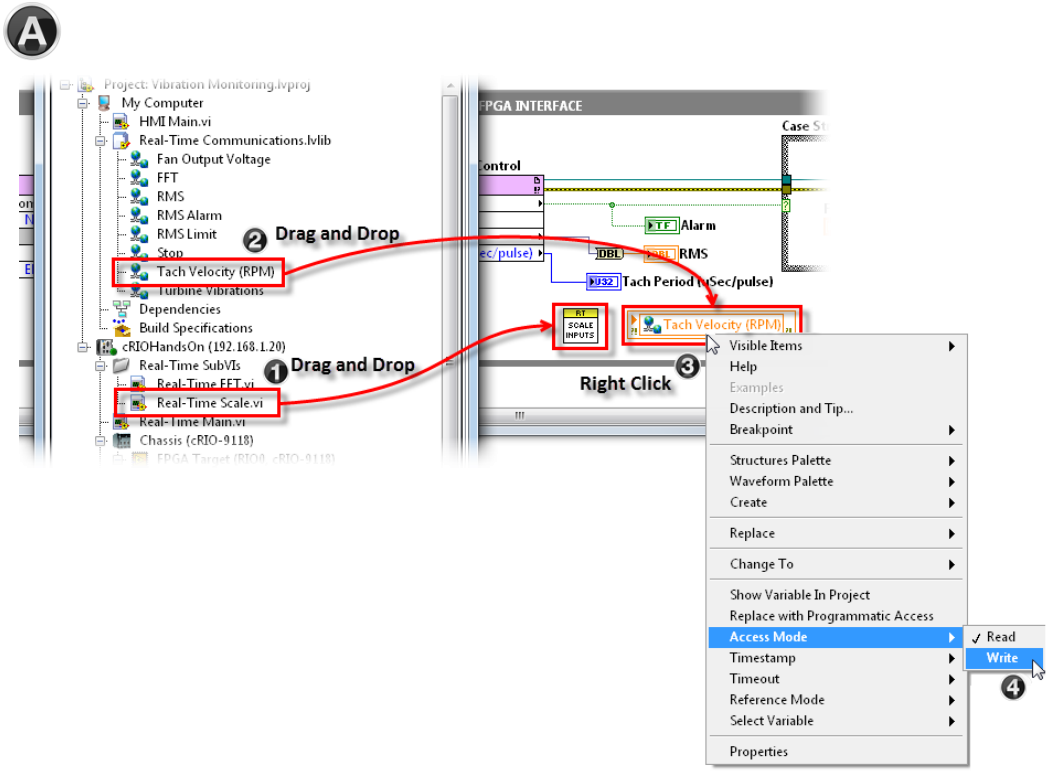
- ***Change the Access Mode of the FFT Network-Published Shared Variable***
  In this part of the code, write to the variable to send the result of the FFT to the user interface. To change the variable to ***Write mode,*** right-click on it ***[1]*** and select ***Access Mode»Write [2]*** as shown in ***Figure B***.

- ***Wire the Rest of the Diagram as Indicated in Figure C***
  Create a new branch from the ***Turbine Vibration indicator*** and wire it to the input of ***Real-Time FFT.vi***. Wire the output of this function to the ***FFT Network-Published Shared Variable***.

  This allows you to transfer the results of the FFT analysis over the vibration signal to show them in the remote user interface.

4. Add the RPM calculation of the tachometer signal to the block diagram.

Use another prebuilt function named Real-Time Scale.vi to calculate the RPM value of the tachometer. Delete the Tach Period (uSec/pulse) indicator and add the Tach Velocity (RPM) network-published shared variable to transfer the RPM value to the remote user interface.

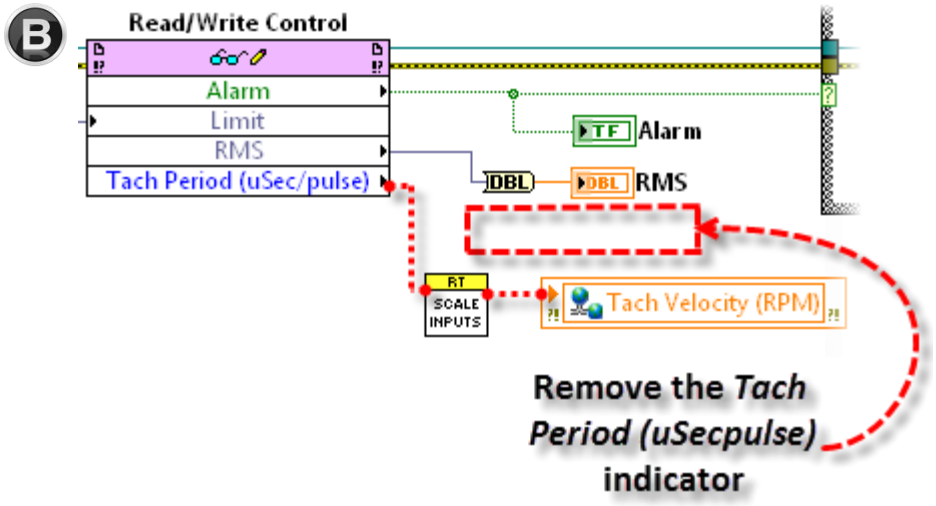


DETAILED INSTRUCTIONS

- ***Add Real-Time Scale.vi to the Block Diagram***
  Drag and drop onto the ***block diagram*** the file ***Real-Time Scale.vi [1]*** located in the folder labeled ***Real-Time SubVIs*** under the ***cRIO HandsOn*** hierarchy as shown in ***Figure A***.

- ***Add the Tach Velocity (RPM) Network-Published Shared Variable to the Block Diagram***
  Expand the ***Real-Time Communications*** variable library under the ***My Computer*** hierarchy and drag and drop the ***Tach Velocity (RPM) variable [2]*** onto the block diagram as shown in ***Figure A***.

  Change the ***Access Mode*** of this variable to ***Write [3,4]*** as shown in ***Figure A***.



- ***Delete the Tach Period (uSec/pulse)***
  Delete this indicator to make room for Real-Time Scale.vi and the Tach Velocity (RPM) variable as shown in ***Figure B***. Click on the indicator and press the Delete key. Remove ***the broken wires*** by selecting ***<Ctrl-B>.***

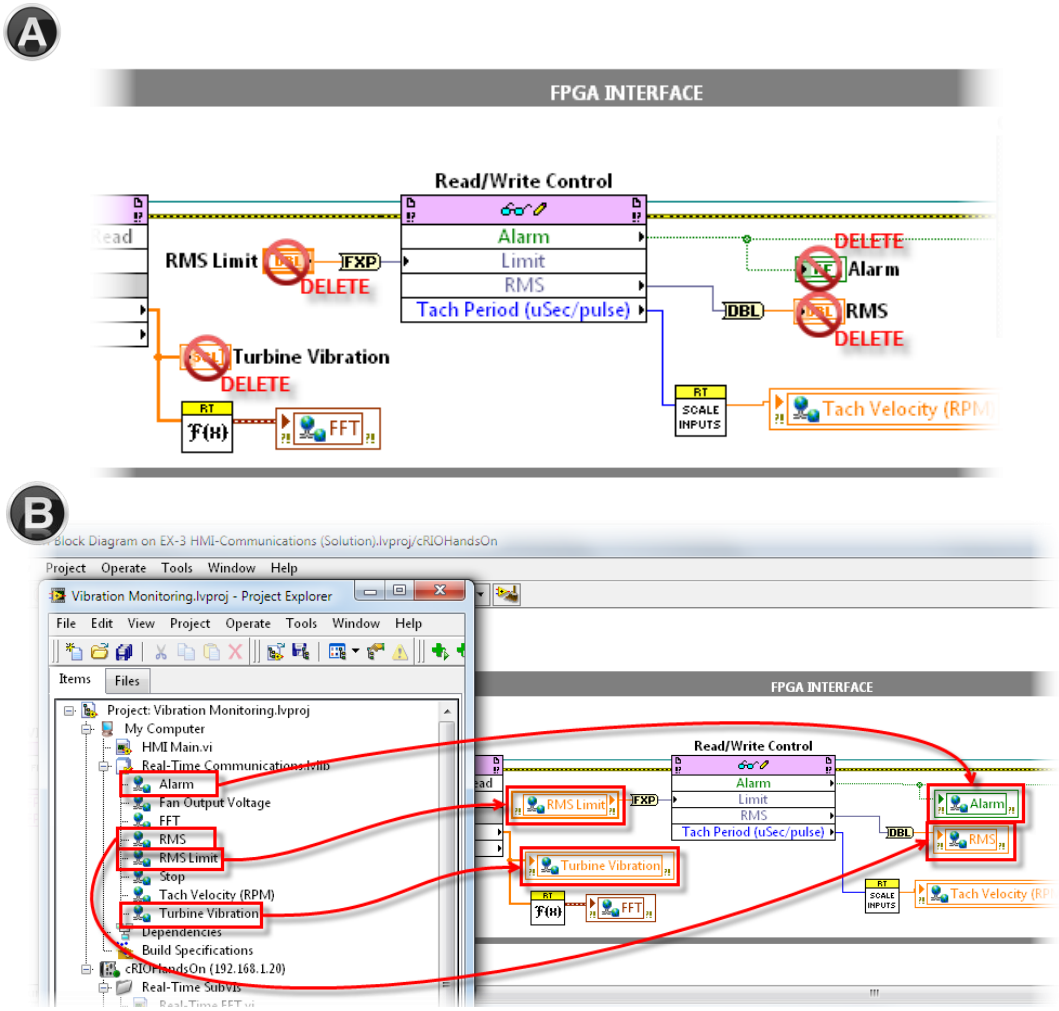
  This helps you maintain a clean, readable block diagram since you are moving the user interface to the remote PC.

