

LabVIEW™ FPGA Course Exercises

Course Software Version 2012
August 2012 Edition
Part Number 323662F-01

Copyright

© 2003–2012 National Instruments. All rights reserved.
Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

National Instruments respects the intellectual property of others, and we ask our users to do the same. NI software is protected by copyright and other intellectual property laws. Where NI software may be used to reproduce software or other materials belonging to others, you may use NI software only to reproduce materials that you may reproduce in accordance with the terms of any applicable license or other legal restriction.

End-User License Agreements and Third-Party Legal Notices

You can find end-user license agreements (EULAs) and third-party legal notices in the following locations:

- Notices are located in the <National Instruments>_Legal Information and <National Instruments> directories.
- EULAs are located in the <National Instruments>\Shared\MDF\Legal\license directory.
- Review <National Instruments>_Legal Information.txt for more information on including legal information in installers built with NI products.

Trademarks

LabVIEW, National Instruments, NI, ni.com, the National Instruments corporate logo, and the Eagle logo are trademarks of National Instruments Corporation. Refer to the *Trademark Information* at ni.com/trademarks for other National Instruments trademarks.

Other product and company names mentioned herein are trademarks or trade names of their respective companies. Members of the National Instruments Alliance Partner Program are business entities independent from National Instruments and have no agency, partnership, or joint-venture relationship with National Instruments.

Patents

For patents covering National Instruments products/technology, refer to the appropriate location: **Help>Patents** in your software, the `patents.txt` file on your media, or the *National Instruments Patent Notice* at ni.com/patents.

Worldwide Technical Support and Product Information

ni.com

Worldwide Offices

Visit ni.com/niglobal to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 683 0100

To comment on National Instruments documentation, refer to the National Instruments Web site at ni.com/info and enter the Info Code feedback.

Contents

Student Guide

A. NI Certification	vii
B. Course Description	viii
C. What You Need to Get Started	viii
D. Installing the Course Software.....	ix
E. Course Goals.....	ix
F. Course Conventions	x

Lesson 2

Getting Started With LabVIEW FPGA

Exercise 2-1	Connecting and Configuring CompactRIO	2-1
Exercise 2-2	Configuring the CompactRIO System.....	2-5
Exercise 2-3	Creating a LabVIEW FPGA Project for an R-Series Device	2-13
Exercise 2-4	Creating a LabVIEW FPGA Project for CompactRIO.....	2-17

Lesson 3

Programming Using LabVIEW FPGA

Exercise 3-1	Executing a VI on the Development Computer.....	3-1
Exercise 3-2	Executing a VI on the FPGA Target.....	3-5

Lesson 4

Using FPGA I/O

Exercise 4-1	Acquiring Data From an R Series Device	4-1
Exercise 4-2	Acquiring Data From a CompactRIO	4-5

Lesson 5

Timing an FPGA VI

Exercise 5-1	Timing a While Loop.....	5-1
Exercise 5-2	Benchmarking a While Loop	5-5

Lesson 6**Executing Code in Single-Cycle Timed Loops**

Exercise 6-1	Comparing a While Loop and a Single-Cycle Timed Loop	6-1
Exercise 6-2	Fixing SCTL Errors	6-4
Exercise 6-3	Implementing Pipelining.....	6-10

Lesson 7**Signal Processing**

Exercise 7-1	Using the Fixed-Point Data Type	7-1
Exercise 7-2	Using the Four-Wire Handshaking Protocol	7-6

Lesson 8**Sharing Data on FPGA**

Exercise 8-1	Transferring Buffered Data on the FPGA.....	8-1
--------------	---	-----

Lesson 10**Modular Programming**

Exercise 10-1	Creating an FPGA SubVI	10-1
---------------	------------------------------	------

Lesson 11**Communicating Between the FPGA and Host**

Exercise 11-1	Developing a Windows Host VI.....	11-1
Exercise 11-2	Developing a Real-Time Host VI	11-9
Exercise 11-3	Transferring Buffered Data Using DMA FIFO	11-25
Exercise 11-4	Interleaving a DMA FIFO	11-33

Appendix A

Alternate CompactRIO Controller Instructions

A. Using an Alternate CompactRIO Controller/Chassis	A-2
B. Hardware Setup.....	A-2
C. Hardware Configuration	A-2
D. Creating a New LabVIEW FPGA Project	A-2
E. Modifying an Existing LabVIEW FPGA Project	A-3

Appendix B

Course Slides

Appendix C

Additional Information and Resources

Student Guide

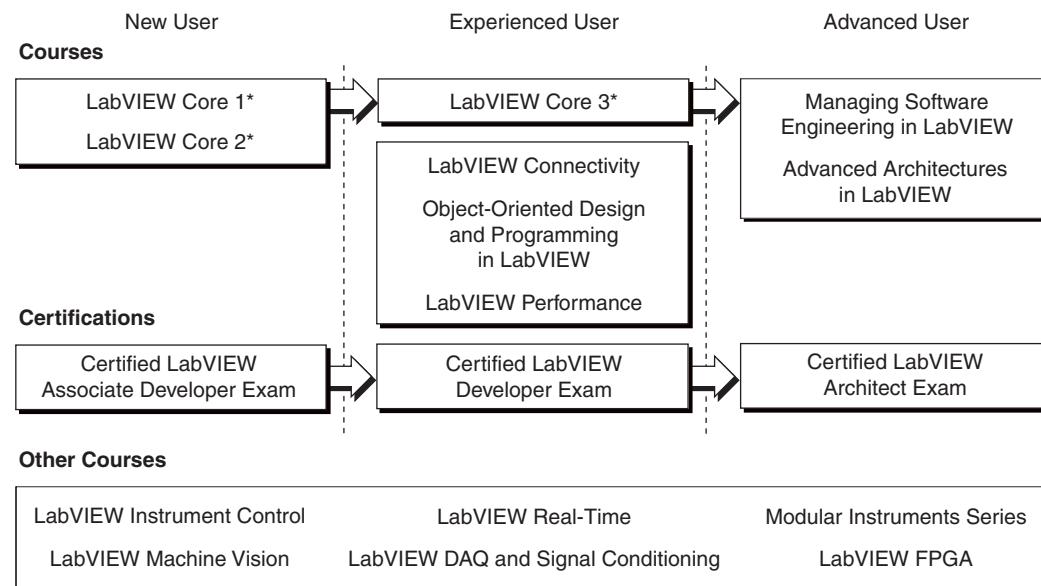
Thank you for purchasing the *LabVIEW FPGA* course kit. This course manual and the accompanying software are used in the three-day, hands-on *LabVIEW FPGA* course.

You can apply the full purchase price of this course kit toward the corresponding course registration fee if you register within 90 days of purchasing the kit. Visit ni.com/training to register for a course and to access course schedules, syllabi, and training center location information.

 **Note** For course manual updates and corrections, refer to ni.com/info and enter the Info Code lvfpga.

A. NI Certification

The *LabVIEW FPGA* course is one of many courses offered by National Instruments. The following illustration shows the courses that are part of the LabVIEW training series. If you want to build your proficiency with LabVIEW and prepare for exams to become an NI Certified LabVIEW Developer and NI Certified LabVIEW Architect, refer to ni.com/training.



B. Course Description

The *LabVIEW FPGA* course teaches you to extend LabVIEW to field-programmable gate array (FPGA) applications. You can use LabVIEW to create custom FPGA applications that run on NI reconfigurable I/O hardware. LabVIEW can execute block diagrams in hardware. This course assumes you have taken the *LabVIEW Core 1* course or have equivalent experience. The *LabVIEW Real-Time 1* course is recommended but not required.

In the course manual, each lesson consists of the following:

- An introduction that describes the purpose of the lesson and what you will learn
- A description of the topics in the lesson
- A summary quiz that tests and reinforces important concepts and skills taught in the lesson

In the exercise manual, each lesson consists of the following:

- A set of exercises to reinforce topics
- (Optional) Self-study and challenge exercise sections or additional exercises

C. What You Need to Get Started

Before you use this course manual, make sure you have the following items:

- Computer running Windows 7/Vista/XP
- LabVIEW Full or Professional Development System 2012 or later
- LabVIEW FPGA Module version 2012 or later
- LabVIEW Real-Time Module version 2012 or later
- NI-RIO 12.0 or later
- cRIO-9074 integrated chassis and controller
- NI 9211 thermocouple module
- NI 9234 analog input module

- NI 9263 analog output module
- NI Sound and Vibration Signal Simulator
- LabVIEW FPGA Course* manual
- LabVIEW FPGA Course Exercises* manual
- LabVIEW FPGA* course CD containing the following files:

Folder	Description
Exercises	Folder for saving VIs created during the course and for completing certain course exercises; also includes subVIs necessary for some exercises
Solutions	Folder containing the solutions to all the course exercises

D. Installing the Course Software

Insert the course CD and follow the onscreen instructions to install the software.

Exercise files are located in the <Exercises>\LabVIEW FPGA\ folder, where <Exercises> represents the path to the Exercises folder on the root directory of your computer.

E. Course Goals

This course presents the following topics:

- Design and implement applications using the LabVIEW FPGA Module
- Control timing and synchronization on the FPGA target
- Compile your LabVIEW FPGA VI and deploy to NI RIO hardware
- Create deterministic control and simulation solutions on the NI LabVIEW platform

This course does not present any of the following topics:

- Every built-in VI, function, or object; refer to the *LabVIEW Help* for more information about LabVIEW features not described in this course
- Developing a complete application for any student in the class; refer to the NI Example Finder, available by selecting **Help»Find Examples**, for example VIs you can use and incorporate into VIs you create

F. Course Conventions

The following conventions are used in this course manual:

»	The » symbol leads you through nested menu items and dialog box options to a final action. The sequence Options»Settings»General directs you to pull down the Options menu, select the Settings item, and select General from the last dialog box.
	This icon denotes a tip, which alerts you to advisory information.
	This icon denotes a note, which alerts you to important information.
	This icon denotes a caution, which advises you of precautions to take to avoid injury, data loss, or a system crash.
bold	Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names.
<i>italic</i>	Italic text denotes variables, emphasis, a cross-reference, or an introduction to a key concept. Italic text also denotes text that is a placeholder for a word or value that you must supply.
monospace	Text in this font denotes text or characters that you enter from the keyboard, sections of code, programming examples, and syntax examples. This font also is used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames, and extensions.
monospace bold	Bold text in this font denotes the messages and responses that the computer automatically prints to the screen. This font also emphasizes lines of code that are different from the other examples.

Getting Started With LabVIEW FPGA

Exercise 2-1 Connecting and Configuring CompactRIO

Goal

Set up the CompactRIO (cRIO) hardware and verify proper connections.

Description

The hardware in your CompactRIO system includes the following components:

- NI cRIO-9074 Integrated Real-Time Controller
- NI 9211 thermocouple input module
- NI 9234 analog input module
- NI 9263 analog output module



Note For instructions on adapting the exercises in this manual for an alternate cRIO controller, refer to Appendix A, *Alternate CompactRIO Controller Instructions*.

Connect your CompactRIO system to the National Instruments Sound and Vibration Signal Simulator. The NI 9263 analog output module controls the fan speed. The NI 9234 analog input module measures fan speed and fan vibration. A tachometer determines fan speed by measuring rotation speed. A two-axis accelerometer measures fan vibration.

The NI 9211 thermocouple input module measures temperatures.

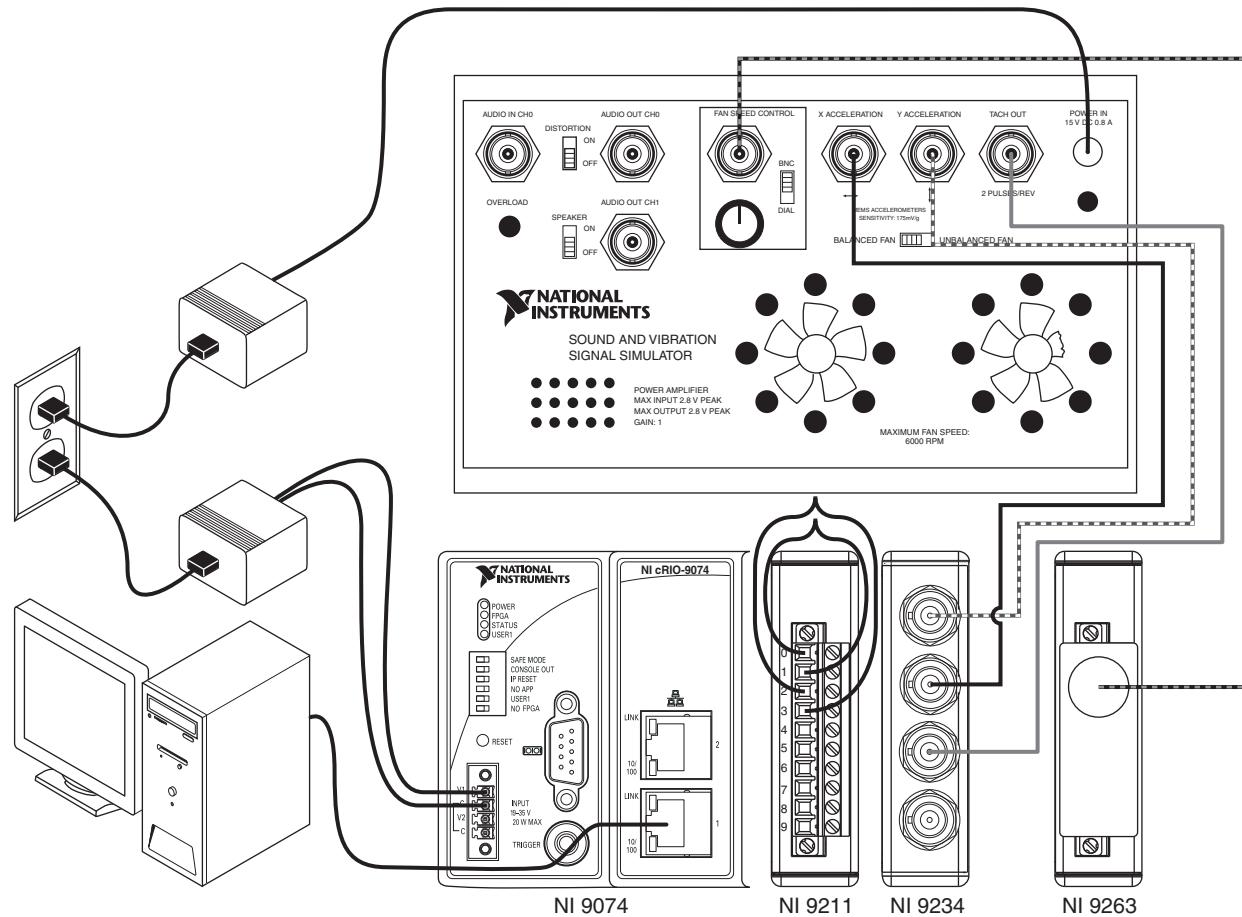
If necessary, refer to the CompactRIO manuals located at <Exercises>\LabVIEW FPGA\Hardware Manuals for more information.

Implementation

Set Up Hardware

1. Refer to Figure 2-1 as you set up your CompactRIO system.

Figure 2-1. System Wiring Diagram for LabVIEW FPGA Course



NI cRIO-9074

1. Connect the Ethernet crossover cable to your PC.
2. Connect the red wire from the power supply to the V1 input.
3. Connect the black wire from the power supply to the C input.
4. Set all the DIP switches on the cRIO-9074 to the OFF position.

NI 9211

1. Connect one of the thermocouples to channel TC0.

Connect the white wire on line 0.

Connect the red wire on line 1.

2. Connect the other thermocouple to channel TC1.

Connect the white wire on line 2.

Connect the red wire on line 3.

NI 9234

 **Note** You can also complete the exercises in this course using a NI 9233. Setup instructions are the same as below.

1. Connect BNC channel 0 to the Y ACCELERATION connection of the Sound and Vibration Signal Simulator.
2. Connect BNC channel 1 to the X ACCELERATION connection of the Sound and Vibration Signal Simulator.
3. Connect BNC channel 2 to the TACH OUT connection of the Sound and Vibration Signal Simulator.

NI 9263

1. Connect the Analog Output line to the FAN SPEED CONTROL of the Sound and Vibration Signal Simulator.

Complete Setup

1. Connect the power supply to the POWER IN 15 V DC 0.8 A input on the Sound and Vibration Signal Simulator.
2. Plug in both power supplies.
3. Watch the LEDs carefully and confirm the power-on self test (POST). During the POST, the POWER, and STATUS LED turns on. The STATUS LED turns off, indicating that the POST is complete.
4. Wait for the system to boot and confirm that the POWER LED remains on after booting. The STATUS LED might blink continuously and slowly if the unit has not yet been configured.

Review CompactRIO Manuals (Optional)

If you need additional information when you set up the system, refer to the manuals located in the <Exercises>\LabVIEW FPGA\Hardware Manuals directory.

End of Exercise 2-1

Exercise 2-2 Configuring the CompactRIO System

Goal

Configure the cRIO target using NI Measurement & Automation Explorer (MAX).

Throughout the course, you use cRIO-9074 controller as your Real-Time target. For instructions on adapting the exercises in this manual for an alternate cRIO controller, refer to Appendix A, *Alternate CompactRIO Controller Instructions*.

For more information about configuring real-time targets, refer to the *MAX Remote Systems Help*. Select **Help»Help Topics»Remote Systems** in MAX.

Implementation

Configuring the IP Address

Follow the instructions for your computer's operating system (Windows XP or Windows 7).

Windows XP

1. If you are attending an instructor-led course and your computer is connected to the Internet, disconnect your Ethernet cable.
2. Verify that your network is set to obtain an IP address automatically.

- Click **Start**. Select **Control Panel»Network Connections**.
- Right-click **Local Area Connection** and select **Properties**.
- Select **Internet Protocol (TCP/IP)** and click **Properties**.

If the **Obtain an IP address automatically** option is selected, your network uses DHCP or AutoIP. Otherwise, your network uses a static IP address.

- If you are attending an instructor-led course, verify that the **Obtain an IP address automatically** option is selected.
- Click **OK**.
- Close the Local Area Connection Properties dialog box.

Windows 7

1. If you are attending an instructor-led course and your computer is connected to the Internet, disconnect your Ethernet cable.
2. Verify that your network is set to obtain an IP address automatically.

- Click **Start**. Select **Control Panel»Network and Sharing Center**.
- Click **Local Area Connection** and click **Properties**.
- Select **Internet Protocol Version 4 (TCP/IPv4)** and click **Properties**.

If the **Obtain an IP address automatically** option is selected, your network uses DHCP or AutoIP. Otherwise, your network uses a static IP address.

- If you are attending an instructor-led course, verify that the **Obtain an IP address automatically** option is selected.
- Click **OK**.
- Close the Local Area Connection Properties dialog box.

Viewing the IP Address of the Host Computer

Complete the following steps to view and record the IP address and subnet mask of the host computer.

1. In Windows, select **Start»Run** and enter cmd to open a command prompt.
2. At the command prompt, enter ipconfig to find the IP address and Subnet mask values for the host computer. Record them below:

- Host Computer IPv4 Address: _____
- Host Computer Subnet Mask: _____



Note If you are attending an instructor-led course, the host computer is connected directly to the RT target using a cross-over cable, so the host computer should default to a link local IP address. Therefore, the host computer IPv4 address should be 169.254.[x].[y], where [x] and [y] are numbers between 0 and 255. The host computer subnet mask should be 255.255.0.0.

3. Close the command prompt when finished.

Configuring the cRIO

1. Connect the cRIO-9074 to the network. Your system may already be connected.
 - If you are attending an instructor-led course, connect the RT target to the computer using a cross-over cable.
 - If the system uses DHCP, connect the RT target to the nearest hub using a standard Ethernet cable.
2. In Exercise 2-1, you set up the CompactRIO hardware and verified proper connections. Verify the following:
 - Both the Sound and Vibration Signal Simulator and the cRIO-9074 are powered on with cables connected.
 - The POWER LED is on.
 - The STATUS LED might blink continuously and slowly if the unit has not yet been configured.

Reformatting the Controller

Complete the following steps to reformat the disk on the CompactRIO target:

1. Put the CompactRIO target in safe mode. To reformat a CompactRIO target, the target must be in safe mode.
 - Set the SAFE MODE switch on the CompactRIO target to the ON position.
 - Set the IP RESET switch on the CompactRIO target to the ON position.
 - Press the reset button on the CompactRIO target.

The reset button is a small button on the controller that can be depressed with a small object, such as a pen.

- Wait for power-on self test (POST) to complete. After the POST completes, the STATUS LED repeatedly flashes three times to indicate that the CompactRIO target is in safe mode.
2. Launch MAX from the desktop or select **Start»All Programs»National Instruments»Measurement & Automation Explorer**.
3. Verify that MAX detects the CompactRIO target.
 - Expand **Remote Systems** in the configuration tree.

- Verify that the CompactRIO target appears under Remote Systems.
 - If no device is listed, refresh the list by selecting **View»Refresh** or pressing <F5>.
4. Reformat the disk on the CompactRIO target.
- Right-click the CompactRIO target under Remote Systems and select **Format Disk**.
 - Click **Format** in the Format Disk dialog.
 - A dialog box appears when the disk on the CompactRIO has been formatted successfully. Click **OK** in this dialog box to reboot the target.

Configuring the Controller

If you are using a crossover Ethernet cable to connect your CompactRIO target to your host computer, you must set the host computer to a static IP address.

Configuring the Real-Time Target

1. Verify that MAX detects the CompactRIO target.

- Expand the **Remote Systems** tree.
- Verify that the CompactRIO target appears in the Remote Systems tree.
- If no device is listed, refresh the list by clicking **Refresh** or pressing <F5>.

2. Configure the CompactRIO target to use a link local IP address.

- Select the CompactRIO target and view the **Network Settings** tab.
- If IPv4 Address is anything other than 0.0.0.0, then verify that Configure IPv4 Address is set to **DHCP or Link Local**.

Your reformatted CompactRIO currently supports an automatic IP assignment behavior where the RT target tries to obtain an IP address from the DHCP server first and then assigns a link local address if that fails.

- If IPv4 Address is 0.0.0.0, then complete the following items.

Your CompactRIO currently does not support the automatic IP assignment behavior described in the previous bullet. However, after you install LabVIEW Real-Time software on the CompactRIO later in this exercise, the CompactRIO will support the automatic IP assignment behavior.

- Verify that Configure IPv4 Address is set to **Static**.
 - Set **IPv4 Address** to $[a].[b].[c].[d]$, where $[a]$, $[b]$, and $[c]$ match the first three numbers of your host computer IP address, and $[d]$ is a number between 0 and 255 that does not match the fourth number of the Host Computer IP Address you recorded in a previous step. For example, you might set the IPv4 Address to 169.254.80.5.
 - Set the Subnet Mask to 255.255.0.0.
 - Set the Gateway and DNS Server to blank values.
- Select the **System Settings** tab.
 - Verify that the **Halt on IP Failure** checkbox is disabled.
-  **Note** When a CompactRIO that supports automatic IP assignment behavior is configured to use a DHCP or link local IP address and the Halt on IP Failure checkbox is disabled, the CompactRIO will use a link local address if it does not find a DHCP server. If you are attending an instructor-led course and using a CompactRIO that supports automatic IP assignment behavior, you connect directly to the CompactRIO to your computer using a cross-over cable, so the CompactRIO will not find a DHCP server and will use a link local IP address instead.
- Set the SAFE MODE switch on the CompactRIO target back to the OFF position.
 - Set the IP RESET switch on the CompactRIO target back to the OFF position.
 - Click the **Save** button unless it is dimmed.
 - Restart your system, if you are prompted to do so. If not prompted to restart, press and release the reset button on the CompactRIO target.
 - Wait for the POST to complete.
3. View the IP address of your CompactRIO RT target.
 - In MAX, select **View»Refresh**.
 - Expand **Remote Systems** and select the CompactRIO target.
 - View the **Network Settings** tab.