- ***Wire the Rest of the Diagram as Indicated in Figure B.***
  Wire the ***Tach Period (uSec/pulse)*** terminal of the ***Read/Write Control node*** to the input terminal of ***Real-Time Scale.vi***. Wire the output of this terminal to the ***Tach Velocity (RPM) variable***.

  By doing this, you can transfer the scaled value of the tachometer signal to the remote user interface.

5. Use the rest of the variables to transfer the application data to the user interface.

Substitute the existing controls and indicators in the block diagram for the corresponding network-published shared variables as shown in the figure below.



DETAILED INSTRUCTIONS

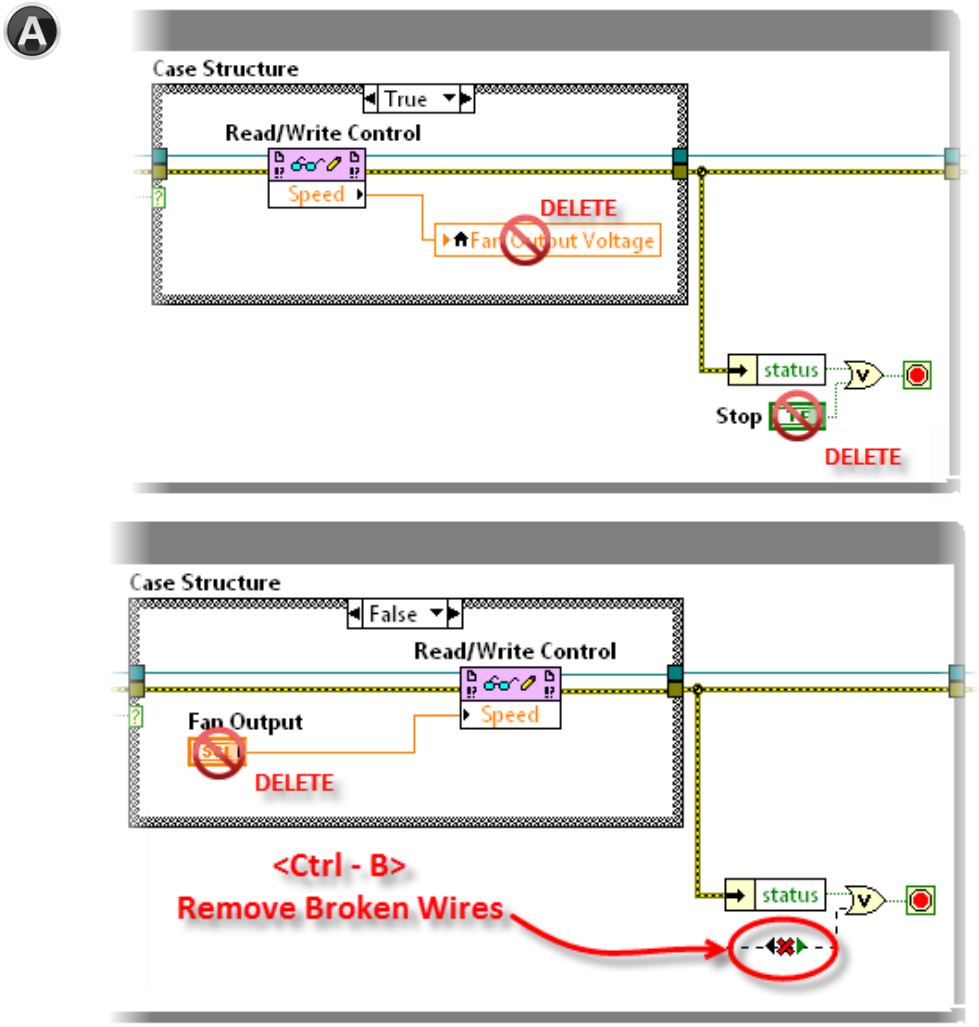
To communicate the relevant data of the process to the user interface and have a clean, readable ***block diagram,*** substitute the controls and indicators for their corresponding ***network-published shared variables***.

- ***Delete the Controls and Indicators shown in Figure A.***
  Delete the following controls and indicators: ***RMS Limit***, ***Turbine Vibration***, ***RMS***, and ***Alarm.*** Remove ***the broken wires*** by pressing ***<Ctrl-B>.***

- ***Add the Corresponding Network-Published Shared Variables***
  Add the variables shown in ***Figure B*** to the block diagram and rewire them accordingly. For this process, you need to configure:
  Turbine Vibration, ***Alarm***, and ***RMS*** variables in ***Write Mode***.
  RMS Limit variable in ***Read Mode***.

6. Complete the speed limit control logic to transfer data to the remote user interface.

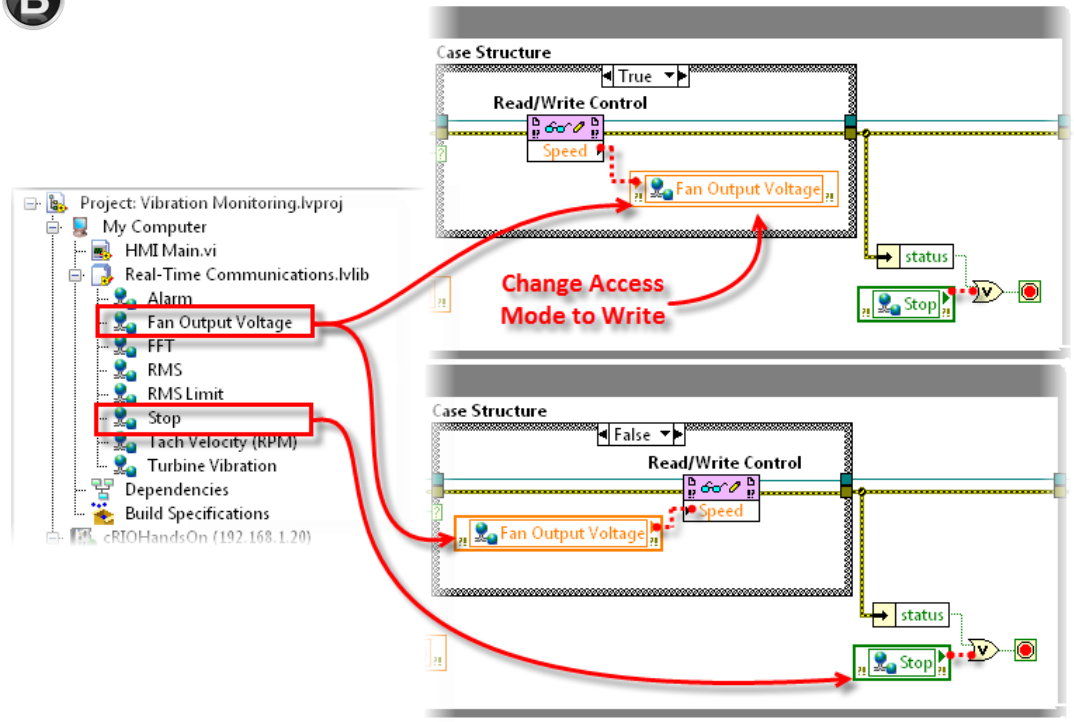
Substitute the Fan Output Voltage control and its local variable inside the Case structure as well as the Stop button with variables so they can operate from the user interface. Delete the mentioned controls and substitute them for their respective variables as shown below.