- Record the IP address and Subnet mask values for the CompactRIO RT target for future reference.

- CompactRIO RT Target IPv4 Address: _____

- CompactRIO RT Target Subnet Mask: _____



Note When an RT target is configured to use a DHCP or link local IP address, the RT target may not always have the same IP address after rebooting. In later exercises, if you cannot find your RT target at the IP address recorded above, check the current IP address of the RT target in the Network Settings tab in MAX and use the current IP address instead. Name the CompactRIO RT target.

4. Name the CompactRIO RT target.

- Click the **System Settings** tab and enter cRIO-9074 in the **Hostname** field of the General Settings section.

- Click **Save** to apply the new name.

5. Verify that the correct software is installed on your system.

- Expand the CompactRIO target under **Remote Systems**.

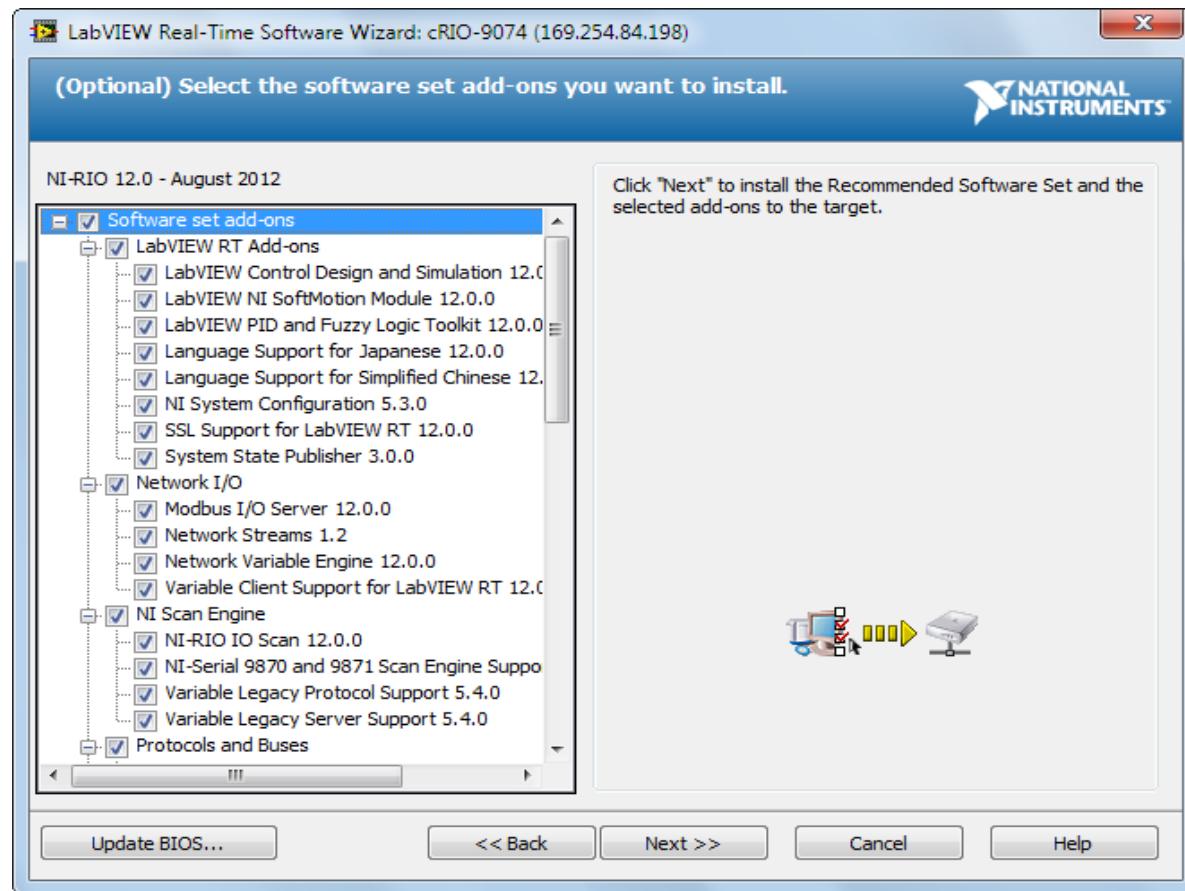
- Right-click **Software** in the configuration tree under cRIO-9074 and select **Add/Remove Software**.

- If a warning dialog appears, click **OK**.

- Select **LabVIEW Real-Time 12.0»NI-RIO 12.0** and click **Next**.

- Enable **Software set add-ons**, as shown in Figure 2-2.

Figure 2-2. LabVIEW Real-Time Software Wizard



- Click **Next**.
- Review the installation selections and click **Next** to begin the installation.
- After the software is installed, click **Finish** to exit the LabVIEW Real-Time Software Wizard.

6. Identify the chassis used in the target.

- In MAX, expand **Devices and Interfaces** under cRIO-9074 in the configuration tree.
- Verify that **RIO0** is displayed.
- Click the **RIO0** item to view the information available in the **General** tab.

End of Exercise 2-2

Exercise 2-3 Creating a LabVIEW FPGA Project for an R-Series Device

Goal

Create a LabVIEW FPGA project for an offline R-Series device.

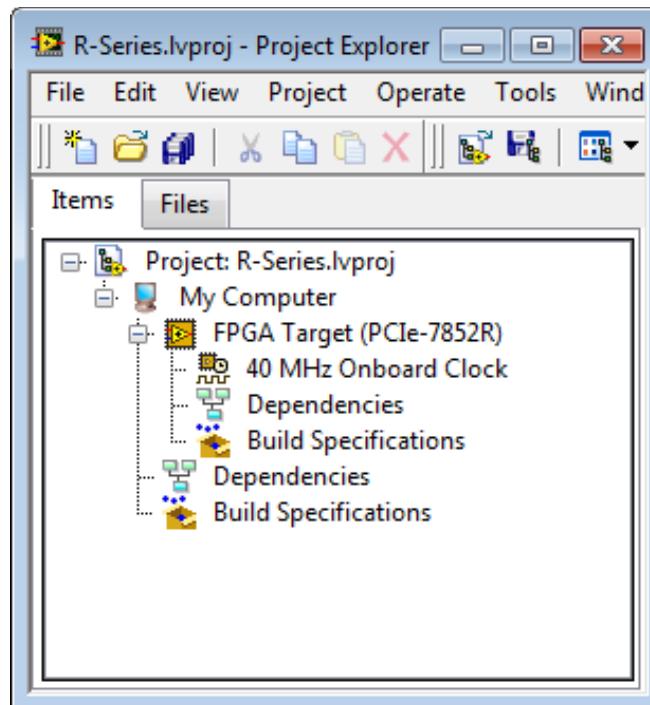
Scenario

You are a system integrator and you must get started on an application for a customer, but the FPGA device has not arrived or has not been purchased yet. You know the model will be an NI PCIe-7852R. Before you begin development, you must create a LabVIEW project that contains the proper configuration for the FPGA target on the NI PCIe-7852R.

Implementation

In this exercise, you build the project shown in Figure 2-3.

Figure 2-3. Basic R-Series Project Under a Windows Computer



1. Launch LabVIEW and verify that the FPGA Module is installed.

- Open LabVIEW from the desktop or select **Start»All Programs»National Instruments»LabVIEW 2012**.
- Select **Help»About LabVIEW** to view the LabVIEW launch screen. Look for the LabVIEW FPGA Module logo shown in Figure 2-4. Logos on this launch screen help you identify installed modules. As you mouse over each module, you can view the name of the module

Figure 2-4. LabVIEW FPGA Module Logo



- Close the launch screen to return to the LabVIEW environment.

2. Create a project.

- Click **Create Project** from the Getting Started window.
- Double-click **Blank Project**.
- Select **File»Save Project** and save the project as R-Series.lvproj in the <Exercises>\LabVIEW FPGA\R-Series directory. The name of the project root changes from Untitled Project to R-Series.lvproj.
- Notice that the project contains the root and the My Computer target.

3. Create an FPGA target for the PCIe-7852R.

- Right-click **My Computer** in the Project Explorer window and select **New»Targets and Devices** to display the Add Targets and Devices dialog box.



Note Be sure to right-click the **My Computer** entry and not the **R-Series.lvproj** entry. Targets under **My Computer** are for internal targets on the computer. Since the PCIe-7852R is a PCIe device, it is an internal device. You can add new targets by right-clicking the project. However, targets under **R-Series.lvproj** are for networked targets, such as a CompactRIO. You will be creating a project for a CompactRIO FPGA target in the next exercise.

- Select **New target or device**.
- Expand **R Series**.
- Select the **PCIe-7852R** target and click **OK**.

- In the Project Explorer window, expand the **FPGA Target (PCIe-7852R)** entry.
 - Verify that the **40 MHz Onboard Clock** exists.
4. Save the project.

End of Exercise 2-3

Exercise 2-4 Creating a LabVIEW FPGA Project for CompactRIO

Goal

Create a LabVIEW project for the cRIO-9074 you configured in MAX.

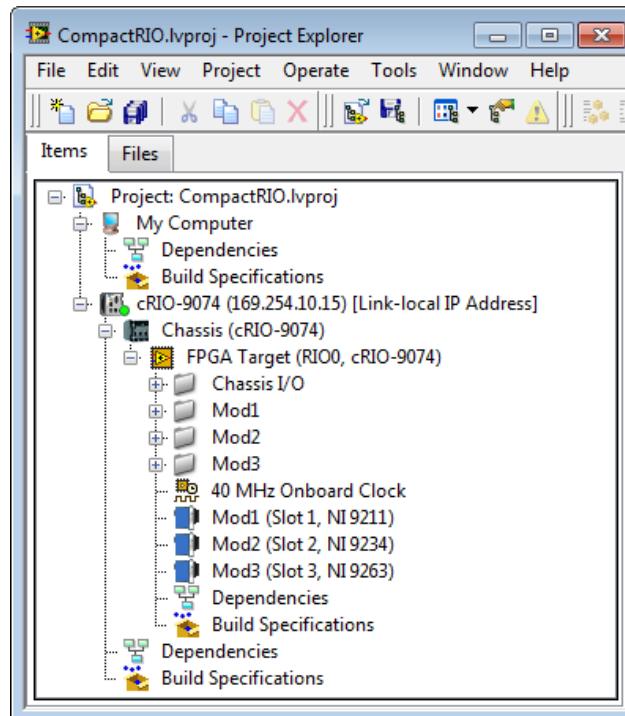
Scenario

The hardware is already configured in MAX. Before you can begin development, you must create a LabVIEW Project that contains the proper configuration for an FPGA target. In this case, the FPGA target is on the cRIO-9074.

Implementation

In this exercise, you build a project similar to the one shown in Figure 2-5.

Figure 2-5. CompactRIO Project



1. Verify that the Real-Time module is installed.

- Select **Help»About LabVIEW** to view the LabVIEW launch screen. Look for the LabVIEW Real-Time logo shown in Figure 2-6. For CompactRIO and Real-Time PXI systems, you need to install both the LabVIEW Real-Time Module and the LabVIEW FPGA Module.

Figure 2-6. Real-Time Icon



- Close the launch screen to return to the LabVIEW environment.

2. Create a project.

- Select **File»Create Project»Blank Project**.
- Select **File»Save Project** and save the project as **CompactRIO.lvproj** in the **<Exercises>\LabVIEW FPGA\CompactRIO** directory. The name of the project root changes from **Untitled Project** to **CompactRIO.lvproj**.
- Notice that the project contains the root and the **My Computer** target.

3. Create the CompactRIO target.

- Right-click **Project: CompactRIO.lvproj** in Project Explorer and select **New»Targets and Devices** to display the Add Targets and Devices dialog box.



Note Be sure to right-click the **CompactRIO** entry and not the **My Computer** entry. You can add new targets by right-clicking **My Computer**. However, targets under **My Computer** must be internal targets on the computer, such as PCI and PCIe Real-Time devices.

- Select **Existing target or device**.
- Expand **Real-Time CompactRIO**.
- Select the **cRIO-9074** target.
- Click **OK**.
- In the Select Programming Mode dialog box, select **LabVIEW FPGA Interface** and click **Continue**. Wait while LabVIEW detects the C Series modules.

4. Verify proper configuration of the project.
 - Expand the **cRIO-9074** entry in the Project Explorer window.
 - Expand the **Chassis (cRIO-9074)** entry.
 - Verify that a node exists for each corresponding module in your CompactRIO chassis. For example, Mod1 (Slot 1, NI 9211).
5. Save the project.
6. Compare the R Series Project with the CompactRIO Project.
 - Notice that both projects have an FPGA Target, but that the targets appear in different hierarchies. For the R Series, the hierarchy is **My Computer»FPGA Target** while the hierarchy for the CompactRIO is **cRIO-9074»Chassis»FPGA Target**.

End of Exercise 2-4

Programming Using LabVIEW FPGA

Exercise 3-1 Executing a VI on the Development Computer

Goal

Create an FPGA VI and verify the functionality of the VI using the development computer.

Scenario

You previously configured your CompactRIO hardware in MAX and created a LabVIEW project. Create an FPGA VI that performs three parallel mathematical operations on two numbers.

Compiling an application can be very time consuming, so you should make sure the code functions properly before compiling. In a simple application, such as this exercise, errors are unlikely. However, as you create more complex applications, it is important that you test the code on the development computer before compiling. Testing on the development computer ensures that you detect and correct errors before you begin a lengthy compilation.

Design

Explore the functions that are available on the Functions palette of an FPGA VI. Design a VI that performs the following three mathematical operations in parallel: add, subtract, and multiply.

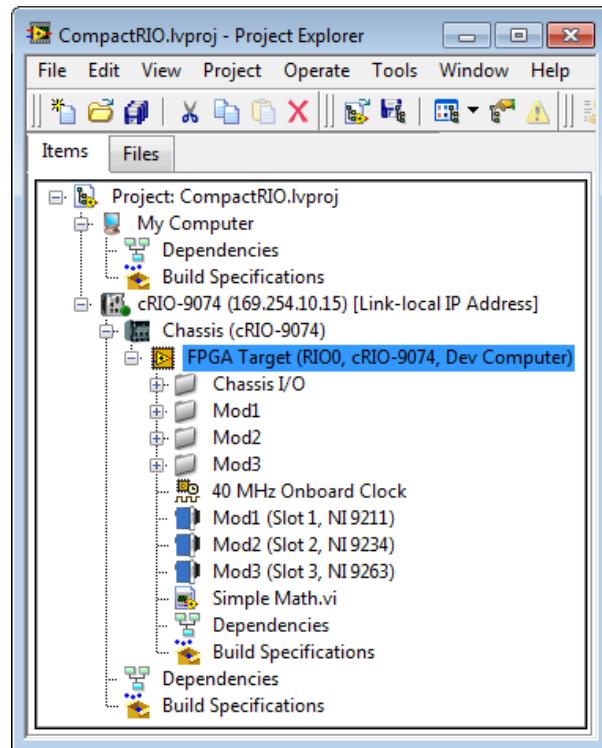
Implementation

1. Open CompactRIO.lvproj from the <Exercises>\LabVIEW FPGA\CompactRIO directory if it is not already open. You created this project in Exercise 2-4.
2. Create a VI on the FPGA Target.
 - Right-click the FPGA Target and select **New»VI**.
 - Save the VI as **Simple Math.vi** in the <Exercises>\LabVIEW FPGA\CompactRIO directory.

3. Set the project to execute the VI on the development computer.

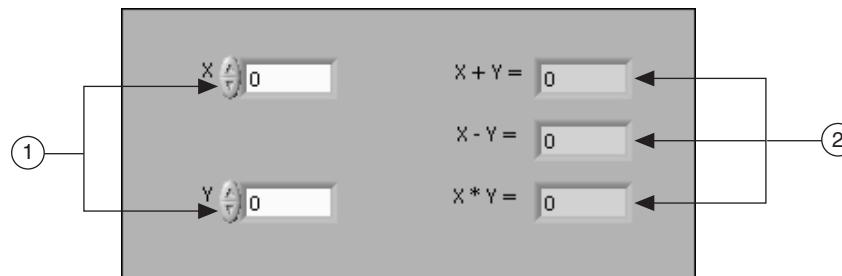
- Right-click the FPGA Target. Select **Execute VI on»Development Computer with Simulated I/O**.
- The project should now resemble Figure 3-1.

Figure 3-1. CompactRIO.lvproj with VI Executing on Development Computer



4. Build the front panel shown in Figure 3-2.

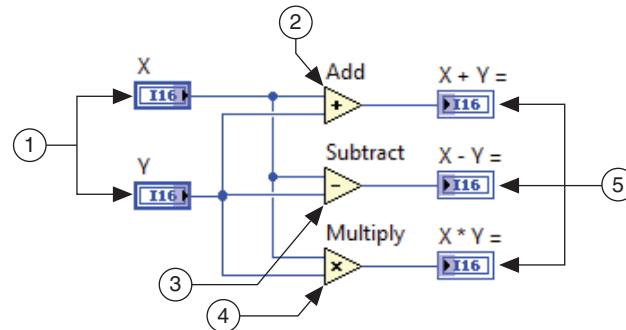
Figure 3-2. Simple Math.vi Front Panel



- 1 Numeric Controls—Search for Numeric Control in the Controls Palette
2 Numeric Indicators—Search for Numeric Indicator in the Controls Palette

5. Complete the block diagram as shown in Figure 3-3.

Figure 3-3. Simple Math.vi Block Diagram



- 1 Numeric Controls
2 Add Function
3 Subtract Function
4 Multiply Function
5 Numeric Indicators

6. Save the VI and project.

Testing

1. Test the VI to verify functionality.
 - Open the front panel of the Simple Math VI and run the VI.
 - Test the functionality of the VI by using several different values for X and Y and verifying that $X+Y=$, $X-Y=$, and $X*Y=$ each return expected values.
2. Use the debugging tools on the Simple Math VI.

Another advantage of testing your VI on the development computer is that you can use the standard LabVIEW debugging tools. The debugging tools help you to quickly diagnose any error that may occur. After you download the application to the FPGA, you can no longer access these tools on the compiled application. It is good practice to debug the FPGA VI as much as possible before you compile the VI for FPGA.

- Open the block diagram of the Simple Math VI.
- Click **Highlight Execution** on the toolbar to turn on Execution Highlighting.



- Update the values on the front panel.
 - Run the VI and observe how the values update on the block diagram.
 - (Optional) Insert breakpoints and probes and run the VI.
3. Turn off Execution Highlighting.
 4. Leave the VI and project open for use in the next exercise.

End of Exercise 3-1

Exercise 3-2 Executing a VI on the FPGA Target

Goal

Compile a working VI to run on an FPGA Target.

Scenario

Compile the Simple Math VI from Exercise 3-1 to run on the cRIO-9074 FPGA. The compile server converts the Simple Math VI to a bitstream file, which can be loaded and executed on the FPGA.

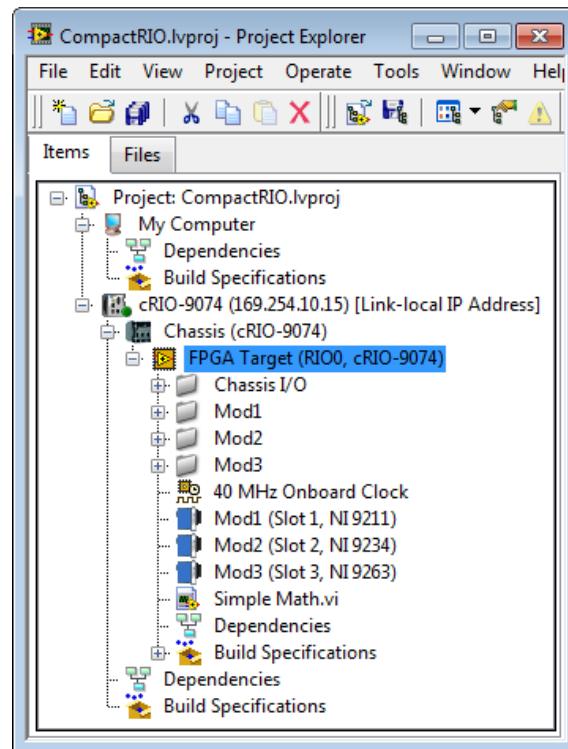
Implementation

Part A

Modify `CompactRIO.lvproj` to execute `Simple Math.vi` on the FPGA target and start the compilation process.

1. Open `CompactRIO.lvproj` from the `<Exercises>\LabVIEW FPGA\CompactRIO` directory if it is not already open.
2. Set the project to execute the VI on the FPGA Target.
 - Right-click the FPGA Target. Select **Execute VI on»FPGA Target**.
 - The project should now resemble Figure 3-4.

Figure 3-4. CompactRIO.lvproj with VI Executing on FPGA Target



3. Estimate the resource usage of `Simple Math.vi`.

- In the Project Explorer, right-click `Simple Math.vi` and select **Create Build Specification**. If prompted to save, click **Yes**.
- Expand **Build Specifications** under the **FPGA Target**.
- Right-click the **Simple Math** build specification and select **Estimate Resource Usage**.
- Verify that the **Use the local compile server** option is selected.

LabVIEW launches the Generating Intermediate Files dialog box. If there are any errors encountered during this process, LabVIEW launches the Code Generation Errors dialog box.

After the intermediate files are generated, LabVIEW launches the Compilation Status window.

- When resource estimation is complete, select **Estimated device utilization (pre-synthesis)** in the Reports pull-down menu of the Compilation Status window.
 - Examine the percentages in this report. If any of the device utilizations exceed 100 percent, you may need to change your design so that it fits on the FPGA.
 - Close the Compilation Status window.
4. Compile Simple Math.vi.
- Open and run Simple Math.vi. When executing on the FPGA Target, clicking the **Run** button in LabVIEW starts the compilation.

When compilation begins, LabVIEW launches the Compilation Status window.