DETAILED INSTRUCTIONS

In this section, finish removing the controls and indicators in Real-Time Main.vi to convert it into a stand-alone application. As part of this, remove the ***Fan Output Voltage*** and ***Stop Control*** from the block diagram.

- ***Delete the Controls and Indicators shown in Figure A.***
  Delete the following controls and indicators: ***Fan Output Voltage, Fan Output Voltage (Local Variable),*** and ***Stop.*** Remove ***the broken wires*** by pressing ***<Ctrl-B>.***



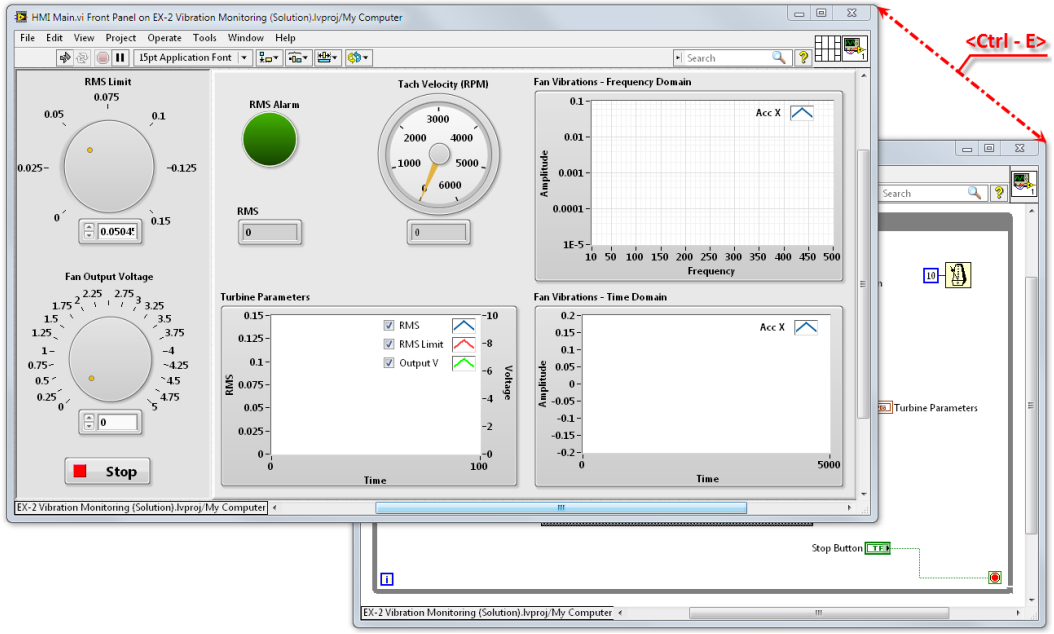
- ***Add the Corresponding Network-Published Shared Variables***
  Add the variables shown in ***Figure B*** to the block diagram and rewire them accordingly. For this process, configure: ***Fan Output Voltage variable*** in ***Write Mode*** for the True case, and in ***Read Mode*** for the False Case.
  Stop variable in ***Read Mode***.

  Save the ***Real-Time Main.vi*** by pressing ***<Ctrl-S>*** or click on ***File»Save***.

With this, ***Real-Time Main.vi*** is ready to receive and send data to the remote user interface. Network-published shared variables facilitate this process by wrapping the required TCP/IP protocol to perform this type of communication. Now you can complete the communication on the HMI side.

7. Open HMI Main.vi and show its block diagram.

Open HMI Main.vi and observe the controls and indicators contained in the front panel. This is going to be your remote user interface running on the development machine.



DETAILED INSTRUCTIONS

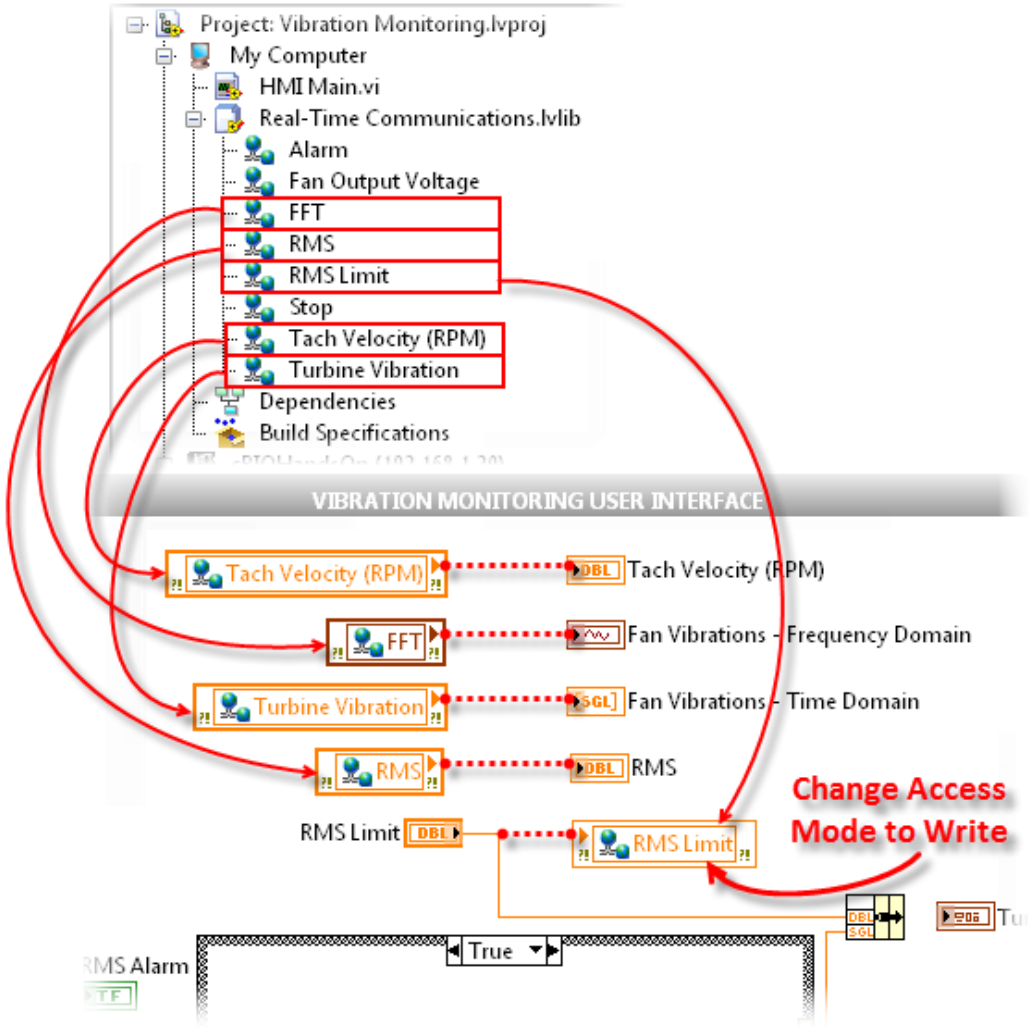
- ***Open HMI Main.vi***
  Double-click on the file named ***HMI Main.vi*** located under the ***My Computer*** hierarchy.