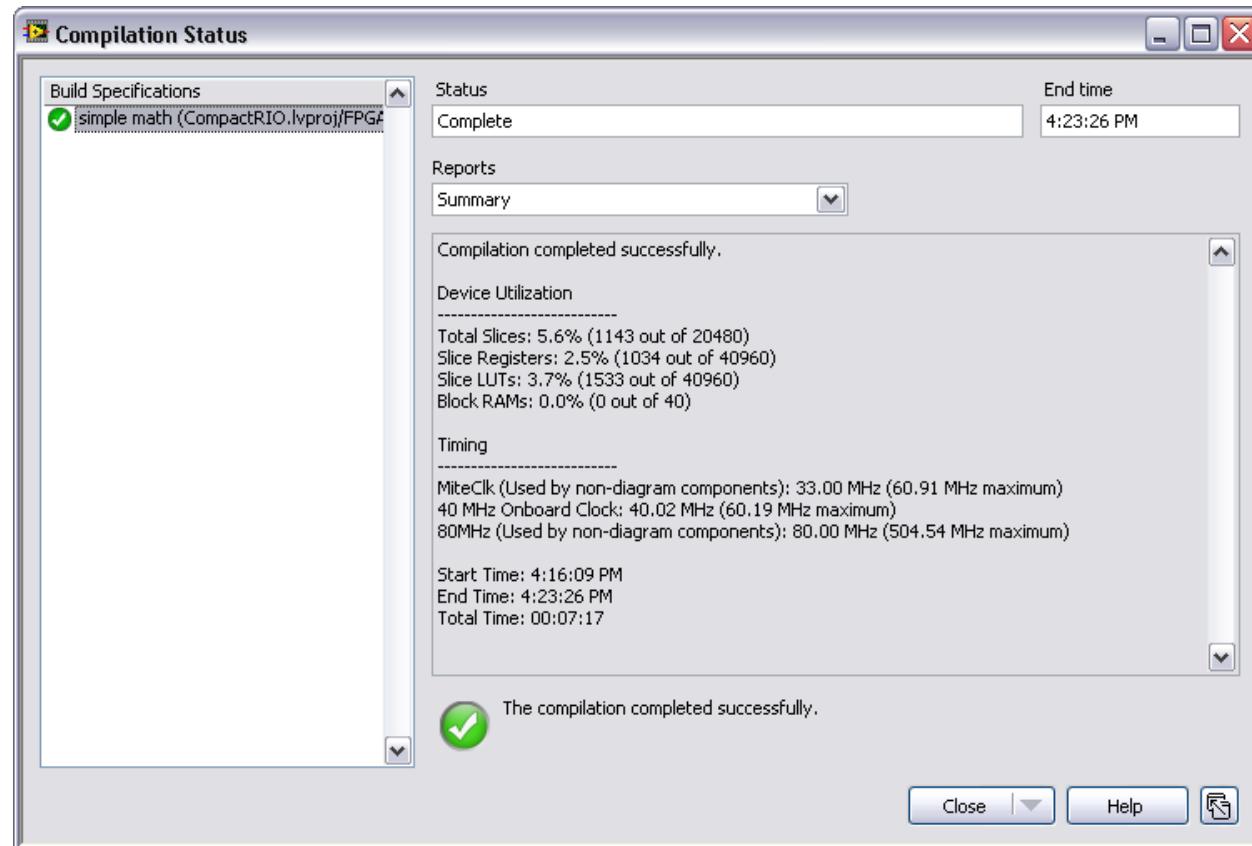


Note You may see a Windows Security Alert dialog if a Windows firewall is blocking the compile server. Click **Unblock** when prompted to continue compilation.

Part B

When compilation is complete, examine the generated reports and test the VI on the FPGA. The Compilation Status window indicates when the compile has finished, as shown in Figure 3-5.

Figure 3-5. Compilation Status Window—Compilation Completed



1. Examine the reports generated during compilation.
 - In the Compilation Status window, **Summary** is selected by default in the Reports pull-down menu. This report contains a summary of the generated bitfile.
 - Select **Configuration** in the Reports pull-down menu. This report displays project information and the Xilinx compiler configuration for the FPGA VI.

- Select **Estimated device utilization (pre-synthesis)** from the Reports pull-down menu. This report contains a summary of the FPGA utilization as estimated before the synthesis of the FPGA VI. Take particular note of the Percent field, which indicates the percentage of the FPGA elements that the VI uses. If any Percent value exceeds 100 percent, then the compilation may fail.
- Select **Estimated device utilization (synthesis)** from the Reports pull-down menu. This report contains a summary of the FPGA utilization as estimated during the synthesis of the FPGA VI. Take particular note of the Percent field, which indicates the percentage of the FPGA elements that the VI uses. If any Percent value exceeds 100 percent, then the compilation may fail.
- Select **Final device utilization (map)** from the Reports pull-down menu. This report contains a summary of FPGA utilization as estimated during the mapping step of compilation.
- Select **Estimated timing (map)** from the Reports pull-down menu. This report contains a summary of the FPGA clocks, as estimated during the mapping of the FPGA VI.
- Select **Final timing (place and route)** from the Reports pull-down menu. This report contains a summary of the FPGA clocks after the routing step of compilation.

2. Close the Compilation Status window.

Testing

1. Run the VI.
 - Change the X and Y values and click the run arrow.
2. Save and close Simple Math.vi.
3. Save and close CompactRIO.lvproj.

End of Exercise 3-2

Using FPGA I/O

Exercise 4-1 Acquiring Data From an R Series Device

Goal

Use I/O in LabVIEW FPGA to acquire analog and digital data from a simulated R Series device.

Scenario

You must design an application to access the analog and digital I/O of an R Series device using FPGA I/O nodes. Since you do not have an R Series device on-hand, you must simulate the PCIe-7852R that will be used in the final application.

Design

Assign unique names to the I/O channels that you acquire data from to make your block diagram more readable. Design a VI that acquires data from those channels and displays them.

Implementation

1. Create a new project with a PCIe-7852R board as the FPGA Target. You created a similar project in Exercise 2-3.
 - Select **Create Project»Blank Project** from the Getting Started window.
 - Select **File»Save Project** and save the project as **R-Series IO.lvproj** in the **<Exercises>\LabVIEW FPGA\R-Series IO** directory.
 - Right-click **My Computer** in the Project Explorer window and select **New»Targets and Devices** to display the Add Targets and Devices dialog box.
 - Add a PCIe-7852R board as the FPGA Target.
2. Set the project to execute the VI on the development computer with simulated I/O.
 - Right-click the FPGA target and select **Execute VI on» Development Computer with Simulated I/O**.

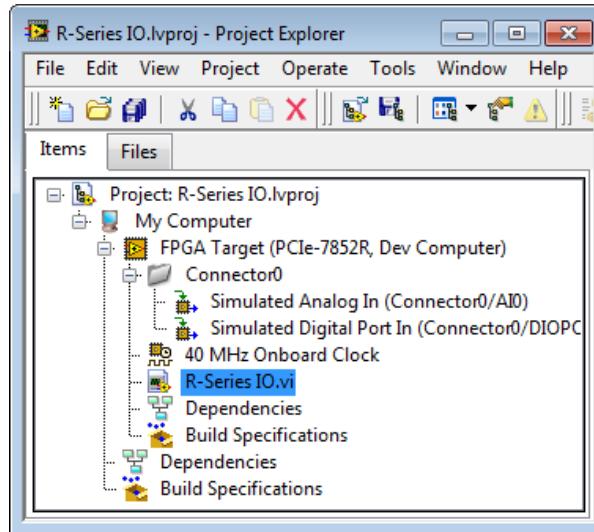


Note For this exercise, we use the default setting **Use Random Data for FPGA I/O Read**. The other option, **Use Custom VI for FPGA I/O**, can be used if you have a separate VI that you would like to use to simulate the I/O.

3. Add AI0 and DIOPORT0 from Connector0 to the FPGA Target.
 - Right-click on the FPGA Target and select **New»FPGA I/O**.
 - In the New FPGA I/O dialog box, expand **Connector0**.
 - Add **AI0** and **DIOPORT0**.
 - Click **OK**. You should now see a Connector0 virtual folder containing two FPGA I/O items in your Project Explorer.
4. Assign unique names to the FPGA I/O resources. The use of unique names for FPGA I/O resources will result in more readable block diagrams.
 - Right-click Connector0/AI0 and select **Rename**. Name the item **Simulated Analog In**.
 - Right-click Connector0/DIOPORT0 and select **Rename**. Name the item **Simulated Digital Port In**.
5. Create a new VI on the FPGA Target.
 - Right-click the target and select **New»VI**.
 - Save the VI as **R-Series IO.vi** in the <Exercises>\LabVIEW FPGA\R-Series IO directory.

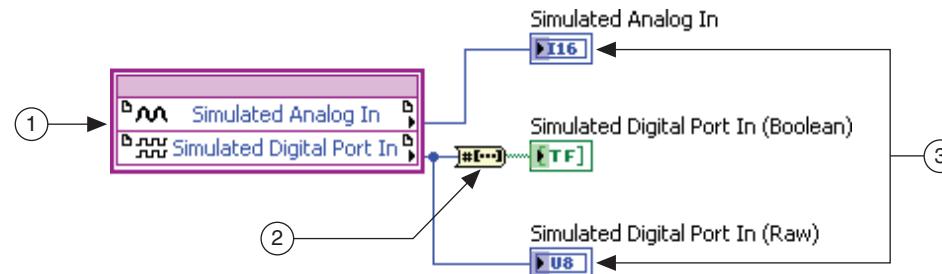
6. Verify that your Project Explorer window resembles Figure 4-1.

Figure 4-1. R-Series I/O Project with FPGA I/O



7. In the R-Series IO VI, build the block diagram shown in Figure 4-2.

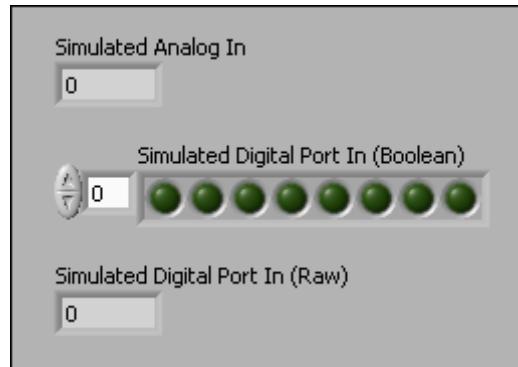
Figure 4-2. R-Series IO.vi Block Diagram



- 1 **FPGA I/O Node**—Drag **Simulated Analog In** from the Project Explorer window and expand the node to display **Simulated Digital Port In**.
- 2 **Number to Boolean Array**—Converts the raw numeric that is returned by the Simulated Digital Port In into a Boolean array. Create an indicator for its output.
- 3 **Indicators**—Right-click the I/O items and create indicators for each.

8. Arrange the front panel shown in Figure 4-3.

Figure 4-3. R-Series I/O.vi Front Panel



- Expand Digital Port In (Boolean) to show all digital lines for the port.
- 9. Save the VI.

Testing

1. Run the VI.

Because the FPGA VI is running on the development computer with simulated I/O, the three indicators contain random data.

Notice that Simulated Digital Port In (Boolean) presents the same data as Simulated Digital Port In (Raw), but in binary form with array index 0 representing the least significant bit.

2. Save and close the VI and project.

End of Exercise 4-1

Exercise 4-2 Acquiring Data From a CompactRIO

Goal

Use I/O in LabVIEW FPGA to acquire analog thermocouple data from two channels of an NI 9211 module. Find the difference in the values returned by these two channels.

Scenario

You must develop an application to access the analog thermocouple data acquired on two channels of the NI 9211 module inserted into slot 1 of your cRIO-9074 chassis. You will then calculate the difference between the data acquired from each channel.

Design

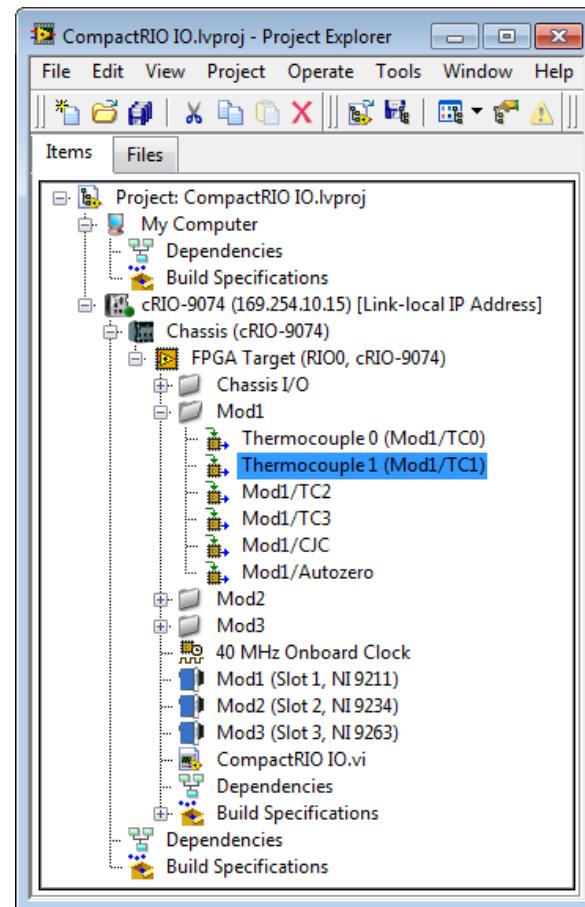
Assign unique names to the I/O channels that you acquire data from to make your block diagram more readable. Name the analog channels Thermocouple 0 and Thermocouple 1. Calculate the difference between the two channels and display that data as Thermocouple Difference.

Implementation

1. Create a new project with a cRIO-9074 as the FPGA Target. You created a similar project in Exercise 2-4.
 - Select **Create Project»Blank Project** from the Getting Started window.
 - Select **File»Save Project** and save the project as **CompactRIO IO.lvproj** in the **<Exercises>\LabVIEW FPGA\CompactRIO IO** directory.
 - Add the cRIO-9074 as the FPGA Target.
 - In the Discover C Series Module? dialog, click **Discover**.
2. Create a new VI on the FPGA Target.
 - Save the VI as **CompactRIO IO.vi** in the **<Exercises>\LabVIEW FPGA\CompactRIO IO** directory.
3. Rename Mod1/TC0 and Mod1/TC1 to Thermocouple 0 and Thermocouple 1, respectively.
 - Expand **Mod 1** and right-click on **Mod1/TC0**. Select **Rename** and enter **Thermocouple 0**.
 - Right-click on **Mod1/TC1**. Select **Rename** and enter **Thermocouple 1**.

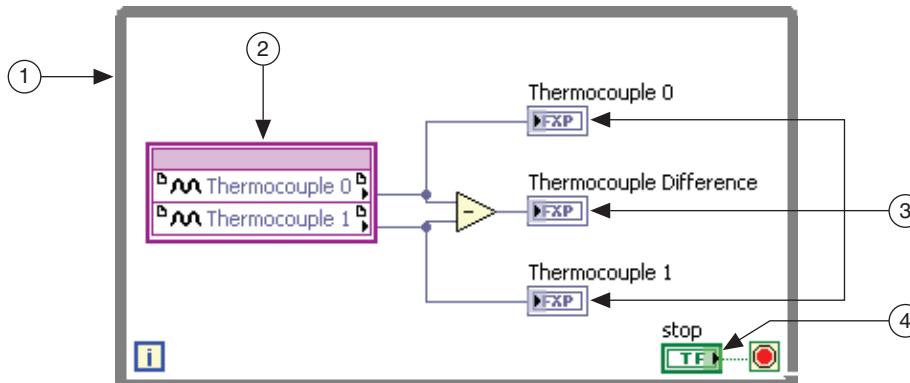
The Project Explorer window should now resemble Figure 4-4.

Figure 4-4. CompactRIO Project with FPGA I/O



4. In the CompactRIO IO VI, build the block diagram shown in Figure 4-5.

Figure 4-5. CompactRIO I/O Block Diagram



- 1 **While Loop**—Create a While Loop.
 - 2 **FPGA I/O Node**—Drag **Thermocouple 0** from the Project Explorer window. Expand the node to display **Thermocouple 1**.
 - 3 **Numeric Indicators**—Right-click the I/O Item outputs and create indicators for each.
 - 4 **Boolean Control**—Create a control to stop the execution of the While Loop.
5. Observe the effect of the Subtract function on the fixed-point data type.

- Open the Context Help and select **Thermocouple 0**, then **Thermocouple 1**. Record the data type for each.

Thermocouple 0 _____

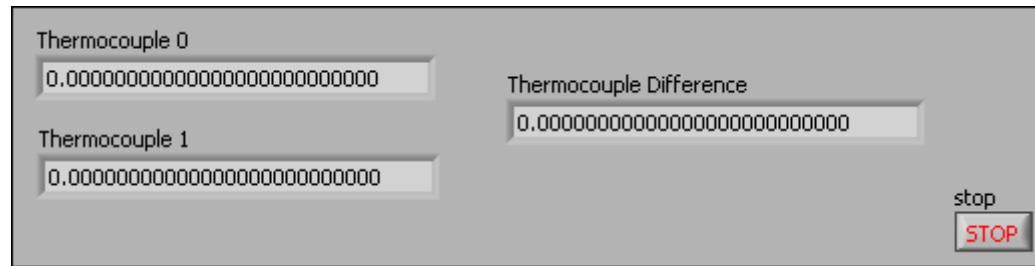
Thermocouple 1 _____

- Select **Thermocouple Difference**. Note that the data type does not match the thermocouple data type. The word length and integer word length have both changed for this indicator. Record the data type.

Thermocouple Difference _____

6. Arrange the front panel window shown in Figure 4-6.

Figure 4-6. CompactRIO IO Front Panel



7. Save the VI.

Testing

1. Compile and run the VI on the FPGA.
2. Touch one of the thermocouples and observe the effect on the Thermocouple Difference.



Note This data is calibrated, but it has not been scaled to Fahrenheit or Centigrade scales. You will perform this conversion later in Exercise 11-2.



Note The indicators update very quickly because the While Loop is executing as quickly as it can. You will learn more about timing your loop execution in Lesson 5, *Timing an FPGA VI*.

3. Click **Stop**.
4. Save and close the VI and project.

End of Exercise 4-2

Timing an FPGA VI

Exercise 5-1 Timing a While Loop

Goal

Use the Loop Timer Express VI to time a While Loop running on an FPGA VI.

Scenario

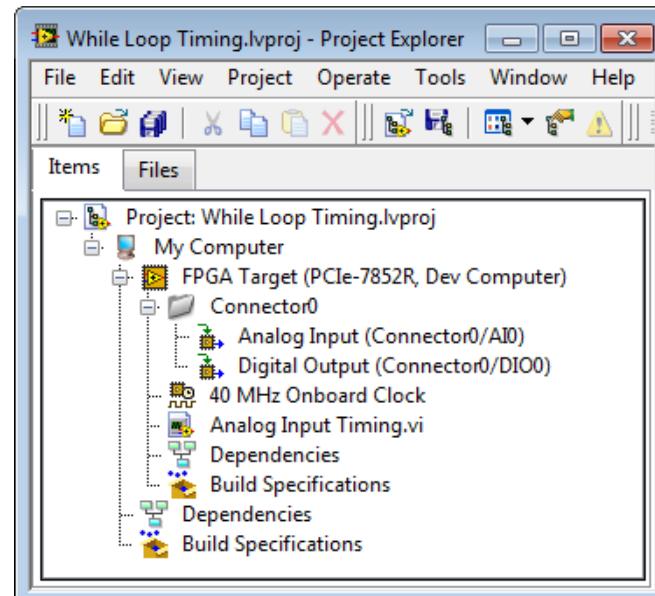
You are assigned the task of creating an FPGA VI that acquires data from the Connector0\AI0 analog input channel of a PCIe-7852R FPGA board. If the data value is higher than the user-specified limit, then the FPGA VI must pass a high voltage on the Connector0\DIO0 digital output line. The FPGA VI must execute these I/O operations at a user-specified sample interval.

Implementation

1. Create a new project with a PCIe-7852R board as the FPGA target.
 - Select **Create Project»Blank Project** from the Getting Started window.
 - Save the project as **While Loop Timing.lvproj** in the **<Exercises>\LabVIEW FPGA\While Loop Timing** directory.
 - Right-click **My Computer** in the Project Explorer window and select **New»Targets and Devices** to display the Add Targets and Devices dialog box.
 - Add a PCIe-7852R board as the FPGA Target.
2. Add AI0 and DIO0 from Connector0 to the FPGA Target.
 - Right-click **FPGA Target** and select **New»FPGA I/O**.
 - In the New FPGA I/O dialog box, expand **Connector0**.

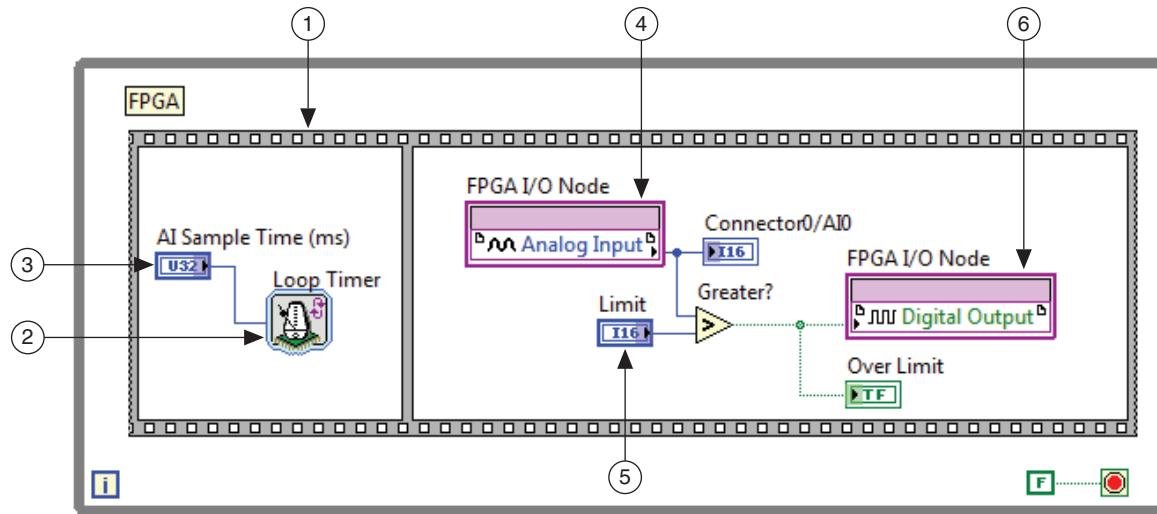
- Add **AI0** and **DIO0**.
 - Click **OK**. You should now see a Connector0 virtual folder containing two FPGA I/O items in the Project Explorer window.
3. Assign unique names to the FPGA I/O resources.
- Right-click Connector0\AI0 and select **Rename**. Rename the item **Analog Input**.
 - Right-click Connector0\DIO0 and select **Rename**. Rename the item **Digital Output**.
4. Create a new VI under the FPGA Target.
- Right-click the target and select **New»VI**.
 - Save the VI as **Analog Input Timing.vi** in the <Exercises>\FPGA\While Loop Timing directory.
5. Verify that your Project Explorer window resembles Figure 5-1.

Figure 5-1. While Loop Timing Project Explorer Window



6. Create the block diagram shown in Figure 5-2 using the following items:

Figure 5-2. Analog Input Timing Block DiagramR-Series IO.vi Block Diagram



- 1 **Flat Sequence Structure**—Right-click and select **Add Frame After**.
 - 2 **Loop Timer Express VI**—Set Counter units to **mSec**. Verify that **Size of internal counter** is set to **32 Bit**.
 - 3 Right-click the **Count (mSec)** input of the Loop Timer Express VI and select **Create»Control**.
 - 4 **Analog Input FPGA I/O Node**—In the Project Explorer window, click **Connector0»Analog Input** and drag it to the block diagram.
 - 5 Right-click the **y** input of the **Greater?** function and set the representation of this numeric to **I16**.
 - 6 **Digital Output FPGA I/O Node**—In the Project Explorer window, click the **Connector0»Digital Output** item and drag it to the block diagram. Right-click the FPGA I/O Node and select **Change to Write**.
7. Save the VI.

Testing

Test the application using simulated I/O values.

1. Configure the project to execute FPGA VIs on the development computer with simulated I/O values.
 - In the Project Explorer, right-click the FPGA target and select **Execute VI on»Development Computer with Simulated I/O**.
2. On the front panel, set **AI Sample Time (ms)** to 500 and **Limit** to 0.
3. Run the VI.

The VI should retrieve values from Connector0/AI0 every 500 ms. The Over Limit indicator should turn on whenever the Connector0/AI0 value is greater than the Limit value.

4. When finished, click **Abort** to stop the application.
5. Save and close the project and VI.

End of Exercise 5-1

Exercise 5-2 Benchmarking a While Loop

Goal

Benchmark the loop period of a While Loop containing code.

Scenario

You are given a pre-built VI containing code in a While Loop. You must benchmark the loop period of the While Loop. You add benchmarking code to the While Loop to display its loop period.



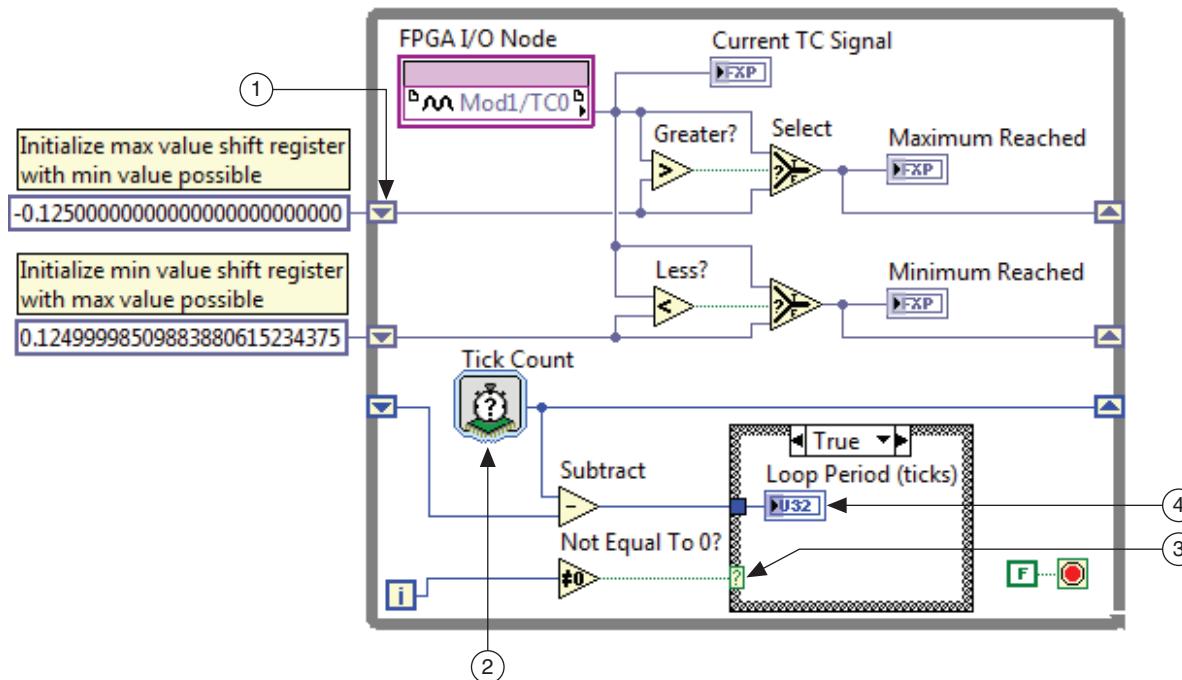
Note For instructions on adapting the exercises in this manual for an alternate cRIO controller, refer to Appendix A, *Alternate CompactRIO Controller Instructions*.

Implementation

1. Open `While Loop Benchmarking.lvproj` in the `<Exercises>\LabVIEW FPGA\While Loop Benchmarking` directory.
2. Configure the CompactRIO target.
 - Right-click the CompactRIO target in the Project Explorer and select **Properties**.
 - In the General category, set the IP Address to the IP Address of your CompactRIO target.
3. Open `TC Max and Min Benchmarking.vi` from the Project Explorer.
4. Examine the block diagram. This VI continuously acquires data from the TC0 channel of the NI 9211 thermocouple module and keeps track of the maximum and minimum values acquired.

5. Modify the block diagram as shown in Figure 5-3 to benchmark the loop iteration period.

Figure 5-3. TC Max and Min Benchmark VI Block Diagram



- 1 **Shift Register**—Right-click the left border of the While Loop and select **Add Shift Register**
- 2 **Tick Count**—Set Counter units to **Ticks**. Verify that **Size of internal counter** is set to **32 Bit**
- 3 **Case Structure**—Wire the output of the **Not Equal to 0?** function to the case selector
- 4 Right-click the tunnel on the Case Structure and select **Create»Indicator**

 **Note** The True case contains the Loop Period (ticks) indicator. The False case remains empty. Because the benchmarking calculation requires two consecutive iterations of the While Loop, this VI does not update the Loop Period (ticks) indicator during the first iteration of the While Loop.

6. Save the VI.

Testing

1. Compile the VI on the FPGA.
 - Run the TC Max and Min Benchmark VI to start the compile process.
2. Test the application.

At the end of compile, the program should download and the FPGA VI should begin to report the current thermocouple signal value, maximum value reached, minimum value reached, and loop period.

- Grip the thermocouple briefly and gently between thumb and forefinger to watch the current thermocouple signal value and the maximum value reached increase.
- Release the thermocouple and watch the current thermocouple signal value decrease and the maximum value reached stay the same.
- When finished, click **Abort** to stop the application.

3. Convert the loop period from ticks to seconds.

- Record the value of the Loop Period (ticks) indicator here:

_____ ticks

- Because this project is using the default 40 MHz Onboard Clock on the FPGA, there are 40,000,000 ticks per second. Convert the loop period to seconds by dividing the number of ticks that you recorded by 40,000,000. Record that value here:

_____ seconds

- Multiply the recorded value of seconds by 1000 to convert seconds to milliseconds. Record that value here:

_____ milliseconds

- Find out how long analog input takes for one channel on the NI 9211.

- Open the *LabVIEW Help* by selecting **Help»LabVIEW Help**.

- Select **NI-RIO»C Series Reference and Procedures»Devices»C Series Module Help»Analog Input Modules»NI 9211** in the Table of Contents.

- In the Hardware Documentation section of the NI 9211 help topic, click on the **NI 9211 Operating Instructions and Specifications** link to open that document.
- Search the *NI 9211 Operating Instructions and Specifications* document to find the conversion time necessary for one channel on the NI 9211. Record that value here:

_____ milliseconds

- Verify that the loop period (ms) is similar to the conversion time (ms) for one channel of the NI 9211.



Note The loop period is close to the conversion time because the most time-consuming node in the loop is the FPGA I/O Node. The execution times of the other functions in the loop are much faster than acquisition time of the FPGA I/O.

4. Save and close the project and VI.

End of Exercise 5-2

Executing Code in Single-Cycle Timed Loops

Exercise 6-1 Comparing a While Loop and a Single-Cycle Timed Loop

Goal

Improve loop execution speeds using a single-cycle Timed Loop. Benchmark the speed of a While Loop and single-cycle Timed Loop running the same code. Observe the difference in speed between the two.

Scenario

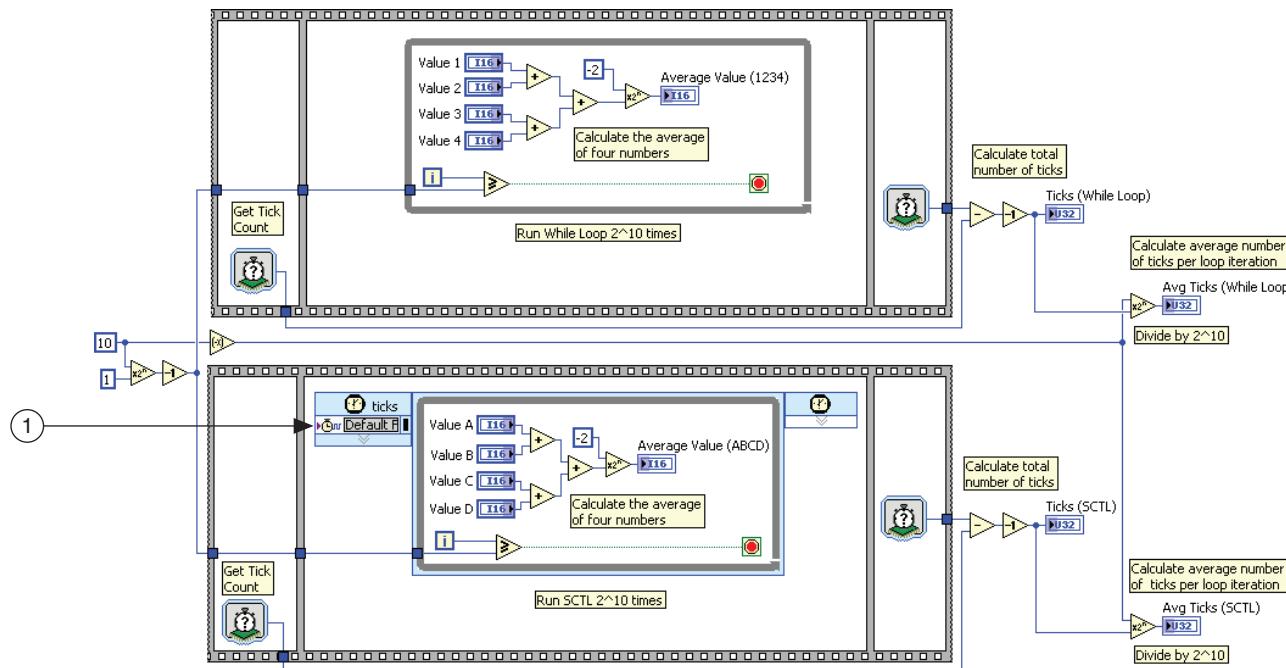
You are given a VI containing code to benchmark the total number of clock ticks it takes for a While Loop to run 1,024 iterations and calculate the average number of clock ticks per iteration. You complete the block diagram to simultaneously obtain the same benchmark and calculation for a single-cycle Timed Loop. You then compare the results.

Implementation

1. Open `While Loop versus SCTL.lvproj` located in the `<Exercises>\LabVIEW FPGA\While Loop versus SCTL` directory.
2. Configure the CompactRIO target.
 - Right-click the CompactRIO target in the Project Explorer and select **Properties**.
 - In the General category, set the IP Address to the IP address of your CompactRIO target.
3. Open `While Loop SCTL Benchmarking.vi` from the Project Explorer. You will finish building this VI in this exercise.
4. Examine the functionality in the block diagram.
 - Notice there are two sequence structures. Because there is no wire or data dependency connecting the two sequence structures to each other, the two sequence structures will run in parallel.
 - Observe the top sequence structure. This sequence structure gets the value of the free running counter in units of ticks before and after the code in the second frame of the sequence structure runs.

- Observe the bottom sequence structure. This sequence structure contains the same functionality as the top sequence structure except that it does not contain a While Loop.
 - Observe the code to the right of each sequence structure. This code calculates the total number of ticks it takes to run the While Loop 1024 times. It also calculates the average number of ticks per loop iteration.
5. Modify the block diagram to benchmark a single-cycle Timed Loop, as shown in Figure 6-1.

Figure 6-1. While Loop SCTL Benchmarking VI Block Diagram



1 Timed Loop—Double-click the Input node of the Timed Loop. Notice that the Timed Loop is configured to use the top-level timing source, which is the 40 MHz Onboard Clock on the FPGA. This means that the Timed Loop is configured to run at a rate of 40 MHz.

6. Save the VI.

Testing

1. Set the front panel controls to the values that you want the VI to average.
2. Compile the VI on the FPGA.
 - Run the While Loop SCTL Benchmarking VI to start the compile process.
3. When the VI finishes running, verify that the values in the Average Value (1234) indicator and Average Value (ABCD) indicator are correct.
4. Compare the Ticks (While Loop) and Ticks (SCTL) indicators.

The Ticks (While Loop) indicator shows the number of FPGA clock ticks required to run the While Loop 1024 times. The Ticks (SCTL) indicator shows the number of FPGA clock ticks required to run the single-cycle Timed Loop 1024 times.

5. Compare the Avg Ticks (While Loop) and Avg Ticks (SCTL) indicators.

The Avg Ticks (While Loop) indicator shows the average number of FPGA clock ticks required to run each iteration of the While Loop. The Avg Ticks (SCTL) indicator shows the number of FPGA clock ticks required to run each iteration of the single-cycle Timed Loop.

6. Notice that the While Loop requires about 7 FPGA clock ticks to run each iteration. The single-cycle Timed Loop is much faster and only requires 1 FPGA clock tick to run each iteration.

 **Note** All operations in a single-cycle Timed Loop must be able to complete within one cycle of the FPGA clock.

End of Exercise 6-1

Exercise 6-2 Fixing SCTL Errors

Goal

Examine and fix errors in a single-cycle Timed Loop caused by unsupported objects and clock rates.

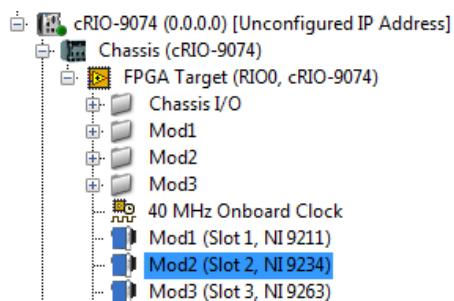
Scenario

You inherited an FPGA VI that contains code in a single-cycle Timed Loop. When you try to compile the VI in this exercise, you run into errors caused by unsupported objects and clock rates. You will use the Code Generation, Compilation Status, and Timing Violation Analysis windows to examine and fix the errors.

Implementation

1. Open Investigate SCTL Errors.lvproj located in the <Exercises>\LabVIEW FPGA\Investigate Timing Errors folder.
2. Configure the CompactRIO target.
 - Right-click the CompactRIO target in the Project Explorer and select **Properties**.
 - In the General category, set the IP address to the IP Address of your CompactRIO target.
3. Notice that the CompactRIO target in the Project Explorer uses a NI 9234 module, as shown in Figure 6-2.

Figure 6-2. NI 9234 Module in Project Explorer



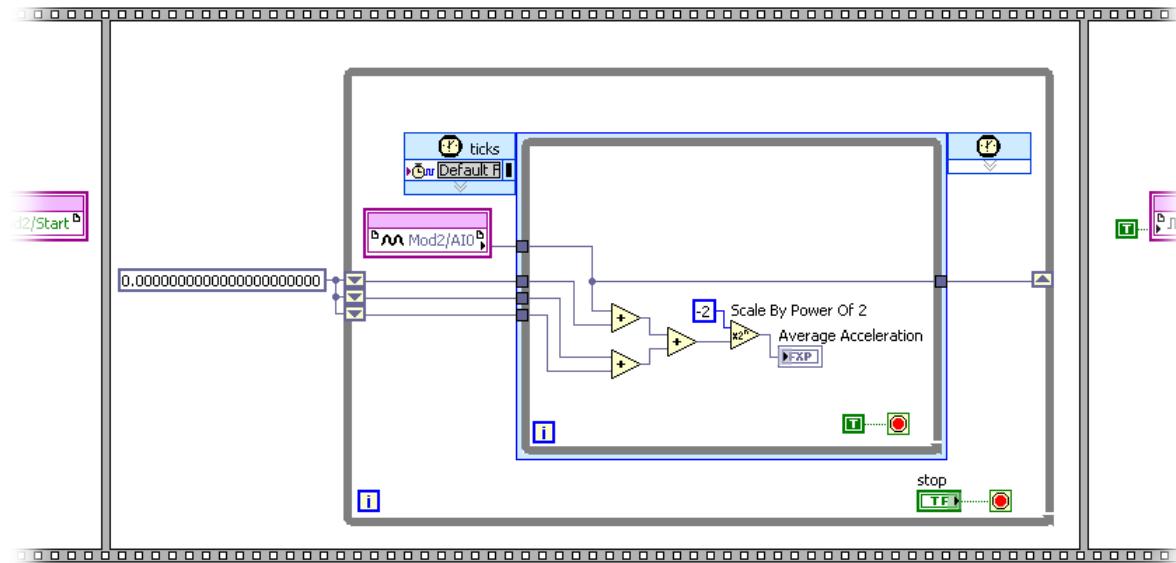
If you are using a NI 9233 module instead of a NI 9234, complete this step to modify the project:

- In the Project Explorer window, right-click the **Mod2 (Slot 2, NI 9234)** module item and select **Remove from Project**.
- Right-click the **cRIO-9074»Chassis»FPGA Target** item and select **New»C Series Modules**.

- In the Add Targets and Devices on FPGA Target dialog box, select **New target or device**. Select **C Series Module** and click **OK**.
 - Set Name to **Mod2**, Type to **NI 9233**, and Location to **Slot 2**, and click **OK**.
4. From the Project Explorer, open `Investigate SCTL Errors.vi`.
5. Examine the block diagram.
- Notice that the first frame of the Sequence structure starts the analog input acquisition on the NI 9234.
 - Notice that the second frame of the Sequence structure contains a single-cycle Timed Loop that contains code to acquire from an analog input channel on the NI 9234 and calculates the average of the last four values.
- What functions do you see in the Timed Loop that are not supported in a single-cycle Timed Loop?
- Notice that the Timed Loop will stop if the Stop control is TRUE.
 - Notice that the last frame of the Sequence structure stops the analog input acquisition on the NI 9234.
6. Run the VI to start the compilation process. Because the single-cycle Timed Loop contains unsupported objects, LabVIEW will pop up a Code Generation Errors window instead of compiling the VI.
7. Examine the Code Generation Errors window.
- Notice that the Divide function and the FPGA I/O Node: Mod2/AI0 (Read) are not supported inside the single-cycle Timed Loop.
-  **Note** Some FPGA I/O Nodes are supported in the single-cycle Timed Loop.
- Select each unsupported object and read the Details section.
 - Select each unsupported object and click **Show Error** to see where the object is on the block diagram.
 - Close the Code Generation Errors window when finished.

8. Modify the block diagram as shown in Figure 6-3 to remove the unsupported objects from the single-cycle Timed Loop by modifying the following items:

Figure 6-3. Investigate SCTL Errors VI Block Diagram



- Divide function—Delete the unsupported Divide function.
- Average Acceleration indicator—Right-click the indicator and select **Adapt to Source**.
- Scale By Power Of 2 function—Right-click the **n** input and select **Create»Constant**. Set the constant to **-2**. This accomplishes the division by **4** functionality using a function that is supported in the single-cycle Timed Loop.
- FPGA I/O Node—Move this object outside the single-cycle Timed Loop because it is not supported in the single-cycle Timed Loop.
- Stop control—Move this object outside of the single-cycle Timed Loop.
- Timed Loop—Right-click the loop condition terminal and select **Create Constant**. Set the constant to **TRUE**.



Note This configures the single-cycle Timed Loop to only run one iteration when it is called.

- While Loop—Use the While Loop to implement the looping functionality of the code. Wire the Stop control to the loop iteration terminal of the While Loop.
- Shift registers—Delete the shift register on the single-cycle Timed Loop. Create and wire the shift register on the While loop as shown in Figure 6-3. Expand the shift register to display three elements.

9. Save the VI.

10. Increase the FPGA clock rate.

- Add a 200 MHz derived clock.
 - Right-click the **cRIO-9074»Chassis»FPGA Target»40 MHz Onboard Clock** item in the Project Explorer and select **New FPGA Derived Clock**.
 - Set Desired Derived Frequency to 200 MHz.
 - Click **OK**.



Note Create derived clocks to create clocks with frequencies other than the base clock frequency.

11. Set the single-cycle Timed Loop to use the 200 MHz derived clock.

- Double-click the Input node of the single-cycle Timed Loop
- Select **Select Timing Source**.
- Select **200MHz** in the Available Timing Sources section.
- Click **OK**.

12. Save the VI.

13. Run the VI to start the compilation process.

14. Investigate the timing violations that caused the compilation process to fail.

- When the compile fails, set Reports to **Summary** in the Compilation Status window. Notice that the compilation failed due to timing violations.
- Click **Investigate Timing Violations**.

- Examine the Timing Violation Analysis window which displays the objects in the Timed Loop.
 - Notice the time requirement of 5.00 ns. Because all operations in the single-cycle Timed Loop must execute within one FPGA clock cycle, the operations must execute within less than 5 ns (1/200,000,000 seconds), which is the FPGA clock period of the 200 MHz derived clock.
 - You can see the total delay required for each path and item.

What is the total delay of Path 1? _____ ns

Notice that the total delay of Path 1 exceeds the time requirement of 5.00 ns.



Note The total delay is the sum of the logic delay and routing delay. The logic delay indicates the amount of time in nanoseconds that a logic function takes to execute. The routing delay indicates the amount of time in nanoseconds that a signal takes to traverse between FPGA logic blocks.

- Select the Timed Loop object and click **Show Element** to display the path on the block diagram.
- Close the Timing Violation Analysis window when finished.

15. Fix the timing error by modifying the derived FPGA clock to an appropriate rate.

- Add a 80 MHz derived clock.
 - Right-click the **cRIO-9074»Chassis»FPGA Target»40 MHz Onboard Clock** item in the Project Explorer and select **New FPGA Derived Clock**.
 - Set Desired Derived Frequency to 80 MHz.
 - Click **OK**.
- Double-click the Input Node of the Timed Loop
- Select **Select Timing Source**.
- Select **80MHz** in the Available Timing Sources section.
- Click **OK**.

16. Save the VI and the project.

Testing



Note Verify the NI 9234 is in slot 2 of the NI cRIO-9074.

1. Compile the VI on the FPGA.
 - Run the Investigate SCTL Errors VI to start the compile process.
2. Test the application.

At the end of compile, the program should download and the FPGA VI should begin to report an averaged acceleration signal value from the Y Acceleration output of the Sound and Vibration Signal Simulator box.

- Switch the Fan Speed Control switch to DIAL.
 - Twist the Fan Speed Control dial to vary the Mod2/AI0 signal.
 - When finished, click **Stop** to stop the application.
3. Close the VI and the project.

End of Exercise 6-2

Exercise 6-3 Implementing Pipelining

Goal

Speed up the path inside a single-cycle Timed Loop by reducing the path length using pipelining.

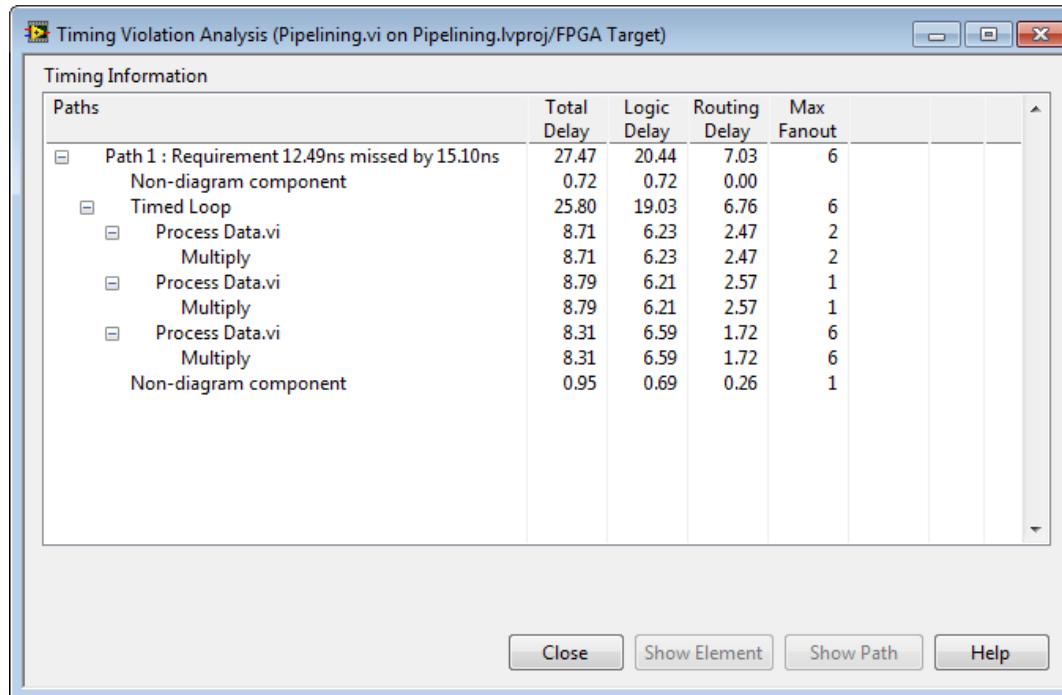
Scenario

You inherited an FPGA VI that contains code in a single-cycle Timed Loop that needs to execute using an 80 MHz derived clock. However, the code currently cannot finish executing within one FPGA clock cycle of the 80 MHz clock. You use pipelining to speed up the code so that it can run using the 80 MHz clock.