- ***Show the Block Diagram***
  Press ***<Ctrl-E>*** or click on ***Window » Show Block Diagram*** to toggle between the block diagram and the front panel.

This file contains the remote user interface for the ***vibration monitoring*** application. In the ***block diagram*** of this file, complete the code to communicate with the ***CompactRIO system***.

8. Complete the communication with the CompactRIO system.

Switch back to the block diagram of HMI Main.vi and add the appropriate variables to complete the communication with the CompactRIO system as shown in the figure below.



DETAILED INSTRUCTIONS

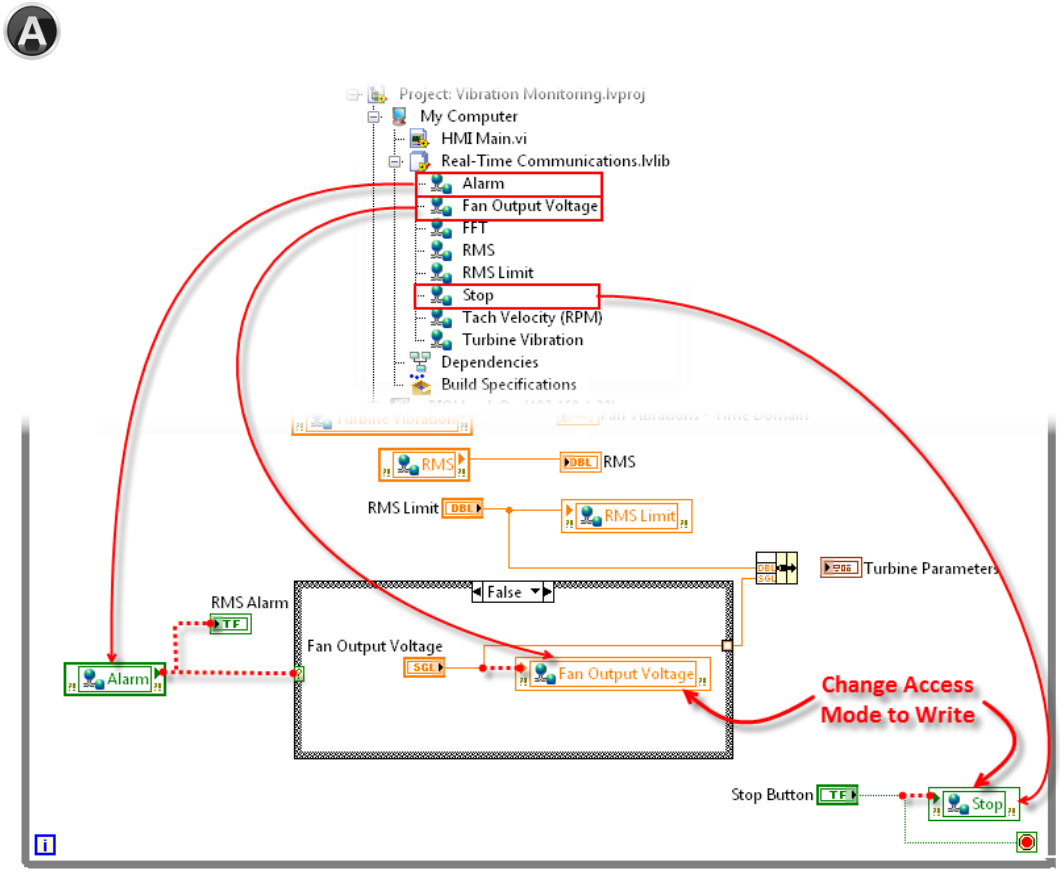
In this section, create the other part of the communication to interact with the process the CompactRIO system is controlling.

- ***Complete the Communication With the CompactRIO System***
  Add the variables shown in ***Figure A*** to the block diagram and wire them accordingly. For this process, configure the ***RMS Limit variable*** in ***Write Mode***.

  By doing this, you are directly mapping the data produced by the ***CompactRIO system*** to the user interface implemented on the ***remote PC***.

9. Complete the speed limit control logic on the user interface side.

Replicate the logic implemented on the real-time portion of the CompactRIO system to manipulate the speed of the fan from the user interface. Add the Fan Output Voltage, Alarm, and Stop variables to the block diagram and wire as shown in the figure below.

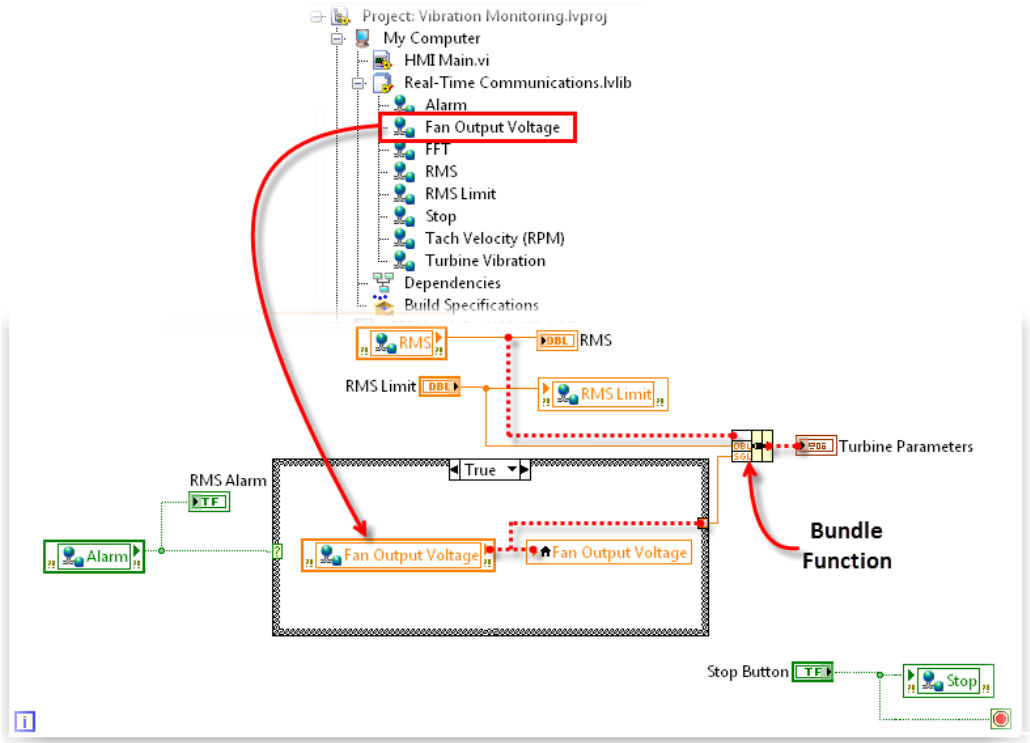


DETAILED INSTRUCTIONS

Add the required components to replicate the logic for the ***Speed Limit Control*** in the user interface.

- ***Add the Corresponding Network-Published Shared Variables***
  Add the Alarm and Stop variables to the block diagram and wire them as shown in ***Figure A***. For this process, configure the ***Stop variable*** in ***Write Mode***.