Implementation

1. Open Pipelining.lvproj located in the <Exercises>\LabVIEW FPGA\Pipelining folder.
2. Configure the CompactRIO target.
 - Right-click the CompactRIO target in the Project Explorer window and select **Properties**.
 - In the General category, set the IP Address to the IP Address of your CompactRIO target.
3. From the Project Explorer, open Pipelining.vi.
4. Examine the block diagram.
 - Notice that the single-cycle Timed Loop is using a 80 MHz derived clock. Because all operations in the single-cycle Timed Loop must execute within one FPGA clock cycle, the operations must execute within less than 12.50 ns (1/80,000,000 seconds), which is the FPGA clock period of a 80 MHz derived clock. If you compile this VI, the compile fails due to timing violations and you would investigate the Timing Violation Analysis window, as shown in Figure 6-4.

Figure 6-4. Timing Violation Analysis window

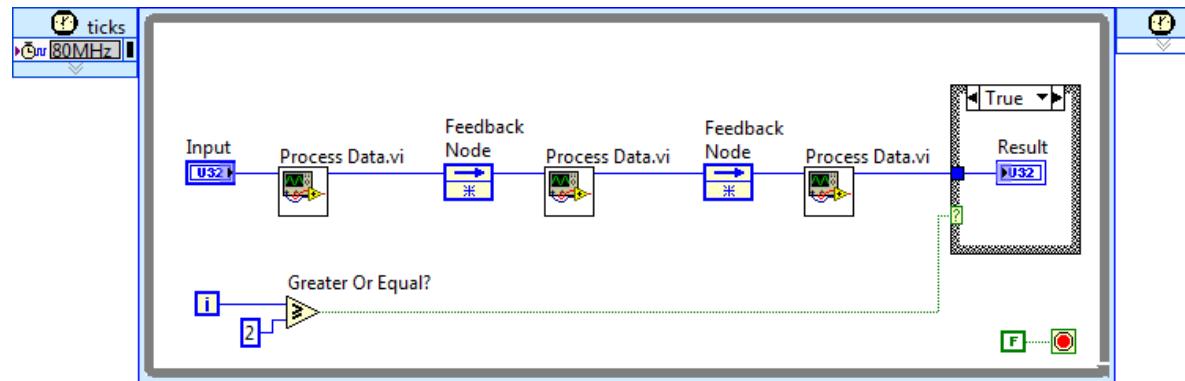


5. Examine the Timing Violation Analysis window in Figure 6-4.

- What is the total delay of Path 1? _____ ns.
- Notice that the total delay of Path 1 exceeds the time requirement of 12.49 ns.
- How can you use pipelining to reduce the total delay of Path 1 to less than or equal to 12.49 ns?

6. Modify the block diagram, as shown in Figure 6-5, to speed up the path inside the single-cycle Timed Loop by reducing the path length using the pipelining technique.

Figure 6-5. Pipelining VI Block Diagram



- Feedback Node—Delete the wires between the Process Data subVIs. Place two Feedback Nodes on the block diagram. Right-click each Feedback node and select **Change Direction**.
- Greater Or Equal? function—Because this pipeline has three steps, the first valid output of the final step does not occur until the third iteration of the single-cycle Timed Loop. Therefore, use the Greater Or Equal? function to indicate whether the output of the final step is valid.



Note When you implement a pipeline, the output of the final step lags behind the input by the number of steps in the pipeline, and the output is invalid for each clock cycle until the pipeline fills. The number of steps in a pipeline is called the pipeline depth, and the latency of a pipeline, measured in clock cycles, corresponds to its depth. For a pipeline of depth N , the result is invalid until the N th clock cycle, and the output of each valid clock cycle lags behind the input by $N-1$ clock cycles.

- Case structure—Place the Result indicator in the True case. Leave the False case empty.
- Wire the block diagram, as shown in Figure 6-5.

7. Save the VI and the project.

Testing

1. Compile the VI on the FPGA.
 - Run the Pipelining VI to start the compile process.
2. Verify that the FPGA VI now compiles successfully due to your pipelining optimization.
3. When finished, abort the execution of the FPGA VI.
4. Close the VI and the project.

End of Exercise 6-3

Signal Processing

Exercise 7-1 Using the Fixed-Point Data Type

Goal

Use the fixed-point data type and configure rounding and overflow modes if necessary.

Scenario

You must read values from a CompactRIO analog input channel that returns fixed-point data. You must implement fixed-point math on this data. You also know that the analog input channel is expected to return values between –2 and 2.

In this exercise, the fixed-point math algorithm scales the data by a constant value of 5. However, other fixed-point algorithms can involve many more operations and grow the number of bits used by the fixed-point data type to be larger.

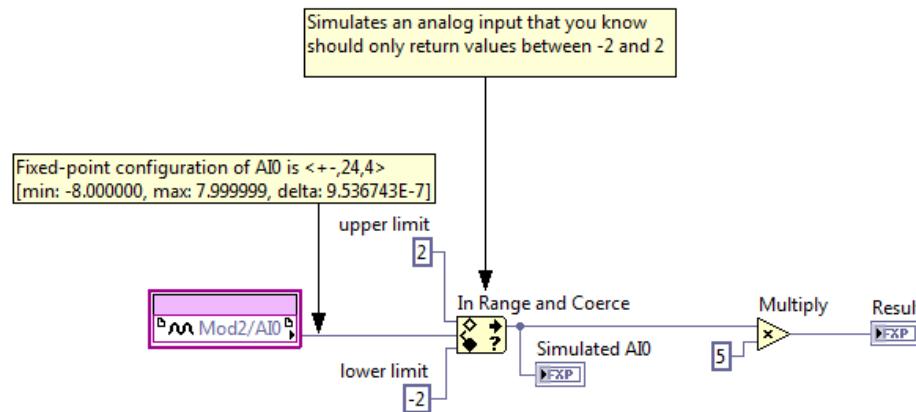
Implementation

Fixed-Point Configuration

1. Open `Fixed-Point.lvproj` located in the `<Exercises>\LabVIEW FPGA\Fixed-Point` directory.
2. In the Project Explorer window, **cRIO-9074»Chassis»FPGA Target** is set to execute FPGA VIs on the development computer with simulated I/O.
3. Open `Fixed-Point Configuration.vi` from the Project Explorer.
4. Examine the block diagram.
 - Show the Context Help window by pressing `<CTRL-H>`. Hover your mouse pointer over the fixed-point wires to see their fixed-point configuration.
 - Notice the In Range and Coerce function. This function coerces the FPGA I/O Node output to simulate an analog input channel that returns values between –2 and 2.
5. Run the VI and notice that the Simulated AI0 indicator will always return a value between –2 and 2.

6. Modify the block diagram, as shown in Figure 7-1, to scale the simulated AI0 value by 5.

Figure 7-1. Fixed-Point Configuration VI Block Diagram



- Multiply function.
 - Wire the x input first.
 - Right-click the y input and select **Create»Constant**.
- Configure the fixed-point constant to represent a value of 5.
 - Right-click the constant and select **Properties**.
 - On the Data Type tab, notice that the Word Length is 24 bits. However, you do not need 24 bits to represent a constant value of 5, so you should modify the fixed-point configuration to use the minimum number of bits required to represent a value of 5.
 - Set the values of the Fixed-Point Configuration section to the following values:
Sign Encoding: Unsigned
Word length: 3 bits
Integer word length: 3 bits



Note This fixed-point configuration corresponds to a minimum of 0, maximum of 7, and a delta of 1. This range and delta can represent a value of 5.

- Click **OK**.
 - On the block diagram, set the constant to 5.
- Create the Result indicator and observe its fixed-point configuration.
- Right-click the output of the Multiply function and select **Create»Indicator**. Rename the indicator as **Result**.
 - Show the Context Help by pressing <Ctrl-H>. Hover your mouse pointer over the input and output wires of the Multiply function and the Result indicator, and observe their fixed-point configurations in the Context Help window.
 - Notice that the Multiply function automatically grew the size of its output fixed-point configuration based on the fixed-point configuration of its inputs.

The x input has a fixed-point configuration of <+,-24,4> and the y input has a fixed-point configuration of <+,3,3>. The Multiply function automatically grew its fixed-point configuration to <+,-27,7>.

7. Run the VI several times and verify that the Result is equal to Simulated AI0 multiplied by 5.
8. At this point, you would normally be ready to compile your fixed-point algorithm on the FPGA. In the previous steps, you allowed the Multiply function to automatically grow the size of the fixed-point configuration, which helps avoid data loss. This is the safest method to use when creating a fixed-point algorithm. However, letting functions automatically grow the size of the fixed-point configuration uses up more FPGA resources and might cause a compilation to fail. If your compilation fails due to lack of FPGA resources, then try to modify the fixed-point configurations in your algorithm to use less space.

Optimizing Fixed-Point Configuration

1. Identify a place where you can optimize the fixed-point configuration.

□ Notice the inputs of the Multiply function.

 - Expected minimum of x input: -2
 - Expected maximum of x input: 2
 - Delta of x input: 9.536743E-7
 - Constant value of y input: 5

- Because the y input is a constant of 5, we can predict the expected range of the Multiply function output.
 - Expected minimum of output (-2×5): -10
 - Expected maximum of output (2×5): 10
- 2. Configure the Multiply function to use the minimum number of bits to represent the expected output.
 - Right-click the Multiply function and select **Properties**.
 - On the Output Configuration tab, disable the Adapt to source checkbox.
 - Decrease the Integer Word Length to the smallest number of bits that can still represent the expected minimum and maximum output values of -10 and 10.
 - Decrease the Word Length to the smallest number of bits that can still represent the original delta of 9.536743E-7.
 - Verify that the configuration is set to the following values:
 - **Sign encoding:** Signed
 - **Word length:** 25 bits
 - **Integer word length:** 5 bits
 - **Rounding mode:** Round Half-Even
 - **Overflow mode:** Saturate
 - **Include overflow status** checkbox: Enabled
 - Click **OK**.
 - Notice that the Multiply function has a blue coercion dot on its output terminal. The blue coercion dot indicates that you have set the fixed-point configuration of the output differently from the default.

3. Modify the Result indicator to update its fixed-point configuration.
 - Right-click the Result indicator and select **Properties**.
 - On the Data Type tab, enable the **Adapt to source** checkbox. Now the Result indicator will adapt to the fixed-point configuration of its input wire.
 - Click **OK**.
4. Run the VI several times and verify that the Result is still equal to Simulated AI0 multiplied by 5. Result should still be accurate even though we have decreased the number of bits used by the fixed-point data type.
5. Save the VI.
6. Observe overflow behavior.
 - On the block diagram, change the constants wired into the In Range and Coerce function from -2 and 2 to -7 and 7.

The previous optimization you implemented was based on an expected Simulated AI0 value between 2 and -2. By increasing the range of Simulated AI0, we are forcing an overflow condition.

 - Go to the front panel.
 - Right-click the Result indicator and select **Visible Items»Overflow Status LED**. You should now see an LED on the indicator that is lit whenever an overflow condition occurs.
 - Run the VI until you see the Overflow Status LED illuminate.
7. Save and close the VI when finished.

End of Exercise 7-1

Exercise 7-2 Using the Four-Wire Handshaking Protocol

Goal

Implement a fast throughput rate for the Scaled Window and FFT Express VIs by using a single-cycle Timed Loop and the four-wire handshaking protocol.

Scenario

You must implement a signal processing algorithm that performs a scaled window and FFT at a rate of 8 MHz or faster. A While Loop containing the Scaled Window and FFT Express VIs will not be able to execute fast enough to meet this requirement. You must implement the Scaled Window and FFT Express VIs in a single-cycle Timed Loop, which will require using a four-wire handshaking protocol.

Implementation

1. Open 4-wire Protocol.lvproj located in the <Exercises>\LabVIEW FPGA\4-wire Protocol directory.
2. Open 4-wire Protocol.vi from the Project Explorer.
3. Examine the block diagram.

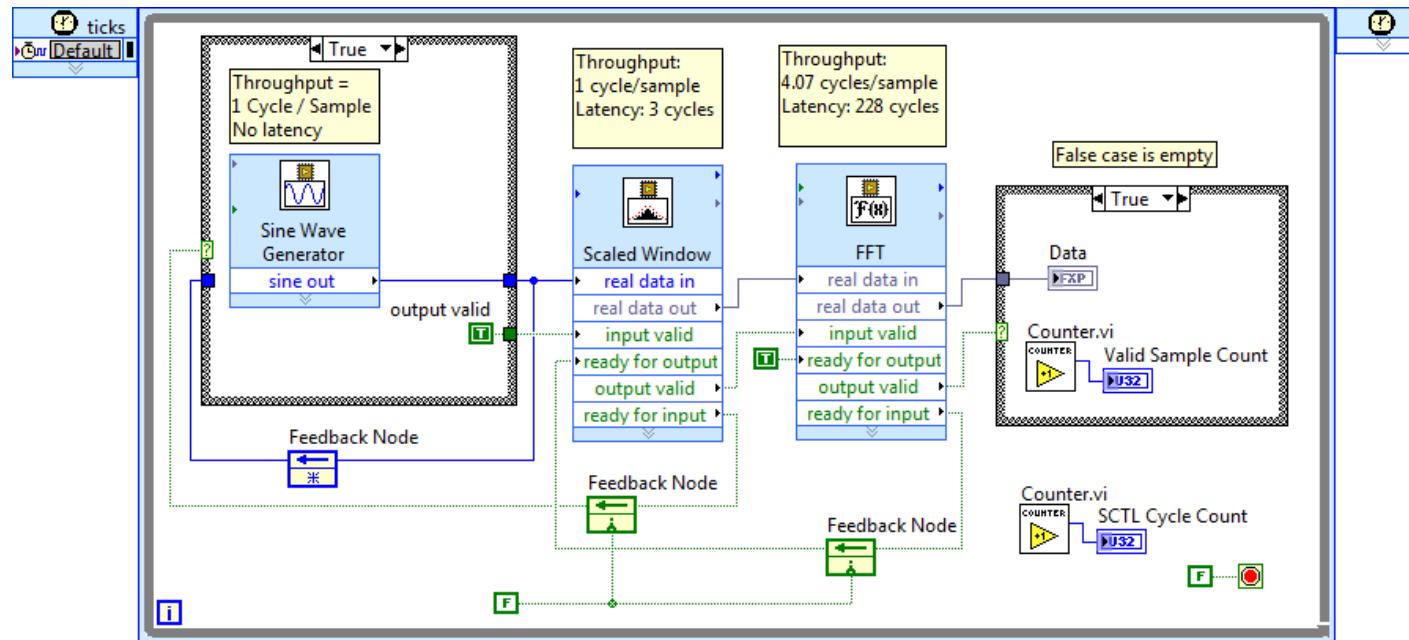
The code in the Case structure simulates a signal. You add Express VIs to process this signal in the next step.



Note The Sine Wave Generator Express VI is not supported in the single-cycle Timed Loop when compiled to run on the FPGA. The Sine Wave Generator Express VI in the Case structure is used to simulate a signal in this exercise because you will only be simulating the execution of this VI on the development computer.

4. Modify the block diagram, as shown in Figure 7-2, to implement the handshaking protocol used by certain nodes inside a single-cycle Timed Loop.

Figure 7-2. 4-Wire Protocol VI Block Diagram



- Scaled Window Express VI—In the configuration window, select **Inside single-cycle Timed Loop** in the Execution Mode section. Leave all other settings at their default values. Notice the Throughput and Latency values. On the block diagram, right-click the Express VI and select **Icon Style»View as Express VI** and arrange the terminals to match the appearance in Figure 7-2.
- FFT Express VI—In the configuration window, select **Inside single-cycle Timed Loop** in the Execution Mode section. Leave all other settings at their default values. Notice the Throughput and Latency values. On the block diagram, right-click the Express VI and select **Icon Style»View as Express VI** and arrange the terminals to match the appearance in Figure 7-2.



Note For most VIs with a throughput setting, the throughput specifies the number of iterations between when the VI can accept a new input. The FFT Express VI throughput behaves slightly different because the FFT Express VI must process frames of n points, where n is defined by the Length parameter in the configuration window of the Express VI. Because of this, the FFT Express VI accepts n new inputs in consecutive iterations and returns the corresponding n outputs in consecutive iterations after the number of iterations determined by latency. The throughput indicates how often the FFT Express VI can accept a new frame of n points.

For example, consider an FFT Express VI configured with a length of 64 and throughput of 4.07 cycles/sample. Assume that the input valid terminal is always True. If this FFT Express VI starts consuming a frame in iteration 1, it will consume 64 valid inputs from iteration 1 to 64. It will then wait 196 iterations before it can consume the next frame starting in iteration 261. This results in an overall throughput of approximately 4.07 cycles/sample ((196+64 cycles)/64 valid inputs).

- Feedback Nodes—Use these nodes to connect the ready for input terminal of a node to the corresponding boolean input terminal of the previous node. Multi-cycle nodes do not return valid data every clock cycle. To ensure the numerical accuracy of an algorithm, nodes that depend on this data must know whether the data is invalid or valid.
- Case structure—The code in the True case only executes when the output from the FFT Express VI is valid. Leave the False case empty.
- Counter subVI—This subVI keeps a running count of the number of times it is called. This instance is used to count how many times the FFT Express VI returns a valid output.

Drag **SubVIs»Counter.vi** from the Project Explorer to the True case of the Case structure. Right-click the output terminal and select **Create»Indicator**. Rename the indicator as **Valid Sample Count**.

- Counter subVI—This subVI keeps a running count of the number of times it is called. This instance is used to count the number of iterations the single-cycle Timed Loop executes.

Drag **SubVIs»Counter.vi** from the Project Explorer to the block diagram. Right-click the output terminal and select **Create»Indicator**. Rename the indicator as **SCTL Cycle Count**.

5. What is the throughput rate for this series of nodes? _____ cycles/sample.



Note To determine the fastest throughput rate for a series of nodes, first determine the throughput rate of each node by looking at the Throughput control for that node. You can see this control by double-clicking a node or by displaying the Context Help window of the node. The Throughput control with the highest value (that is, the slowest node) is the fastest rate (that is, the fewest number of cycles) at which the system can send data to this series of functions.

Therefore in this exercise, the fastest throughput rate for the series of nodes is 4.07 cycles/sample.

6. What is the latency of this series of nodes? _____ cycles.



Note To determine the total latency for a series of nodes, add up the latencies of every node in the series.

Therefore in this exercise, the total latency is 231 cycles (3 cycles + 228 cycles). This means after the Scaled Window Express VI is ready for and receives its first valid input, the FFT Express VI returns the corresponding output 231 cycles later.

7. Save the VI.

Testing

1. Go to the front panel.
2. Run the VI for a few seconds and then click the Abort button to stop the VI.
3. Examine the results in the Output Valid Count and SCTL Count indicators.
4. Calculate how many cycles of the single-cycle Timed Loop executed per every valid output from the series of nodes.

SCTL Cycle Count divided by Valid Sample Count = _____ cycles/sample

 **Note** This value should be close to your answer to Step 5 in the *Implementation* section.

5. Calculate the throughput of the algorithm in samples per second.

40 MHz SCTL clock divided by <answer to Step 4> = _____ MHz.

 **Note** The answer should be approximately 9.83 MHz (40 MHz divided by 4.07 cycles/sample). This meets the 8 MHz or faster requirement described in the *Scenario* section.

6. Save and close the VI and project when finished.

End of Exercise 7-2

Sharing Data on FPGA

Exercise 8-1 Transferring Buffered Data on the FPGA

Goal

Modify a VI on the FPGA with parallel loops that pass data using an FPGA FIFO.

Scenario

You are given a partially completed VI with three parallel While Loops. Your task is to modify the VI so that it acquires data from the NI 9234 in slot 2 of the cRIO-9074 controller. This data should be processed in a parallel loop so that the acquisition can proceed as quickly as possible. If an overflow occurs, the VI should terminate.

In the processing loop, compare the accelerometer data to a user-defined threshold value. If the accelerometer data exceeds that threshold, then turn on the FPGA LED on the 9074 controller. Use a target-scoped FIFO to pass data from the acquisition loop to the processing loop without losing any data points.

Design

You must create a target-scoped FIFO in order to pass data between loops on the FPGA without losing any data. Modify the acquisition loop so that data is acquired and passed into this FIFO. If an overflow occurs, stop execution of all three loops. The user should have the ability to manually stop execution of this VI as well. When the VI completes execution, clear the contents of the FIFO.

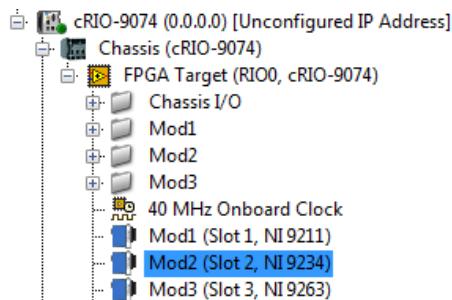
The processing loop should read data from the FIFO and compare that data to the user-defined threshold. If the data exceeds that value, use an FPGA I/O node to write a TRUE value to the FPGA LED controller I/O.

Implementation

1. Open Accelerometer FIFO.lvproj in the <Exercises>\LabVIEW FPGA\Accelerometer Threshold directory.
2. Configure the CompactRIO target.
 - Right-click the CompactRIO target in the Project Explorer and select **Properties**.
 - In the General category, set the IP Address to the IP Address of your CompactRIO target.