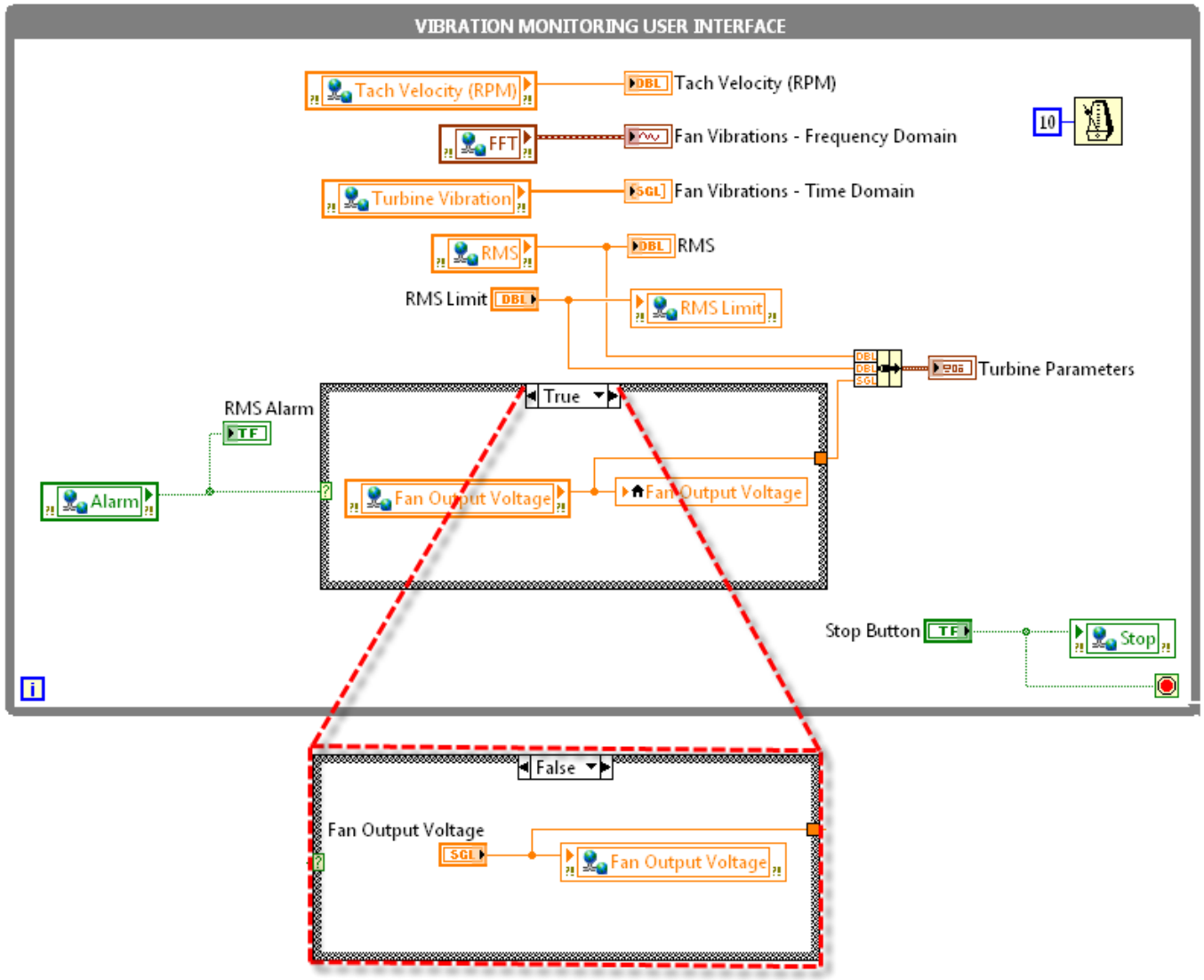
- ***False Case***
  Add the ***Fan Output Voltage variable*** and wire it as shown in ***Figure A***. For this process, configure the ***Fan Output Voltage variable*** in ***Write Mode***.



- ***True Case***
  Add another instance of the ***Fan Output Voltage variable*** and wire it as shown in ***Figure B***.

- ***Turbine Parameters Graph***
  Wire the ***RMS terminal*** to the first terminal of the ***Bundle Function***. Wire the output of this function to the Turbine Parameters graph. With this graph, you can visualize the ***RMS***, ***RMS Limit***, and ***Fan Output Voltage*** values on a single graph.

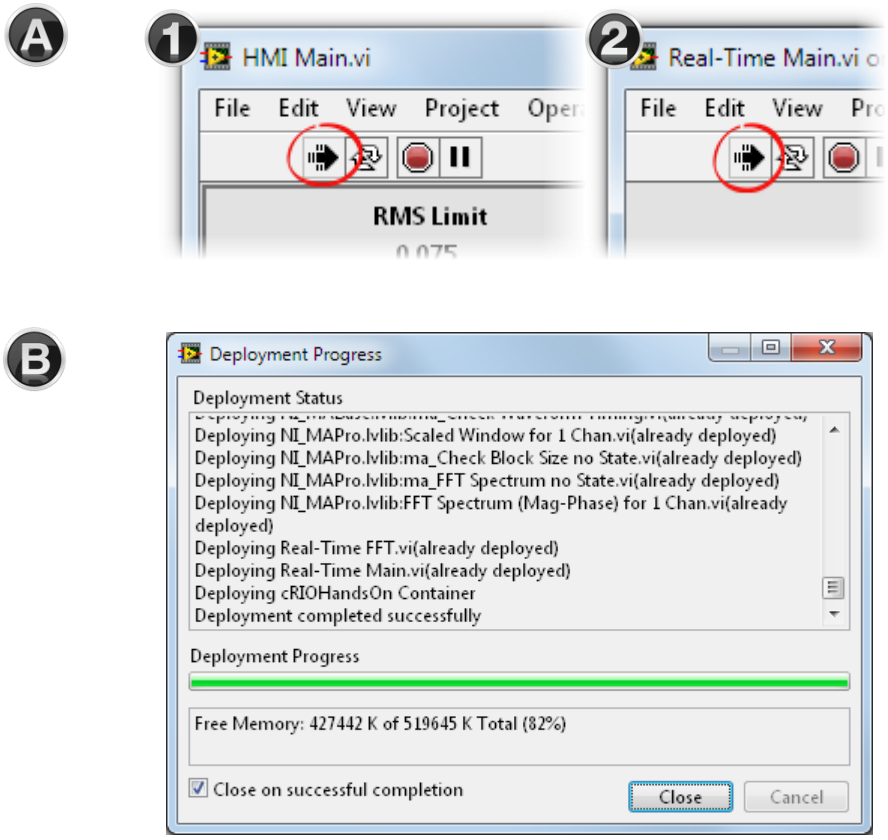
10. The complete block diagram for HMI Main.vi should look like the figure below.



Part C—RUN THE APPLICATION

11. Run HMI Main.vi.

Run HMI Main.vi to deploy the variables library and make the variables accessible through the network. Then run Real-Time Main.vi to deploy the application to the CompactRIO system. Interact with the RMS limit and Fan Output Voltage controls on the remote user interface to verify the correct behavior of the application.



DETAILED INSTRUCTIONS

- ***Run HMI Main.vi***
  Run the remote user interface first ***[1]*** by clicking on the ***arrow button*** at the top of the front panel as shown in ***Figure A***. Save the VI when prompted to do so. This process makes the variables library public over the network because the library is located under the ***My Computer*** hierarchy in the project.

- ***Run Real-Time Main.vi***
  Run ***Real-Time Main.vi*** by clicking on the ***arrow button [2]*** at the top of the front panel as shown in ***Figure A***. Save the VI when prompted to do so. By doing this, the real-time application is downloaded to the CompactRIO system and starts the communication via Ethernet with the remote user interface as shown in ***Figure B***.

- ***Interact With the Front Panel of HMI Main.vi***

  In the front panel, set the desired ***RMS Limit*** and manually increase the speed of the fan by modifying the value of the ***Fan Output Voltage control*** as shown in ***Figure C***. The fan increases its speed and the vibration it produces, which triggers the alarm. When an alarm goes off, the Fan Output Voltage control automatically changes its value to control the speed of the fan.

  Additionally, observe how the FFT graph and the RPM value change according to the Fan Output Voltage. These calculations are performed on the CompactRIO system and are distributed across the real-time processor and the FPGA.