3. Notice that the CompactRIO target in the Project Explorer uses a NI 9234 module, as shown in Figure 8-1.

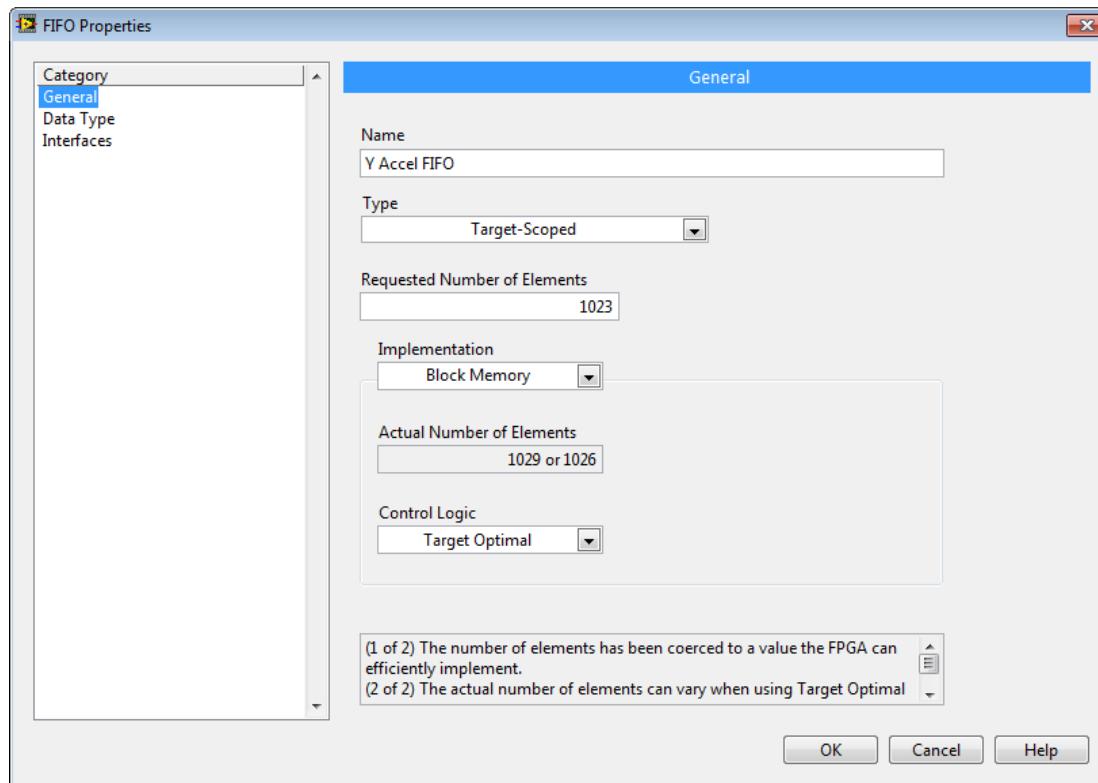
Figure 8-1. NI 9234 Module in Project Explorer



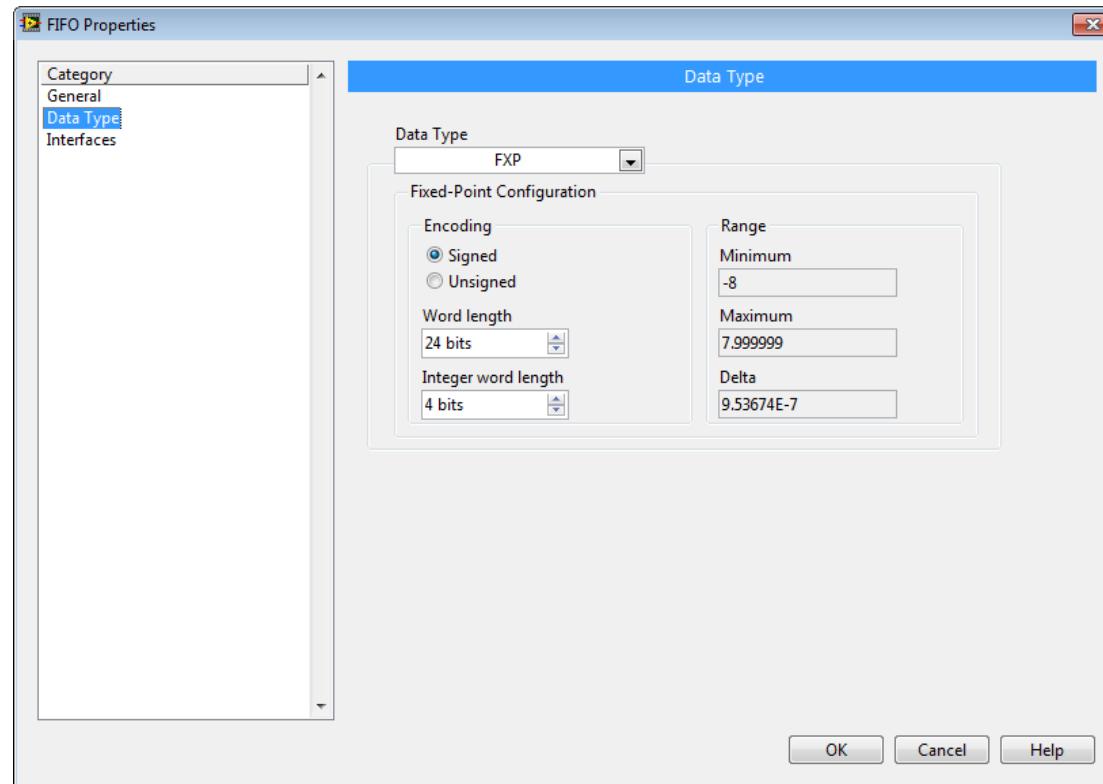
If you are using a NI 9233 module instead of a NI 9234, complete this step to modify the project:

- In the Project Explorer window, right-click the **Mod2 (Slot 2, NI 9234)** module item and select **Remove from Project**.
 - Right-click the **cRIO-9074»Chassis»FPGA Target** item and select **New»C Series Modules**.
 - In the Add Targets and Devices on FPGA Target dialog box, select **New target or device**. Select **C Series Module** and click **OK**.
 - Set Name to **Mod2**, Type to **NI 9233**, and Location to **Slot 2**, and click **OK**.
4. Rename Mod2/AI0 as **Y Accel**.
5. Create a target-scoped FIFO named **Y Accel FIFO**. This FIFO passes **Y Accel** data between loops of your FPGA VI.
- Right-click the **FPGA Target**. Select **New»FIFO**.
 - In the **FPGA FIFO Properties** dialog box, select **General** from the Category list and configure the FIFO as shown in Figure 8-2.

Figure 8-2. Y Accel FIFO Properties—General Category



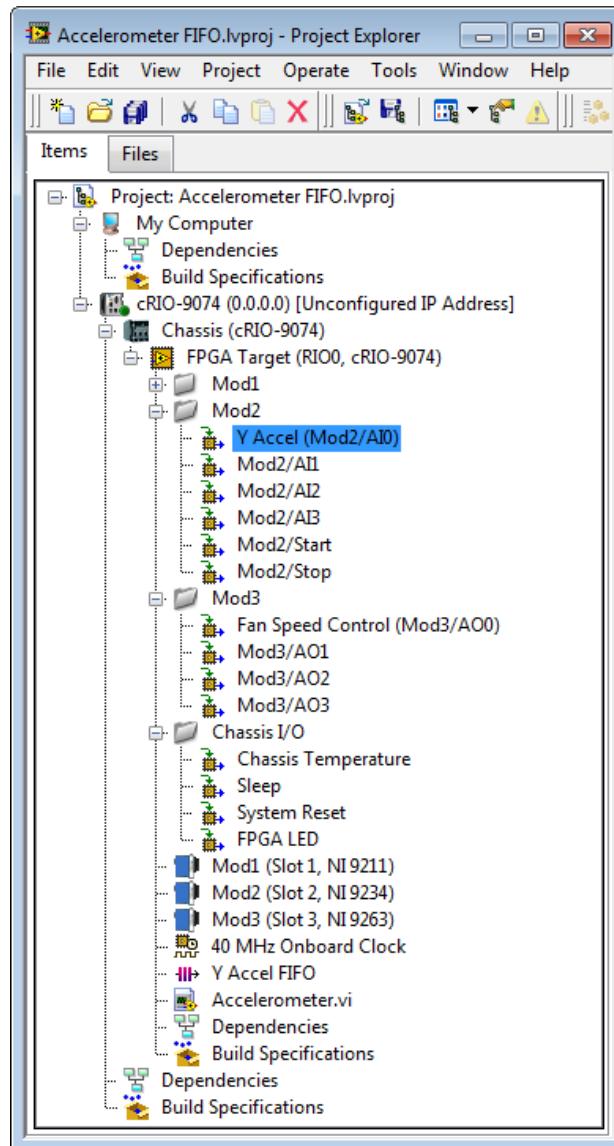
- In the FPGA FIFO Properties dialog box, select **Data Types** from the Category list and configure the FIFO as shown in Figure 8-3.

Figure 8-3. Y Accel FIFO Properties—Data Type Category

Note In the Data Type category, the values shown in Fixed-Point Configuration are set to match the range of the data generated by the NI 9234. This will help to avoid coercion of data.

- Click **OK**. The project should now resemble Figure 8-4.

Figure 8-4. Accelerometer FIFO Project

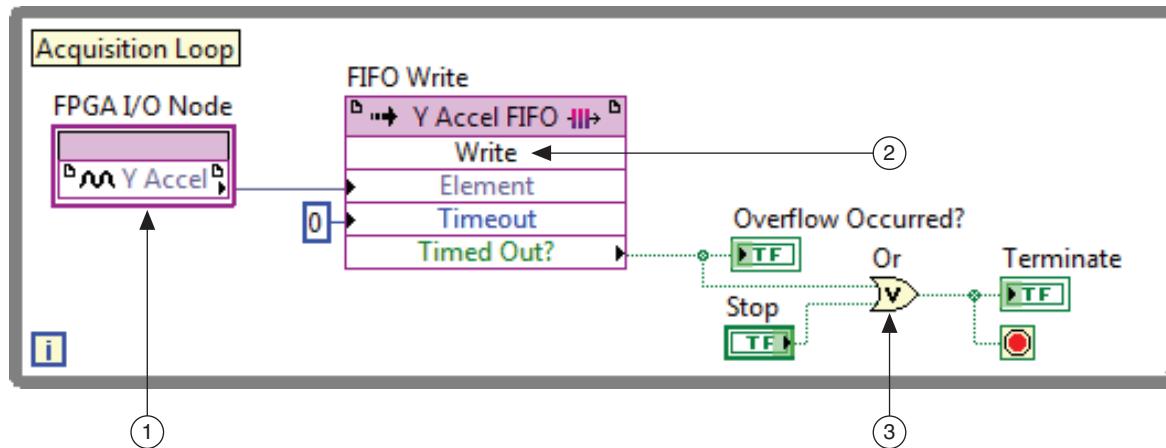


6. Open Accelerometer.vi from the Project Explorer window. View the block diagram and observe the following three loops contained in the unfinished FPGA VI:

- Acquisition Loop—This loop handles acquisition of the Y axis acceleration from the NI 9234 and writes the data to a target-scoped FIFO.
- Processing Loop—This loop reads data from the FIFO and compares the result to a threshold value.
- Fan Speed Control Loop—This loop controls the speed of the fan on the Sound and Vibration Signal Simulator.

7. Modify the Acquisition Loop, as shown in Figure 8-5.

Figure 8-5. Accelerometer VI—Acquisition Loop



1 FPGA I/O node—In the Project Explorer window, click the **Mod2»Y Accel** item and drag it into the block diagram

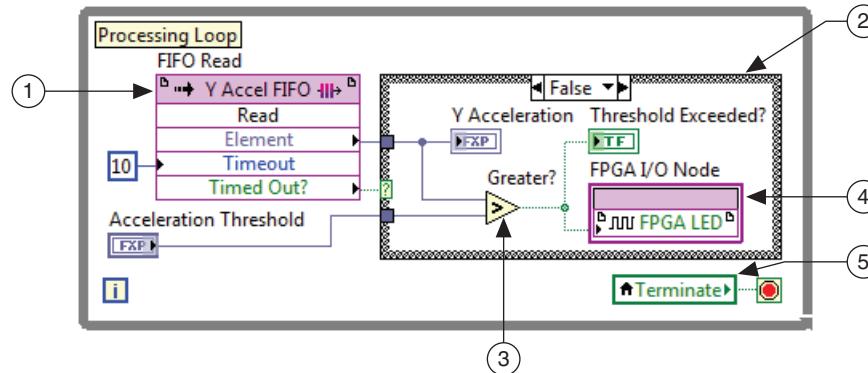
2 FIFO Write node—In the Project Explorer window, click the **Y Accel FIFO** item and drag it to the block diagram

3 Or function—Stops the execution of the loop if the Stop button is clicked or the Y Accel FIFO times out

8. Turn on the context help and note the data type of the data passed from the FPGA I/O node to the FPGA FIFO. This is the same data type that you used when you defined Y Accel FIFO.

9. Modify the Processing Loop, as shown in Figure 8-6, using the following items:

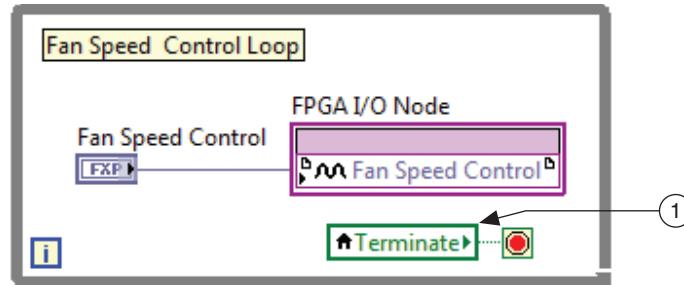
Figure 8-6. Accelerometer VI—Processing Loop



- 1 FIFO Read node—Right-click the Y Accel FIFO node and select **Select Method»Read**
- 2 Case Structure—The False case of this structure executes if the FIFO Read node does not time out. The True case remains blank so that the loop quickly proceeds to the next loop iteration if the FIFO Read node times out
- 3 Greater? function—This function compares the element output of the FIFO Read node to the value of the Acceleration Threshold control
- 4 FPGA LED node—This node turns on the FPGA LED on the CompactRIO chassis if the Y Acceleration value exceeds the Acceleration Threshold. In the Project Explorer window, click **Mod2»Y Accel** and drag it to the block diagram. Right-click the FPGA I/O node and select **Select FPGA I/O»Chassis I/O»FPGA LED**
- 5 Local Variable—This variable stops the Processing Loop at the same time as the Acquisition Loop. Add a Local Variable to the block diagram. Right-click the Local Variable and select **Select Item»Terminate**

10. Modify the Fan Control Loop, as shown in Figure 8-7.

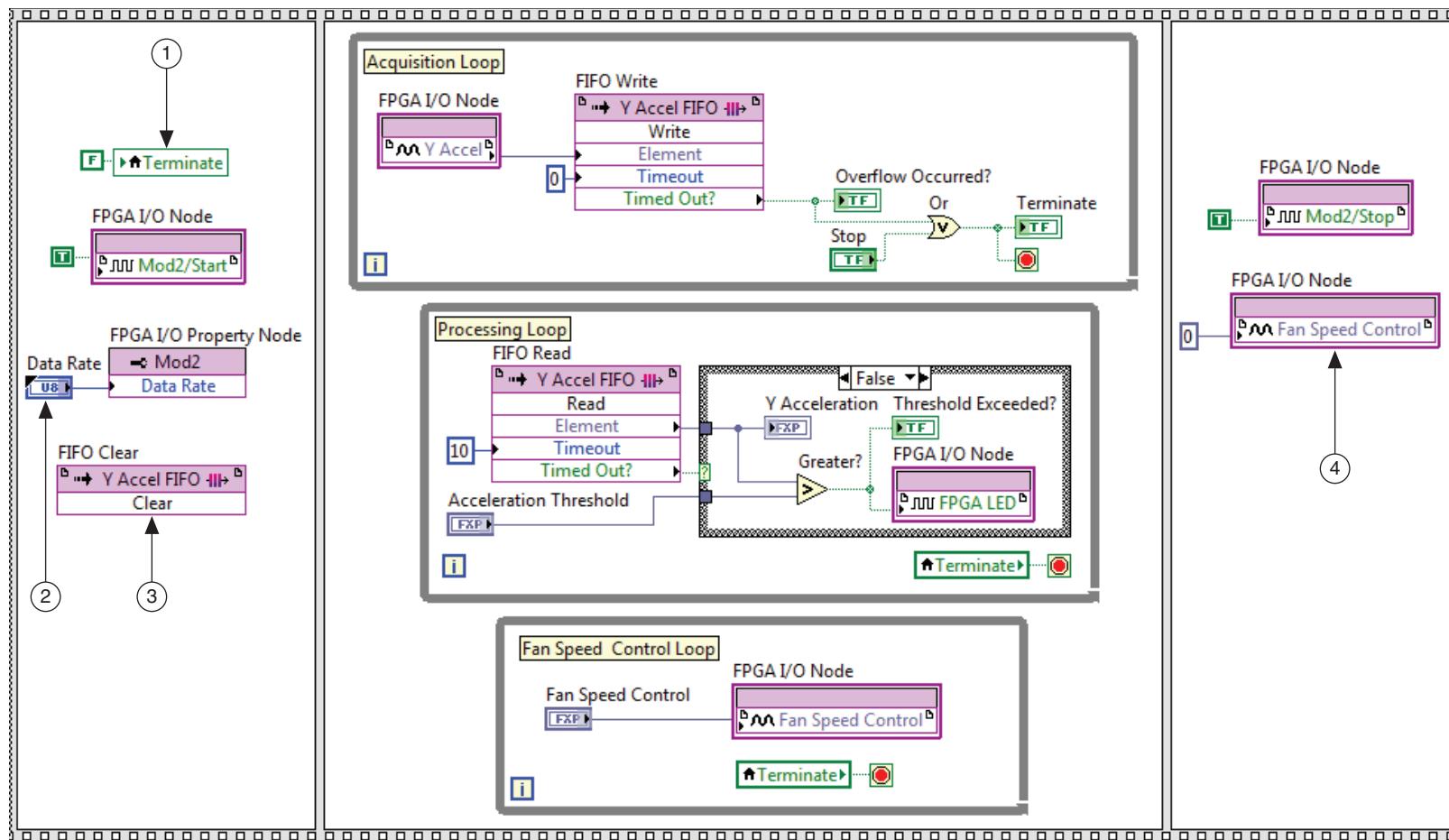
Figure 8-7. Accelerometer.vi—Fan Speed Control Loop



1 Local Variable for the Terminate indicator—This variable stops the Fan Control Loop at the same time as the Processing Loop and the Acquisition Loop

11. Modify the block diagram of the Accelerometer VI, as shown in Figure 8-8.

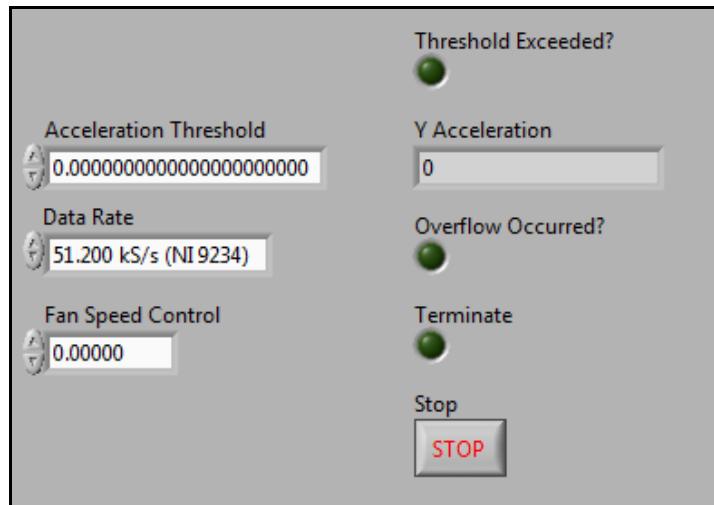
Figure 8-8. Accelerometer.vi—Block Diagram



- 1 Local Variable for the Terminate indicator—This initializes the value of the Terminate indicator and local variable to FALSE.
- 2 Data Rate control—This control specifies the rate at which the module acquires data in the Acquisition Loop. Right-click the input of the FPGA I/O Property Node and select **Create»Control**.
- 3 FIFO Clear—This node clears the FIFO before executing second frame of the Sequence structure. Right-click the **Y Accel FIFO** node and select **Select Method»Control»Clear**.
- 4 FPGA I/O node—This node sets the Fan Speed back to 0. In the Project Explorer window, click **Mod 3»Fan Speed Control** and drag it to the block diagram.

12. Arrange the front panel as shown in Figure 8-9.

Figure 8-9. Accelerometer.vi – Front Panel



- Resize the Acceleration Threshold and Fan Speed to display the full precision of these controls.

13. Select **File»Save All** to save your VI and project.

Testing

1. Test on the Development Machine.

 **Note** Always test your FPGA code on the development machine to verify the functionality prior to compiling.

- Set the FPGA target to execute the VI on the **Development Computer with Simulated I/O**.
- Enter the following values for your front panel controls:
 - Acceleration Threshold: 0
 - Data Rate: If using NI 9234, select 51.200 kS/s. If using NI 9233, select 50.000 kS/s.
 - Fan Speed Control: 5.00000



Note Since the development machine, by default, uses random data for I/O nodes, the acceleration data that is returned by the FPGA I/O node will be randomly generated and will range from -8 to +7.9999. An Acceleration Threshold value of zero should result in the LED being turned on for approximately half of the time.

2. Compile and run the VI on the FPGA Target.
 - Set the FPGA target to execute the VI on the **FPGA Target**.
 - Run the VI.
 - Once compilation finishes, close the Compilation Status window.
 - Use the following values for your front panel controls:
 - Acceleration Threshold: 0
 - Data Rate: If using NI 9234, select 51.200 kS/s. If using NI 9233, select 50.000 kS/s.
 - Fan Speed Control: 1.00000
3. Flip the Fan Speed Control switch to **BNC**.
4. Change the value of the Acceleration Threshold and note the impact on Threshold Exceeded? indicator on the front panel and the FPGA LED on the NI cRIO-9074 chassis.
5. Change the value of the Fan Speed Control and observe the change in the speed of the fan. On the Sound and Vibration Signal Simulator, toggle the fan between balanced and unbalanced. Notice the impact on the Y-Acceleration values that are acquired.
6. Close the VI and the Project.

End of Exercise 8-1

Modular Programming

Exercise 10-1 Creating an FPGA SubVI

Goal

Create an FPGA subVI and call it from an FPGA Main VI.

Scenario

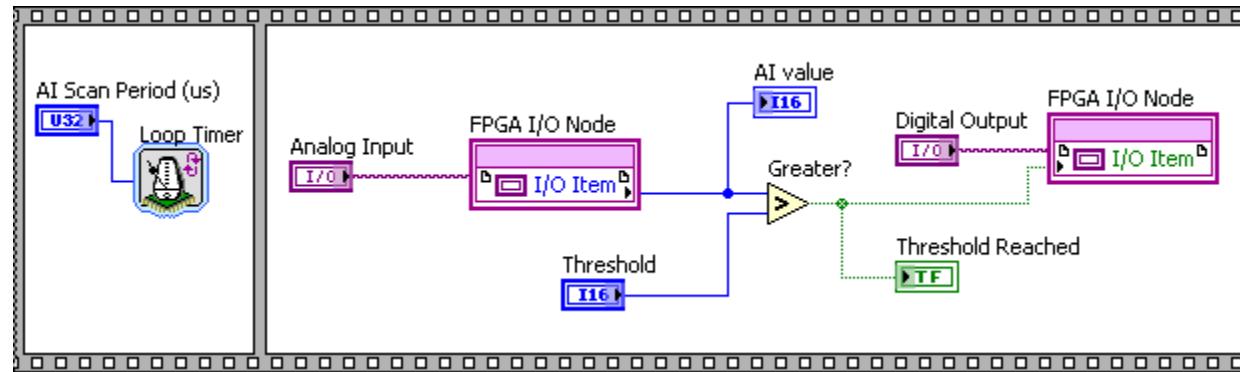
You are given the task of creating an FPGA VI that must continuously acquire data from three analog input channels and output a high voltage on three corresponding digital output lines if a threshold is exceeded. You acquire data from each analog input channel at a different rate, so the FPGA VI contains three While Loops running in parallel. Because there is similar logic for each While Loop, you create a subVI to produce modular code and save space on the FPGA.

Implementation

1. Open an existing project with a R Series FPGA target.
 - Open `Multiple AI Monitor.lvproj` in the `<Exercises>\LabVIEW FPGA\Modular Programming` directory.
 - Notice that six FPGA I/O items have already been added to the **My Computer»FPGA Target»Connector0** virtual folder. You will use these FPGA I/O items later.
2. Create a new FPGA subVI.
 - Right-click the FPGA Target in the Project Explorer and select **New»VI**.
 - Save the VI as `Max AI (SubVI).vi` in the `<Exercises>\LabVIEW FPGA\Modular Programming` directory.
3. Examine the properties of the VI.
 - Click **File»VI Properties**.
 - Select **Execution** for Category.

- Notice that the **Preallocated clone reentrant execution** is selected by default. This option allows multiple callers to be able to call this VI simultaneously.
4. Click **OK**.
 5. Create the block diagram, as shown in Figure 10-1, to monitor a user-selectable analog input channel and control a user-selectable digital output line after waiting a set amount of time using the following items:

Figure 10-1. Max AI (SubVI) VI Block Diagram

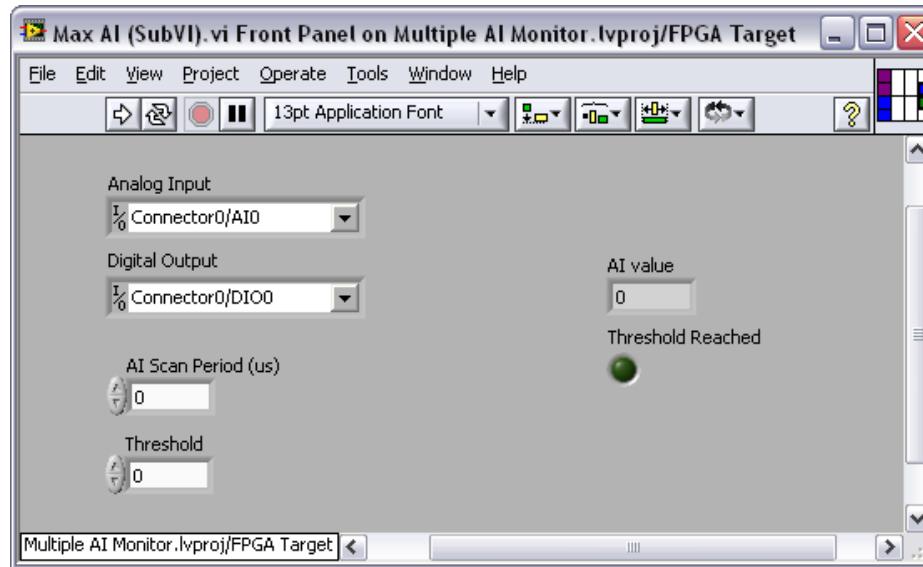


- Flat Sequence structure—Right-click the Sequence structure and select **Add Frame After**.
- Loop Timer Express VI—Add the Loop Timer Express VI in the first frame of the Flat Sequence structure. In the Configure Loop Timer dialog box, set Counter Units to **uSec**. Click **OK**.
- AI Scan Period (us) control—Right-click the Count(uSec) input of the Loop Timer Express VI and select **Create»Control**. Rename the control **AI Scan Period (us)**.
- Analog Input FPGA I/O Node and FPGA I/O Name Control
 - Place an FPGA I/O Node on the block diagram.
 - Click I/O Item and select **Connector0»AI0**.
 - Right-click the FPGA I/O In input of the FPGA I/O Node and select **Create»Control** to create a FPGA I/O name control.
 - Rename the control as **Analog Input**.

- AI value indicator—Right-click the I/O Item output of the Analog Input FPGA I/O Node and select **Create»Indicator**. Rename the indicator as `AI_value`.
- Greater? function
- Threshold control—Wire the I/O Item output of the Analog Input FPGA I/O Node to the `x` input of the Greater? function. Right-click the `y` input of the Greater? function and select **Create»Control**. Rename the control as `Threshold`.
- Threshold Reached indicator—Right-click the output of the Greater? function and select **Create»Indicator**. Rename the indicator as `Threshold_Reached`.
- Digital Output FPGA I/O Node
 - Place an FPGA I/O Node on the block diagram.
 - Click I/O Item and select **Connector0»DIO0**.
 - Right-click the I/O item and select **Change to Write**.
 - Right-click the FPGA I/O In input of the FPGA I/O Node and select **Create»Control**.
 - Rename the control as `Digital_Output`.

6. Arrange the front panel as shown in Figure 10-2.

Figure 10-2. Max AI (SubVI) Front Panel with Connector Pane



7. Edit the VI Icon.
- Right-click the VI Icon in the top-right corner of the VI and select **Edit Icon**.
 - Use the Icon Editor window to create an icon.
8. Edit the VI Connector Pane to correspond to the arrangement of the Front Panel objects. Inputs are on the left, outputs are on the right. Refer to the Connector Pane in the upper right corner of Figure 10-2.
- Right-click each input and select **This Connection is»Required**. Since they are required, you must wire to these inputs to avoid a broken Run arrow.
9. Edit the VI Description.
- Select **File»VI Properties**.
 - Select the **Documentation** category.

- Set VI description to This VI monitors a user-selectable analog input channel and controls a user-selectable digital output line after waiting a set amount of time.



Note The VI Description contains the text that appears in the Context Help window if you move the cursor over the VI icon.

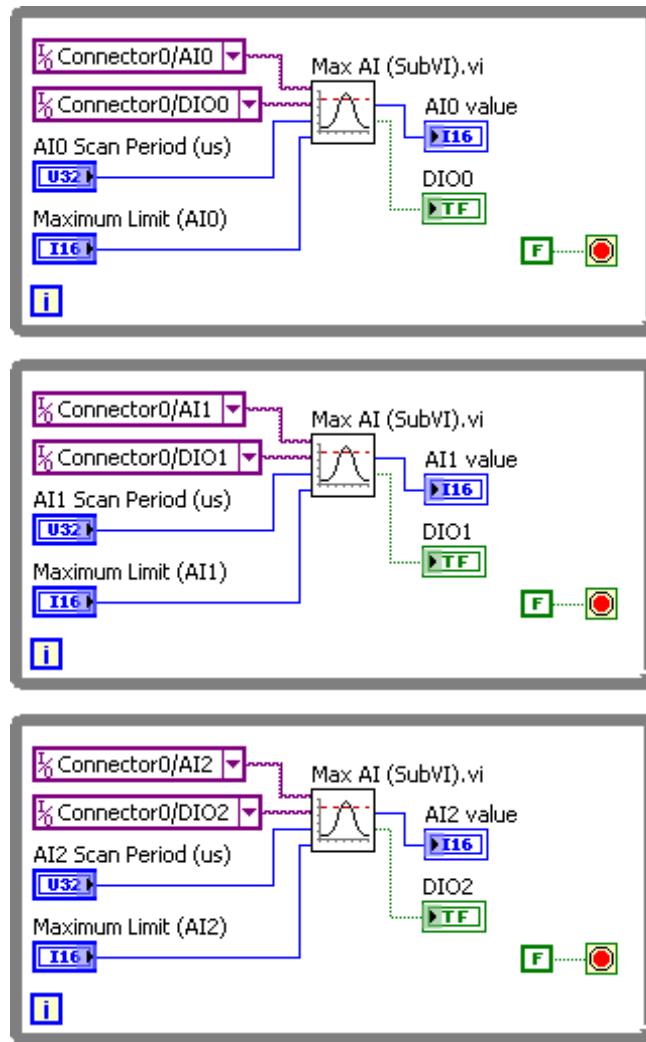
10. Save the FPGA subVI.

11. Create a new main FPGA VI.

- Right-click the FPGA Target in the Project Explorer and select **New»VI**.
- Save the VI as `Multiple AI Monitor FPGA (Main).vi` in the `<Exercises>\LabVIEW FPGA\Modular Programming` directory.

12. Create the block diagram as shown in Figure 10-3 to monitor multiple analog input channels and control multiple digital output lines at multiple scan periods using the following items:

Figure 10-3. Multiple AI Monitor FPGA (Main) VI Block Diagram



- While Loop—Wire a false constant to the Loop Condition terminal.
- Max AI (SubVI) VI
 - Drag the `Max AI (SubVI).vi` item from the Project Explorer to the block diagram.
 - Right-click the Analog Input input and select **Create»Constant**. Set the constant to **Connector/AI0**.
 - Right-click the Digital Output input and select **Create»Constant**. Set the constant to **Connector/DIO0**.
 - Right-click the AI Scan Period (us) input and select **Create»Control**. Rename the control `AI0 Scan Period (us)`.
 - Right-click the Threshold input and select **Create»Control**. Rename the control `Maximum Limit (AI0)`.
 - Right-click the AI value output and select **Create»Indicator**. Rename the indicator `AI0 value`.
 - Right-click the Threshold Reached output and select **Create»Indicator**. Rename the indicator as `DIO0`.
- Create the other two While Loops to match the constants, controls, and indicators shown in Figure 10-3.



Tip To create additional copies of the While Loop, you can copy the first While Loop and then change control and indicator names as shown in Figure 10-3.

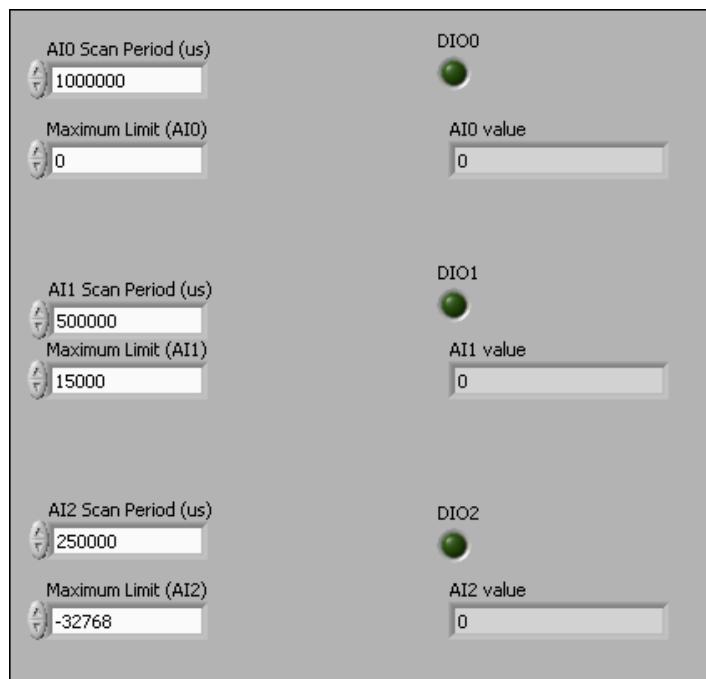
13. Save the main FPGA VI.

Testing

Test the application using simulated I/O values.

1. Configure the project to execute FPGA VIs on the development computer with simulated I/O values.
 - In the Project Explorer, right-click the FPGA target and select **Execute VI on»Development Computer with Simulated I/O**.
2. Set the controls to the values shown in Figure 10-4.

Figure 10-4. Multiple AI Monitor FPGA (Main) VI Front Panel



3. Run the Multiple AI Monitor FPGA (Main) VI and observe the results. By creating a subVI, you have created modular and reusable code.

End of Exercise 10-1

Communicating Between the FPGA and Host

Exercise 11-1 Developing a Windows Host VI

Goal

Create a Windows host VI on a Windows computer that interacts with an FPGA VI on a simulated PCIe-7852R board.

Scenario

You are assigned the task of creating an FPGA application that continuously acquires data from an analog input channel of the PCIe-7852R board and outputs a high voltage on a digital output line if the analog input value exceeds the threshold. You must design the application so that the user communicates with the FPGA through a VI running on a Windows system.

The user needs to set the threshold in units of voltage and see the analog input values in units of voltage using the front panel of the host VI. Because the front panel will use voltage units and the analog input FPGA I/O node uses binary units, the host VI must handle the conversions between voltage and binary units.

Design

Table 11-1 lists the front panel objects on the Windows host VI.

Table 11-1. Threshold Windows Host VI Inputs and Outputs

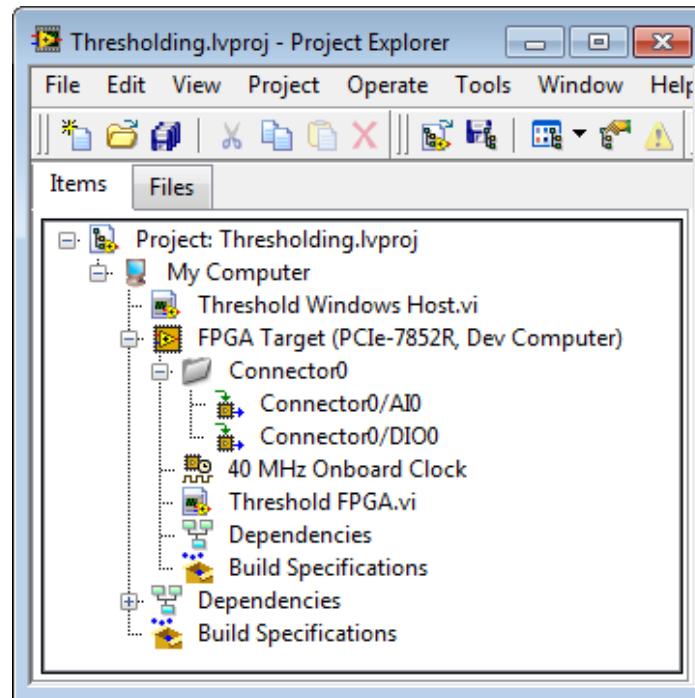
Type	Name	Properties
Numeric control	Monitoring Loop Period (ms)	32-bit Unsigned Integer, default = 1000 ms
Numeric control	Threshold (Volts)	Double-precision, default = 0
Numeric indicator	AI0 (Volts)	Double-precision, default = 0
Boolean control	Stop Host	Boolean, default = false
Boolean indicator	Threshold Exceeded	Boolean, default = false

Implementation

1. Examine an existing FPGA VI in an FPGA project.
 - Open Thresholding.lvproj located at <Exercises>\LabVIEW FPGA\Thresholding.
 - In the Project Explorer, open the Threshold FPGA VI located in **My Computer»FPGA Target**.
 - Examine the front panel controls and indicators. You will send values to these controls and retrieve values from the indicators from the host VI.
 - Examine the block diagram. This FPGA VI continuously acquires analog input values. If the value of the analog input is greater than the user-specified threshold, the FPGA VI will output a high voltage on a digital line and display the status using a Boolean indicator. This FPGA VI runs until the Stop FPGA Boolean control has a value of TRUE.
2. Create a host VI and send and retrieve values from the FPGA VI.
 - In the Project Explorer, right-click **My Computer** and select **New»VI**.
 - Save the VI as Threshold Windows Host.vi in the <Exercises>\LabVIEW FPGA\Thresholding directory.

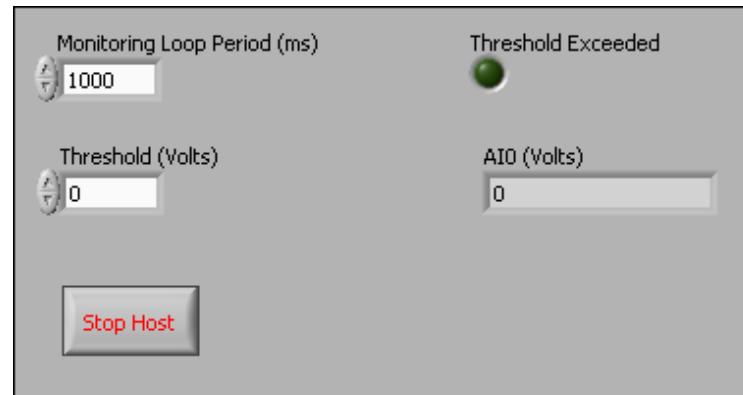
3. Verify that your Project Explorer window resembles Figure 11-1.

Figure 11-1. Thresholding Project Explorer



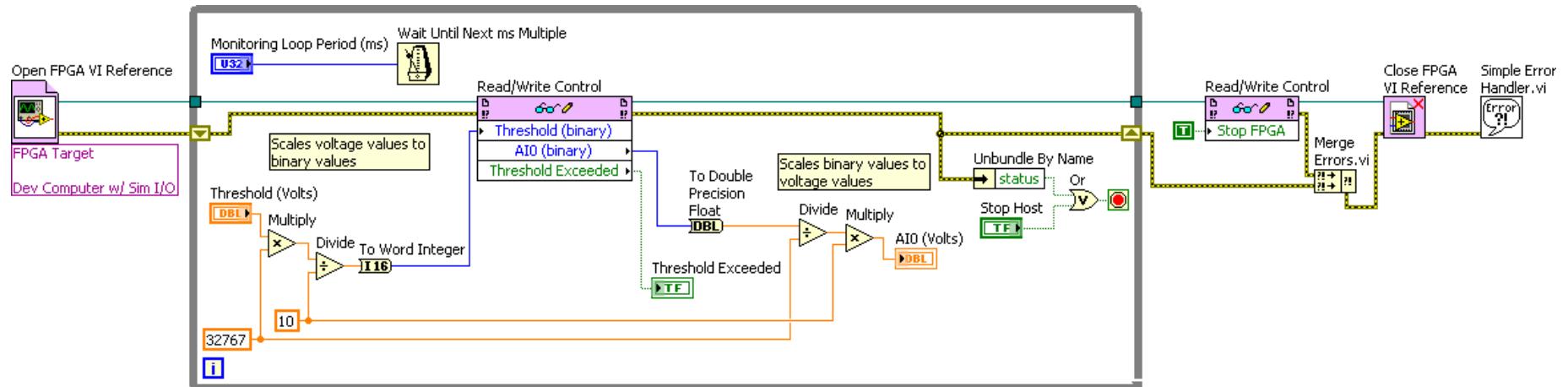
4. Create the front panel controls and indicators of the Threshold Windows Host VI as described in Table 11-1 as shown in Figure 11-2.

Figure 11-2. Threshold Windows Host VI Front Panel



5. Create the block diagram of the Threshold Windows Host VI to send and retrieve values from the FPGA VI and scale between binary and voltage values, as shown in Figure 11-3, using the following items:

Figure 11-3. Threshold Windows Host VI Block Diagram



- Open FPGA VI Reference function
 - Right-click the Open FPGA VI Reference function and select **Configure Open FPGA VI Reference**.
 - Click **VI** and select **Threshold FPGA.vi** and click **OK**. This selects the FPGA VI that this Open FPGA VI Reference function will reference.
 - Verify that the **Run the FPGA VI** checkbox is enabled. Enabling this checkbox specifies to run the FPGA VI if it is not already running when the Open FPGA VI Reference function executes.
 - Disable the **Dynamic Mode** checkbox.
 - Click **OK**.
- While Loop
 - Wire the error out terminal of the Open FPGA VI Reference function to the left border of the While Loop.
 - Right-click the tunnel and select **Replace with Shift Register**.
- Read/Write Control function
 - Wire the FPGA VI Reference wire from the Open FPGA VI Reference function to the Read/Write Control function. This allows the Read/Write Control function to populate with the names of all the controls and indicators of the FPGA VI.
 - Resize the Read/Write Control function to show three elements.
 - Click the first element and set it to **Threshold (binary)**.
 - Click the second element and set it to **AI0 (binary)**.
 - Click the third element and set it to **Threshold Exceeded**.
- Wait Until Next ms Multiple function
- Two Multiply functions
- Two Divide functions



Note The analog inputs of the PCIe-7852R have a 16-bit resolution and a voltage range from –10 V to +10 V. This VI uses the multiple and divide functions to scale between signed 16-bit integer binary values and voltage values.

- To Word Integer function
- To Double Precision Float function
- Unbundle by Name function
- Or function
 - Right-click the lower input of the Or function and select **Create»Control**.
 - Rename the control as `Stop Host`.
- Two double-precision numeric constants
- Read/Write Control function
 - Click the element and set it to Stop FPGA.
 - Right-click the Stop FPGA input and select **Create»Constant**.



Note Notice that the error in input of this Read/Write Control function is not wired. This is because you want this Read/Write Control function to execute even if an error occurs. This Read/Write Control function stops the Threshold FPGA VI by sending a TRUE value to the Stop FPGA control of the Threshold FPGA VI.

- Merge Errors function
 - Close FPGA Reference function
 - Simple Error Handler VI
6. Save the project and VI.

Testing

Test the application using simulated I/O values.

1. On the front panel of the Threshold Windows Host VI, set Monitoring Loop Period (ms) to 1000 and Threshold to 0.
2. Run the VI. The Threshold Windows Host VI should run the Threshold FPGA VI and send and retrieve values from the Threshold FPGA VI every 1000 ms. The Threshold Exceeded indicator should turn on whenever the AI0 (Volts) value is greater than the Threshold (Volts) value.



Note If the VI displays a broken Run arrow, verify in the Project Explorer that the FPGA target is set to execute on the development machine with simulated I/O.

3. When finished, click **Stop Host** to stop the application.

End of Exercise 11-1

Exercise 11-2 Developing a Real-Time Host VI

Goal

Develop an RT Host VI that communicates with the FPGA VI and a Windows VI that serves as a user interface for the Temperature Monitor project.

Scenario

You are assigned the task of developing a user interface for a PC for the Temperature Monitor application. The user interface displays the temperature versus time in a graph, stops the CompactRIO controller and FPGA applications, controls the sample rate, and logs the data to the PC's hard drive. The CompactRIO starts and runs independently of the user interface VI running on Windows.

Design

User interface objects are listed in Table 11-2. The Temperature Monitor RT Host VI will be modified by moving the Waveform Chart to the user interface on the PC.

Table 11-2. Temperature Monitor Windows PC Host Inputs and Outputs

Type	Name	Properties
Waveform Chart	Temperature History (C)	Default = Single-precision floating-point (SGL), X-axis title = Sample, Y-axis title = Temperature (C)
Square LED	Temperature Alarm	Boolean, default = False
File Path Control	Save File Path	Show browse button, default = C:\Exercises\LabVIEW FPGA\RT Host and Windows Integration\Temperature Data.lvm
Stop button	Stop RT	Boolean, Latch when released, default = False
Stop button	Stop Windows GUI	Boolean, Latch when released, default = False
Horizontal Pointer Slide	Temperature Alarm Level (C)	Single Precision, Show Digital Display, default = 30
Dial	Sample Interval (ms)	Unsigned 32-bit, default = 500, show digital display

Communicate temperature data between the CompactRIO unit and the Windows PC with a shared variable. Communicate Sample Interval and Stop commands from the Windows PC Host to the cRIO with shared variables. Host the variables on the CompactRIO RT controller.

Implementation

FPGA VI

The FPGA VI runs on the FPGA target and acquires data from the NI 9211.

1. Open Temperature Monitor.lvproj in the <Exercises>\LabVIEW FPGA\RT Host and Windows Integration directory.
2. Configure the CompactRIO target.
 - Right-click the CompactRIO target in the Project Explorer and select **Properties**.
 - In the General category, set the IP Address to the IP Address of your CompactRIO target.
3. Examine the existing FPGA VI in an FPGA project.
 - In the Project Explorer, open the Temperature Monitor FPGA VI located in **cRIO-9074»Chassis»FPGA Target**.
 - Examine the front panel controls and indicators. You will send values to these controls and retrieve values from the indicators on the RT Host VI.
 - Examine the block diagram. This FPGA VI acquires the temperature, CJC, and autozero values from the NI 9211 at a user-defined sample period. It also unbundles the status and code elements of the error cluster.
4. Run the VI to compile. You will see the Compilation Status window. Minimize this window and continue the exercise while compiling.



Caution Failure to compile at this point delays completion of this exercise.

RT Host VI

The RT Host VI runs on the Real-Time target, transfers data to and from the FPGA VI, and processes the received data.

1. Examine the existing RT host VI in the project.
 - In the Project Explorer, open the Simple Temperature Monitor RT Host VI located under **cRIO-9074**.
 - Examine the block diagram. Notice that the Simple Temperature Monitor RT Host VI uses the same FPGA Interface functions as the Threshold Windows Host VI that you developed in Exercise 11-1. The elements in the Read/Write Control function correspond to the controls and indicators on the FPGA VI.

The RT host VI updates the sample period of the FPGA VI, reads temperature data from the FPGA VI, and converts the temperature data into units of Celsius.



Note Use the FPGA Interface functions on the host VI, whether the FPGA is hosted on a Windows computer or a Real-Time target, to communicate with the FPGA VI.

- Notice that the Run arrow is broken. This is because the FPGA VI that this RT host VI references has not finished compiling yet. Once the FPGA VI has successfully compiled, the run arrow will become solid.
- 2. Examine the network-published shared variables with the RT FIFO enabled in the project. You use these variables to transfer data between the RT host VI running on the cRIO RT target and a VI running on the Windows PC.