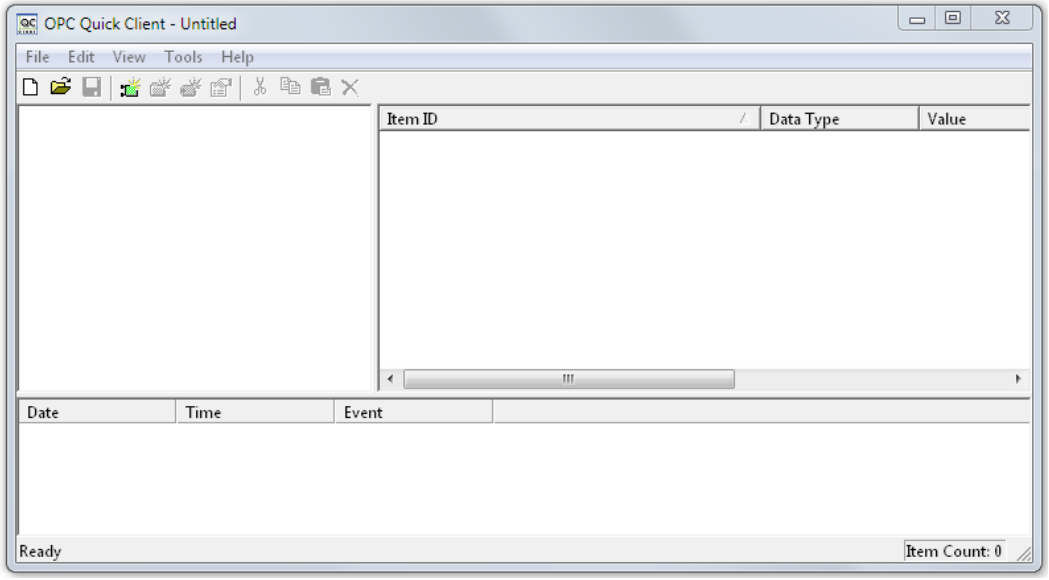
12. Stop the application.

Stop the execution of the code by pressing the Stop button on the front panel of ***HMI Main.vi***. Notice that the ***Real-Time Main.vi*** stops as well.

Part D (OPTIONAL)—OPC CLIENT

13. Create an OPC client.

At this moment, the variables published in the LabVIEW project are public over the network so other devices can access them. LabVIEW uses the Shared Variable Engine to make this possible. You also can use this engine as an OPC server through which third-party devices can access the variables of the process that the CompactRIO system is controlling. Open the ***OPC Quick Client*** by going to ***Start » All Programs » National Instruments » OPC Servers 2013 » OPC Quick Client.***



DETAILED INSTRUCTIONS

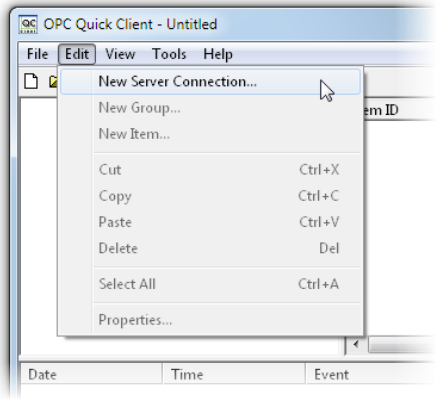
In this section, use LabVIEW as an OPC server and transfer the data of the vibration monitoring application to other devices.

- ***Open the OPC Quick Client***
  Simulate a third-party device requesting information from the vibration monitoring application through an OPC client. Open the ***OPC Quick Client*** by going to ***Start » All Programs » National Instruments » OPC Servers.***

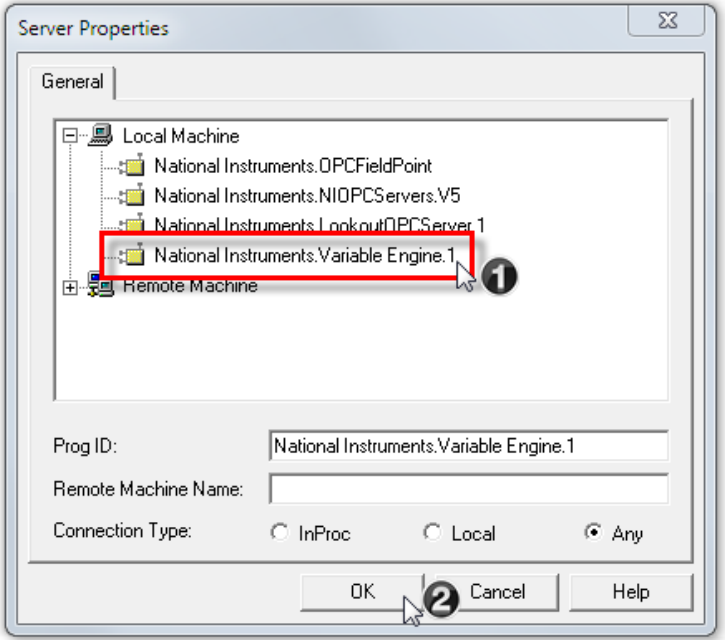
14. Create a new server connection to the LabVIEW Shared Variable Engine.

Configure a connection to the Shared Variable Engine to gain access to the published variables of the vibration monitoring application. Go to ***Edit » New Server Connection…*** and select ***National Instruments.Variable Engine.1.***

A



B



DETAILED INSTRUCTIONS

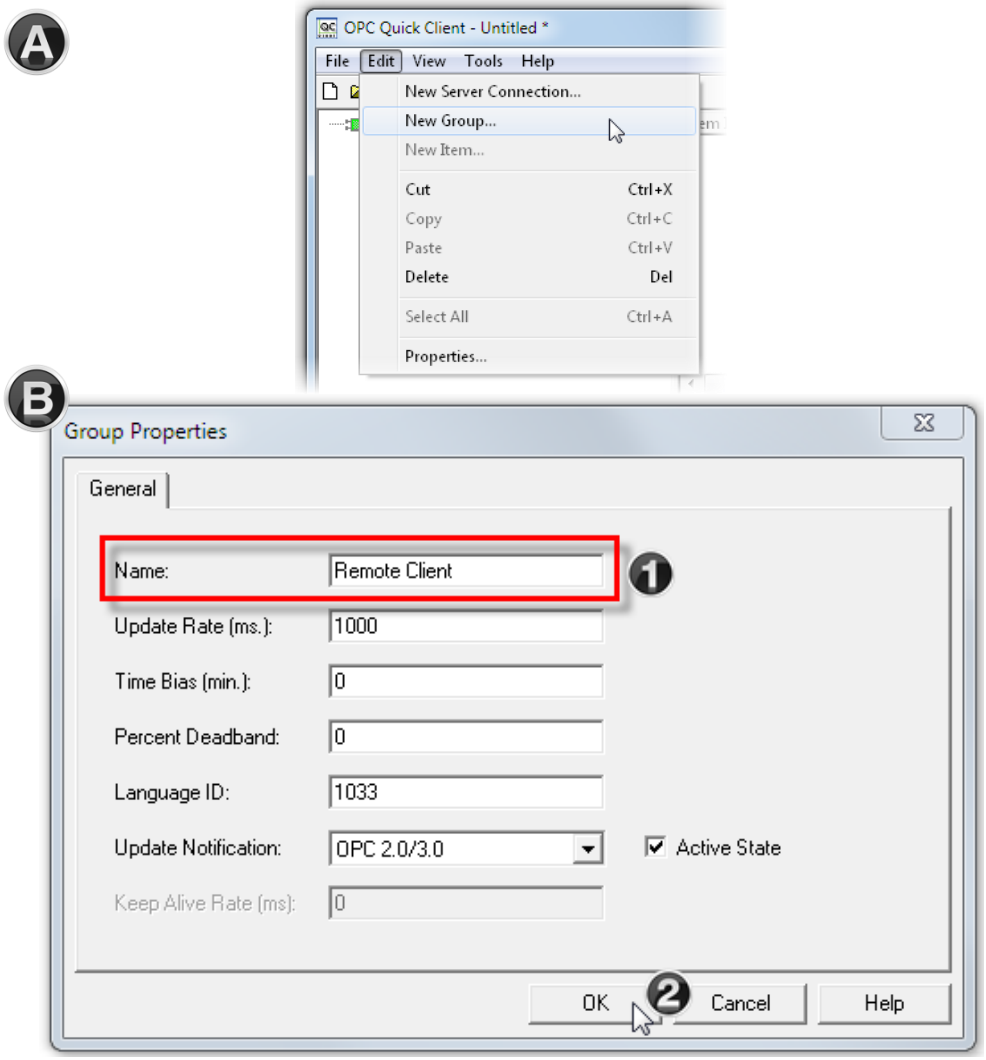
- ***Create a New Server Connection to the Shared Variable Engine***
  Go to ***Edit » New Server Connection…*** as shown in ***Figure A***. This opens a dialog box to configure the access to existing OPC servers.

  Select the Shared Variable Engine OPC server labeled as ***National Instruments.Variable Engine.1 [1]*** as shown in ***Figure B*** and click on the OK button ***[2].***

  This creates a connection to ***LabVIEW,*** which is working as an ***OPC server*** and is exposing the variables of the vibration monitoring application.

15. Create a new group to host items on the OPC quick client.

Create a new group to store items on the OPC quick client. Go to ***Edit » New Group…*** and name the group ***Remote Client***.



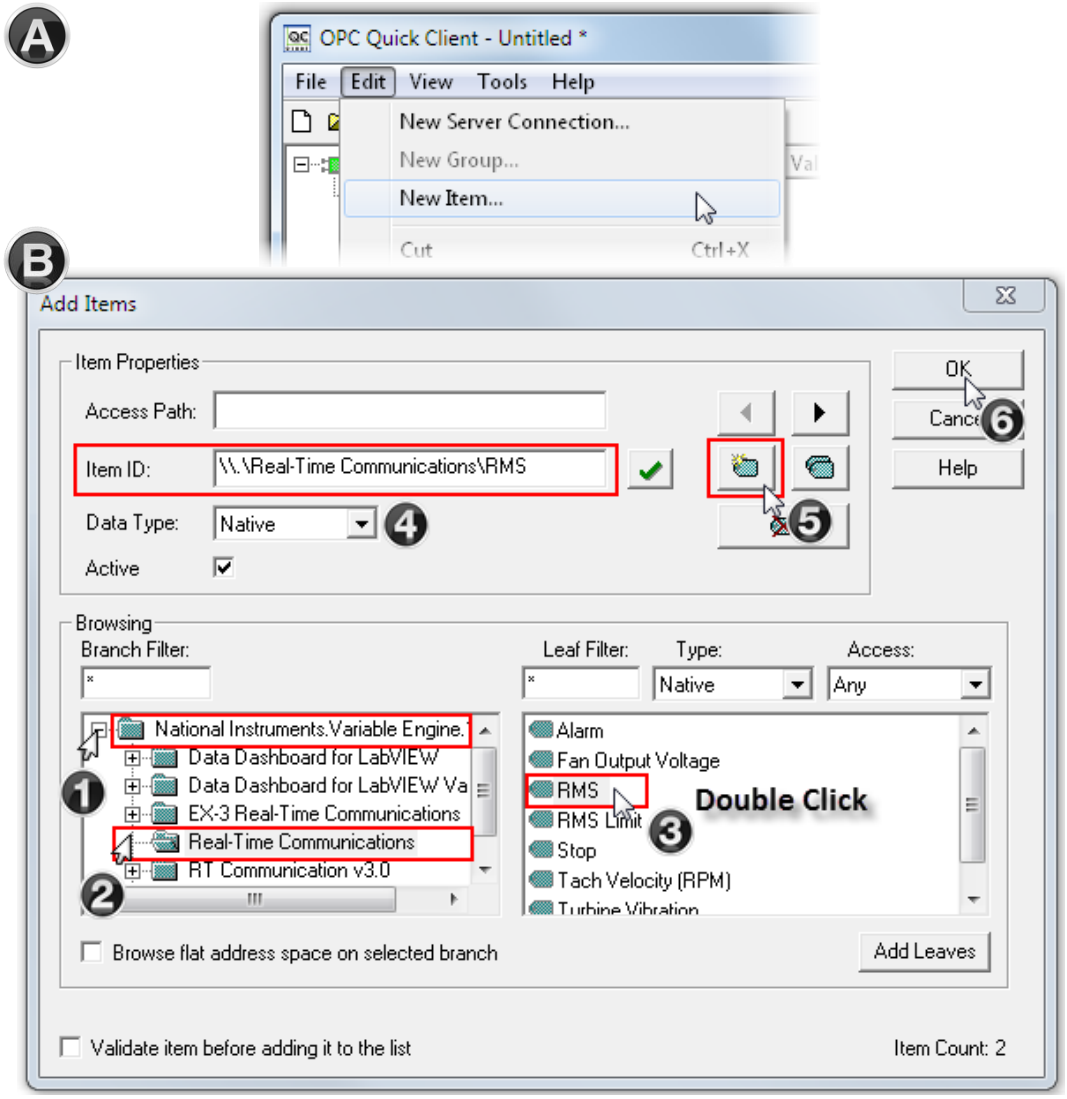
DETAILED INSTRUCTIONS

- ***Create a New Group***
  Create a new group to store items on the OPC quick client. Go to ***Edit » New Group…*** and name the group as ***Remote Client [1]*** as shown in ***figures A*** and ***B***. Leave any other fields with default values and click on OK ***[2].***

  This creates a repository for tags linked to the variables living in the ***Shared Variable Engine***.

16. Add new items to the group.

Expand each hierarchy to reveal the measurement modules and I/O channels that you will use during the exercise.



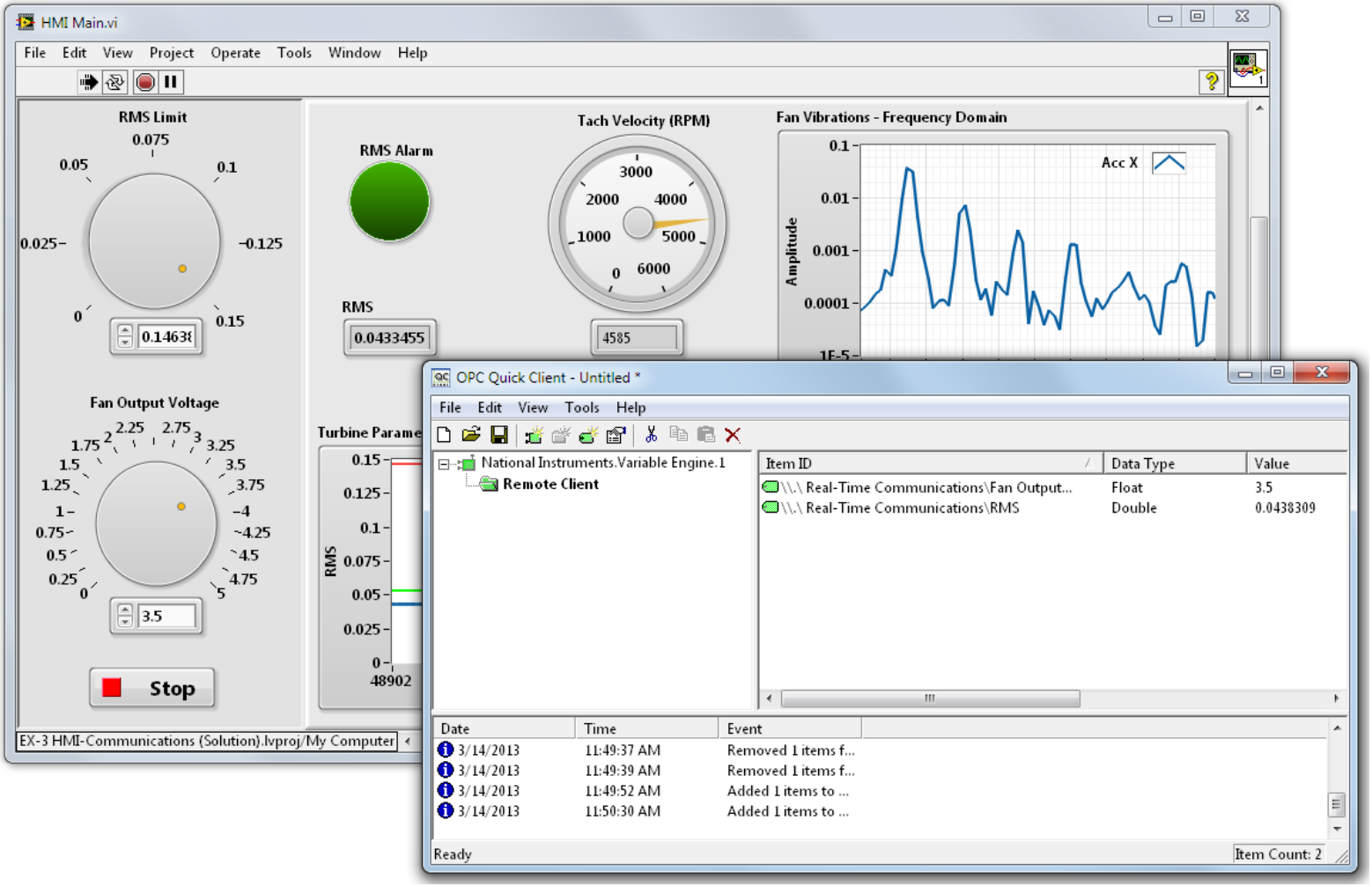
DETAILED INSTRUCTIONS

- ***Add Items to the Group***
  Add items to the group by going to ***Edit » New Item…*** as shown in ***Figure A***. This opens the Add Items window, which you can use to select the desired items as shown in ***Figure B***:

  1. Expand the ***National Instruments.Variable Engine.1*** server to reveal the published libraries
  2. Select the library ***Real-Time Communications,*** which is the one used in the vibration monitoring application
  3. Select one of the variables by double-clicking it
  4. The ***Item ID*** field is automatically populated with the right syntax to access the variable
  5. Create additional items if desired
  6. Save changes by clicking on the OK button

17. Run the application.

Run the application as explained in Step 11 and observe how the OPC client automatically retrieves data from the Shared Variable Engine.



18. Stop the application.

Stop the execution of the code by pressing the Stop button on the front panel of HMI Main.vi.

Part E—CHALLENGE

19. Add logging capabilities to the application.

Add the necessary code to create a spreadsheet file containing the current RMS value, RPM, output voltage, and system time whenever an alarm goes off.

<END OF EXERCISE 3—HUMAN MACHINE INTERFACE (HMI) AND COMMUNICATIONS>

ADDITIONAL RESOURCES

Required Software

The following software is installed on the computers.

- LabVIEW 2013 SP1 Professional Development System
- LabVIEW Real-Time Module 2013
- LabVIEW FPGA Module 2013
- NI OPC Servers 2013 (Optional)
- PID Toolkit 2013
- Xilinx Compile Tools 14.4
- NI RIO 13.1
- Note: You can also use LabVIEW2013 and RIO 13.0 without having to recompile the FPGA code.

Required Hardware

The following hardware is used.

- NI cRIO-9022 (Controller) or NI cRIO 9076 (Controller + Chassis)
- NI cRIO-9113 (Chassis)
  - Slot 1 NI 9211 (Thermocouple)
  - Slot 2 NI 9474 (Digital Output)
  - Slot 3 NI 9234 (Accelerometer / Tachometer)
  - Slot 4 NI 9263 (Analog Output)

BUILD YOUR OWN EMBEDDED SYSTEM WORKSHOP

In the NI Build Your Own Embedded System hands-on workshop, focus on extending your NI LabVIEW skills into FPGA-based embedded design using NI reconfigurable I/O (RIO) hardware.

Purchase the LabVIEW RIO Evaluation kit through the registration process and attend the workshop to receive the following:

- 90-day evaluation of LabVIEW and the LabVIEW FPGA and LabVIEW Real-Time modules
- Board-level NI RIO evaluation hardware device and daughterboard for easy I/O interfacing
- Introduction to the NI LabVIEW RIO architecture with a qualified NI instructor
- Hands-on experience building your first FPGA-based RIO embedded system with the evaluation kit

To find an event in your area, check out ***www.ni.com/byoes***.

LabVIEW RIO EVALUATION KIT

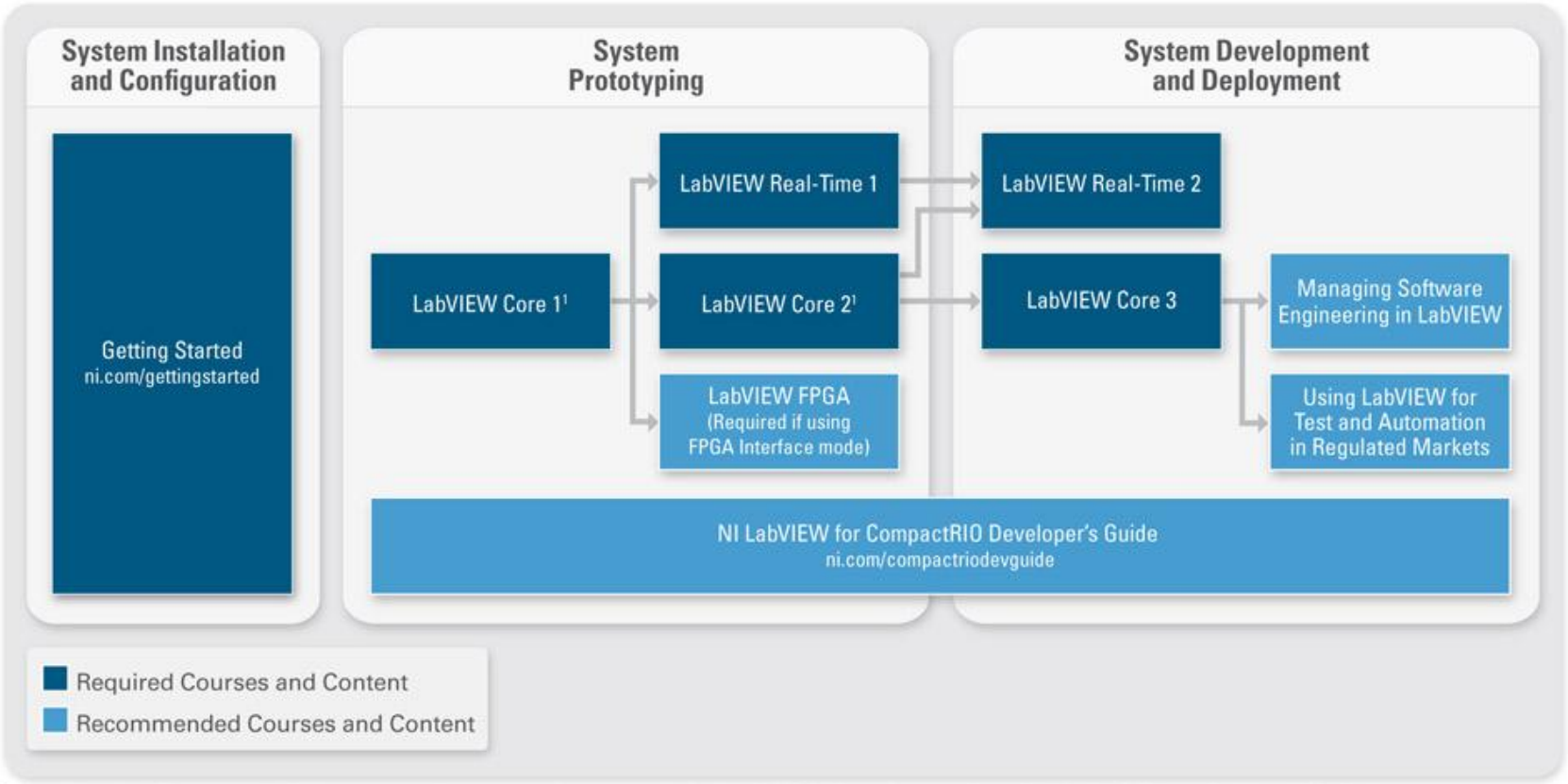
NATIONAL INSTRUMENTS LabVIEW™

Using the evaluation kit, develop an embedded system with the LabVIEW RIO architecture. LabVIEW system design software helps you program NI reconfigurable I/O (RIO) hardware, which includes a real-time processor, FPGA, and I/O. NI CompactRIO uses this same architecture for prototyping through deployment with a flexible array of configuration, expansion, and NI C Series module I/O options.

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

To learn more, visit ***www.ni.com/rioeval***

NI LabVIEW Real-Time and NI LabVIEW FPGA RECOMMENDED RESOURCES AND TRAINING OPTIONS



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