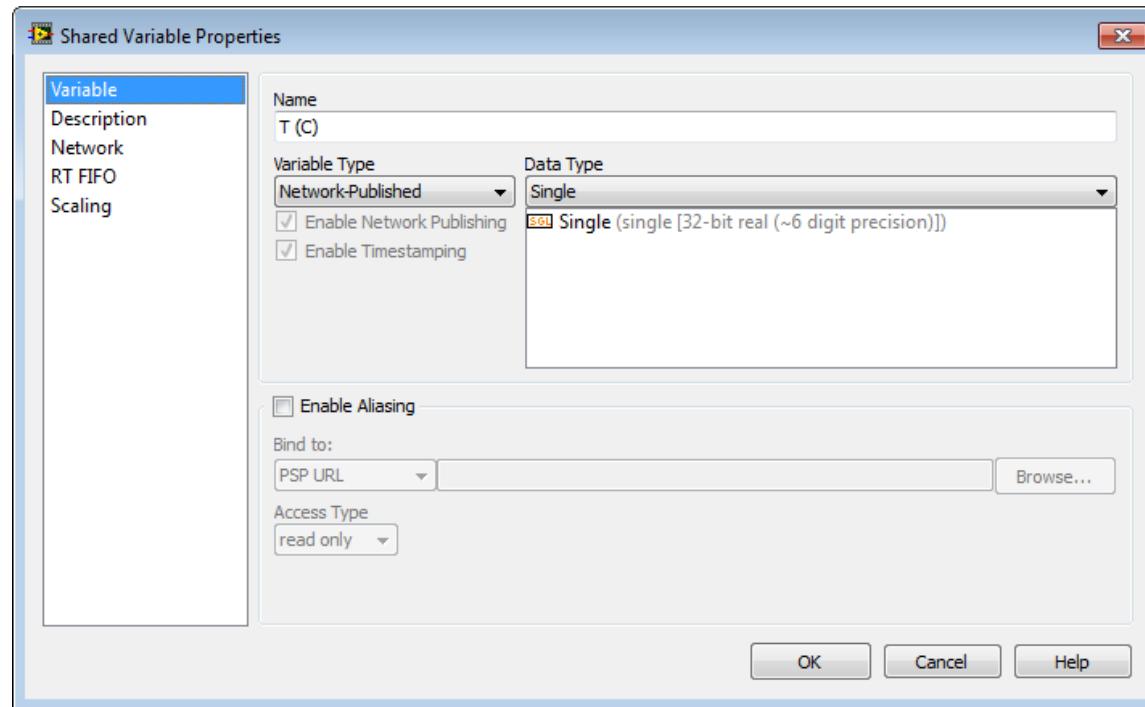


Note For more detailed information on network communication between an RT target and a Windows PC, refer to the *LabVIEW Real-Time 1* course.

- In the Project Explorer window, right-click the **cRIO-9074»Variables.lvlib»T (C)** item, and select **Properties** to examine the T (C) network-published shared variable.

- The Shared Variable Properties window opens. Examine the configuration of the variable, as shown in Figure 11-4.

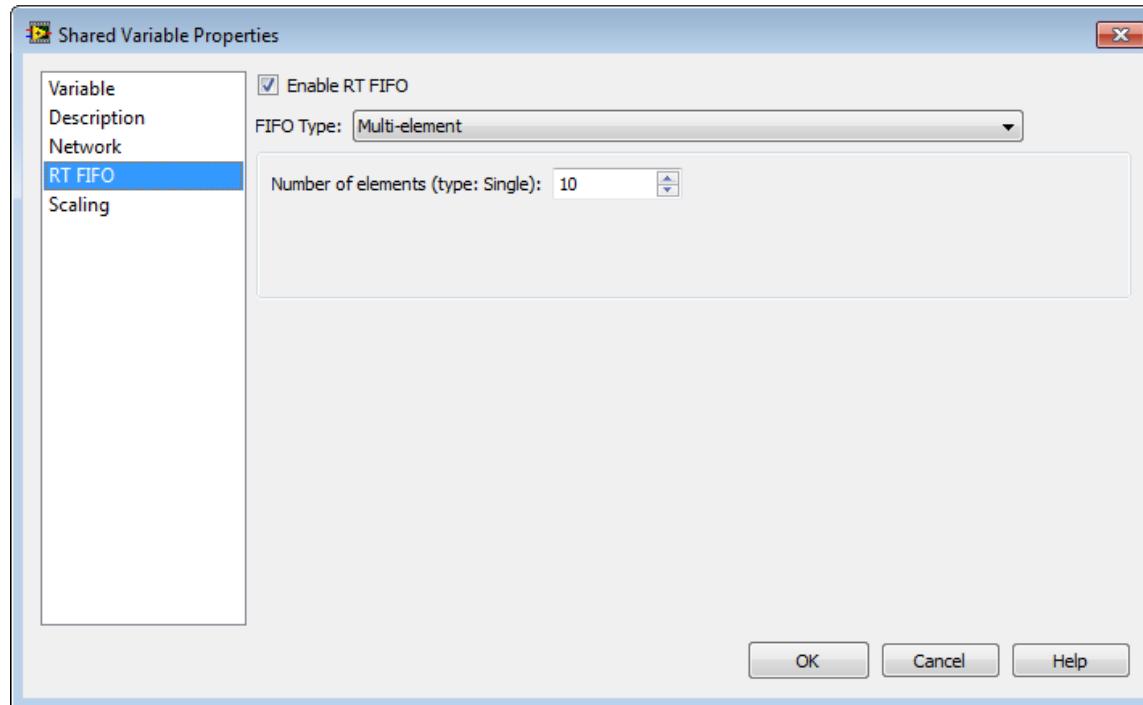
Figure 11-4. T (c) Shared Variable Properties



- Select the RT FIFO category.

- Examine the RT FIFO properties of the variable as shown in Figure 11-5.

Figure 11-5. Temperature Shared Variable RT FIFO Properties



- Click **OK**.
- Refer to Table 11-3 to see the configuration of the other shared variables in the Variables library.

Table 11-3. Temperature Monitor Project Shared Variable Properties

Variable Type	Name	Data Type	RT FIFO	FIFO Type
Network- Published	Stop	Boolean	Yes	Single Element
Network- Published	dt (ms)	32-bit unsigned integer	Yes	Single Element

3. Create a Networked Temperature Monitor RT Host VI.

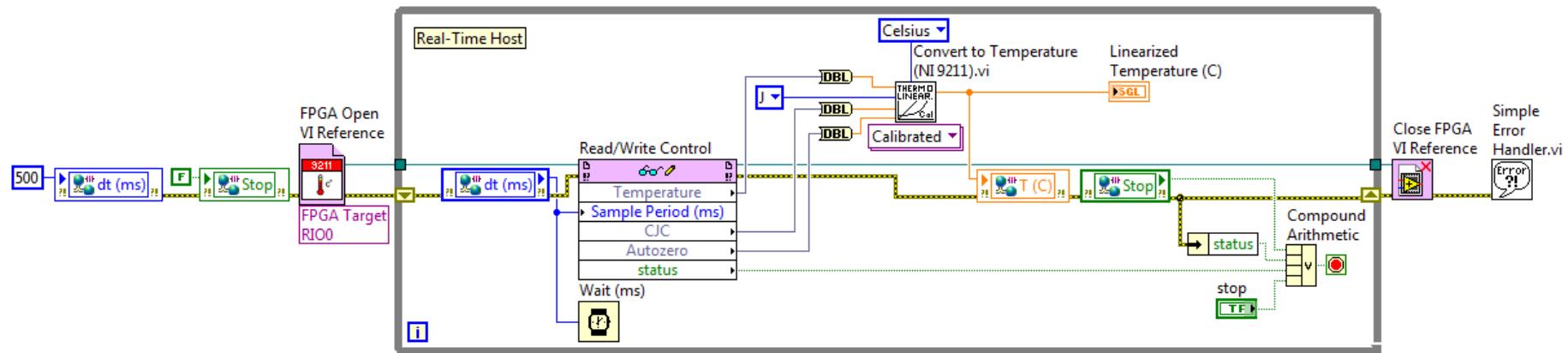
- Open the Simple Temperature Monitor RT Host VI.
- Select **Save As**, select **Open Additional Copy** and select **Add copy to Temperature Monitor.lvproj**.
- Click **Continue**. Save the VI as <Exercises>\LabVIEW FPGA\Temperature Monitor\Networked Temperature Monitor RT Host.vi.

This creates a second RT host VI in the project. The remainder of this exercise uses the Networked Temperature Monitor RT Host VI.

- Close the Simple Temperature Monitor RT Host VI. Do not save any changes.

In the following steps, you build the block diagram shown in Figure 11-6:

Figure 11-6. Networked Temperature Monitor RT Host VI Block Diagram



4. Add the shared variables to the block diagram.

- Select all of the shared variables in the Project Explorer by holding down the **<Shift>** key and clicking the first and last item in the list.
- Drag all of the variables to the Networked Temperature Monitor RT Host VI block diagram.

5. Initialize the shared variables.

- <Ctrl>-click the Stop shared variable and drag a copy before the Open FPGA VI Reference function.
- Right-click the Stop shared variable and select **Access Mode»Write**.
- Wire a False constant to the input terminal of the Stop shared variable.
- Place a copy of the dt (ms) shared variable before the Stop shared variable.
- Right-click the dt (ms) shared variable and select **Access Mode»Write**.
- Wire a numeric constant with a value of 500 to the input terminal of the dt (ms) shared variable.
- Right-click the 500 constant and select **Representation»U32**.
- Wire the initialization code as shown in Figure 11-6.

6. Read from and write to the shared variables.

- Delete the Sample Period (mSec) control on the block diagram.
- Press <Ctrl-B> to delete all broken wires.
- Expand the block diagram to create room for the shared variables.
- Place a copy of the dt (ms) shared variable before the Read/Write Control function.
- Place a copy of the T (C) shared variable after Convert to Temperature (NI 9211) VI.
- Right-click the T (C) shared variable and select **Access Mode»Write**.
- Place a copy of the Stop shared variable after the T (C) shared variable.
- Wire the shared variables as shown in Figure 11-6.

7. Set equal delays in the system to avoid race conditions. You execute the FPGA, RT host VI, and Windows host VI at the same rate. You develop better synchronization techniques in a later exercise.
 - Wire the dt (ms) shared variable to the Wait (ms) function.
8. Handle shutdown.
 - Expand the Compound Arithmetic (OR) function to four elements.
 - Wire the output from the Stop shared variable to the new element, as shown in Figure 11-6.
9. Save the VI and save the project.

Test the RT Host

1. Test the Networked Temperature Monitor RT Host VI.

- If prompted, save any unsaved changes.

- Run the Networked Temperature Monitor RT Host VI.



Note The temperature data in the Waveform Chart is delayed because it takes some time for the shared variable engine to deploy.

- Touch the thermocouple to vary the temperature. Observe the result.

- Stop the VI.



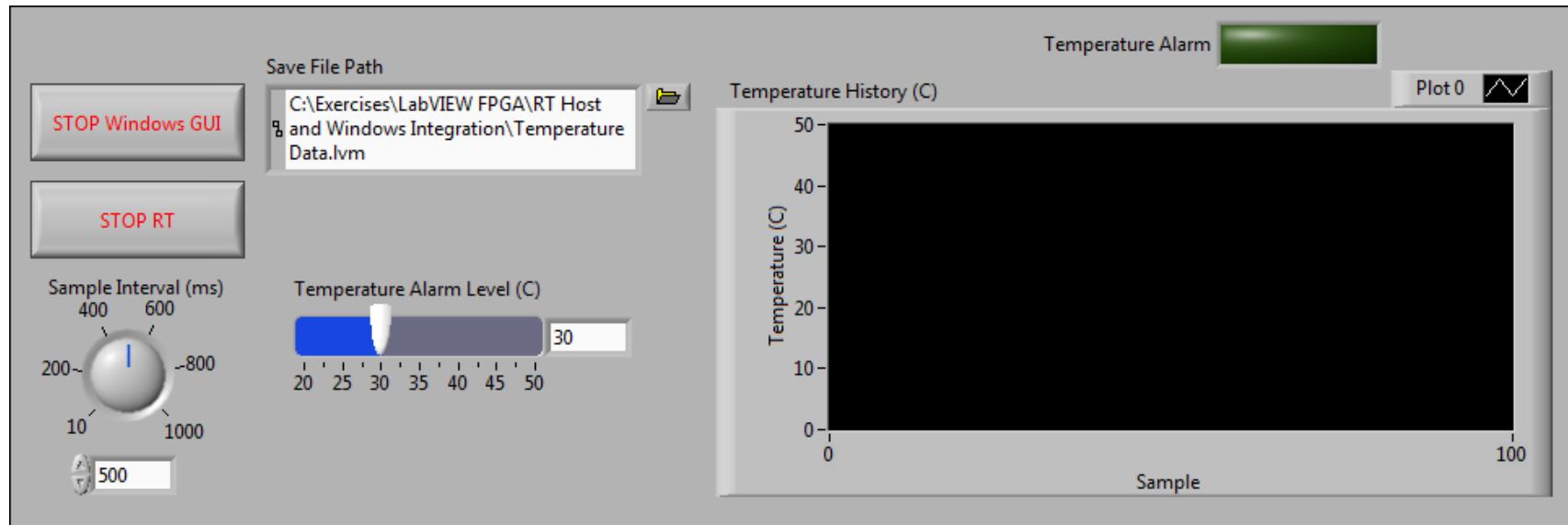
Note If you are building a LabVIEW Real-Time application, you must take proper steps to deploy your final application as a startup executable to your RT target. In this exercise, we use interactive front panel communication for debugging purposes, and we will not go into the details of how to deploy your final application. This information can be found in the *LabVIEW Real-Time 1* course and the *LabVIEW Real-Time Help*.

Create the Windows VI

1. Add a Windows VI to the project.
 - In the Project Explorer, right-click **My Computer** and select **New»VI**.
 - Save the VI as <Exercises>\LabVIEW FPGA\RT Host and Windows Integration\Temperature Monitor PC.vi.
2. Build the front panel.
 - Place the following on the front panel window:
 - Two stop controls
 - A dial control
 - A file path control
 - A horizontal pointer slide
 - A square LED
 - A waveform chart
 - Set the properties of the controls as specified in Table 11-2, as shown in the *Design* section.

- Arrange the controls and indicators on the front panel as shown in Figure 11-7.

Figure 11-7. Temperature Monitor PC VI Front Panel

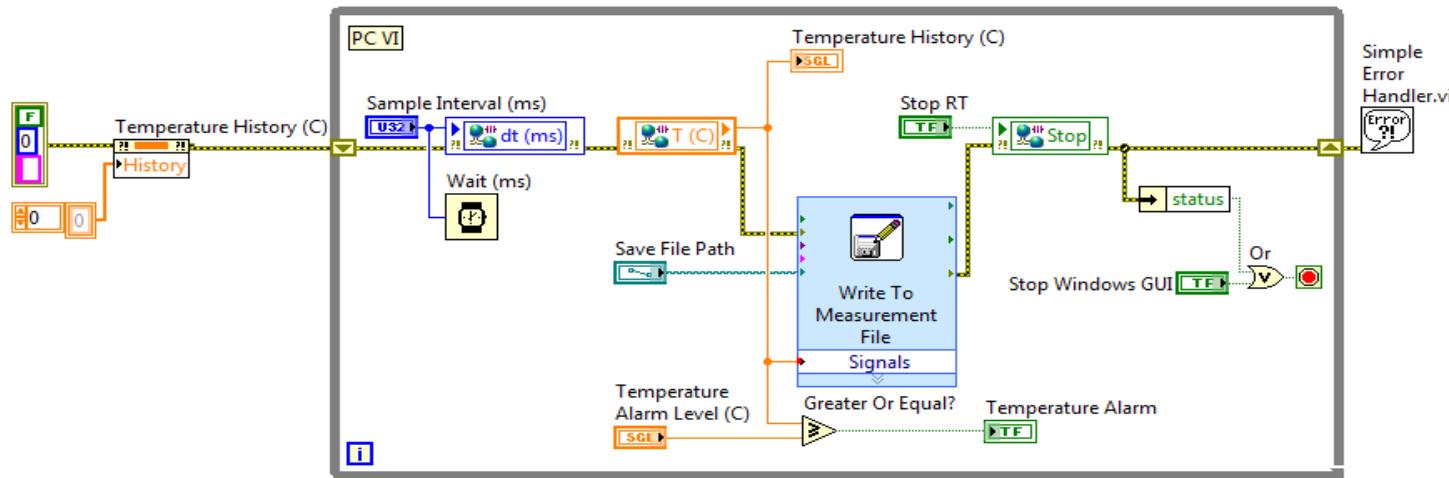


In the following steps, you build the block diagram shown in Figure 11-8:

3. Add shared variables for communication between the PC and RT VIs.

- Drag the **dt (ms)**, **T (C)**, and **Stop** shared variables from the Project Explorer to the Temperature Monitor PC VI block diagram and arrange them as shown in Figure 11-8.

Figure 11-8. Temperature Monitor PC VI Block Diagram



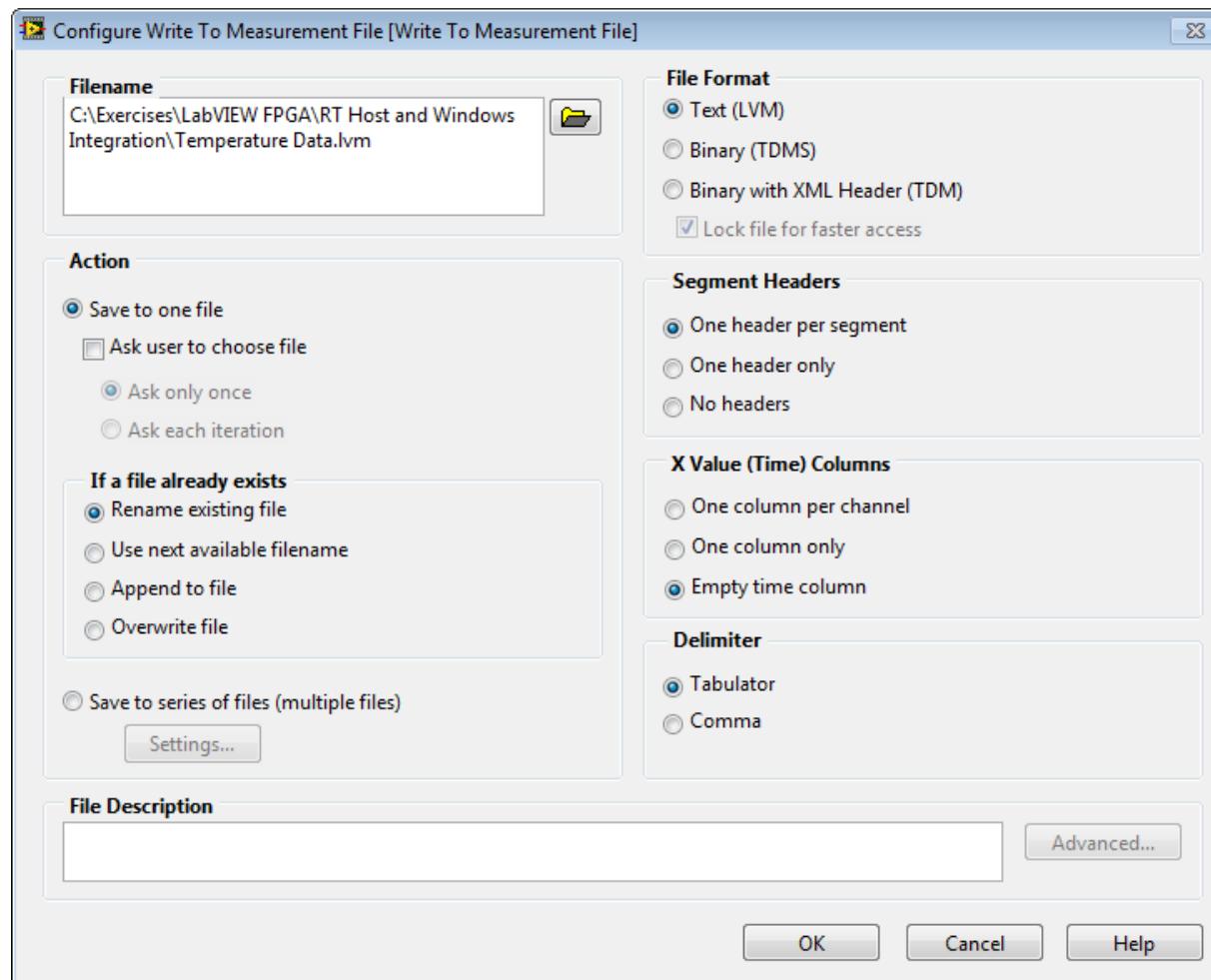
4. Write the temperature data to a file on the Windows PC.

- Add a Write to Measurement File Express VI to the block diagram.



- When the configuration dialog box opens, configure it as shown in Figure 11-9 and click **OK**.

Figure 11-9. Write to Measurement File Configuration



5. Add a loop and timing.
 - Add a While Loop to the block diagram as shown in Figure 11-8.
 - Add a Wait (ms) function to the block diagram.
6. Compare the actual temperature with the alarm set point.
 - Add a Greater Or Equal? function to the block diagram.
 - Wire the T (C) shared variable to the x input of the Greater Or Equal? function.
 - Wire the T (C) shared variable to the Temperature History (C) Waveform Chart.
 - Wire the Temperature Alarm Level (C) control to the y terminal of the Greater Or Equal? function.
7. Stop the targets.
 - Insert an Or function between the Stop Windows GUI control and the conditional terminal.
 - Wire the Stop Windows GUI control to the y input of the Or function.
 - Right-click the error wire between the Stop shared variable and the shift register and select **Cluster, Class and Variant Palette»Unbundle By Name**.
 - Wire the Stop shared variable error out output to the Unbundle By Name function.
 - Wire the Unbundle By Name function output to the x input of the Or function.
8. Handle Errors.
 - Add a Simple Error Handler VI to the right side of the While Loop.
 - Wire the dt (ms) shared variable error out as shown in Figure 11-8.
 - Right-click the error tunnel on the right loop border and select **Replace With Shift Register**.



9. Clear the chart.

- Right-click the Temperature History (C) chart and select **Create»Property Node»History Data** and place the Property Node on the left side of the While Loop.
- Right-click the Property Node and select **Change All to Write**.
- Wire errors as shown in Figure 11-8.
- Right-click the History Data Property Node and select **Create»Constant**.
- Right-click the Error In terminal of the History Data Property Node and select **Create»Constant**.

10. Wire the controls to the shared variables, Wait function, Write to Measurement File VI, and the conditional terminal as shown in Figure 11-8.

11. Save the VI and project.

Test the Windows VI

1. Test the Temperature Monitor PC VI.

- Open and run the Networked Temperature Monitor RT Host VI. Leave the front panel visible on the monitor.
- On the front panel of the Temperature Monitor PC VI, set the Sample Interval (ms) control to 500.
- Run the Temperature Monitor PC VI. You may have to wait for communication to be established.
- When the temperature values stabilize, touch the thermocouple to create a temperature variation and observe the results.
- Click the **Stop Windows GUI** button to stop the Temperature Monitor PC VI without stopping the CompactRIO. Notice that the RT Host continues to run.
- Restart the Temperature Monitor PC VI. The shared variable may report some values of **0** during redeployment and while communication is re-established.
- Click the **Stop RT** button to stop the Networked Temperature Monitor RT Host VI.
- Stop the Temperature Monitor PC VI. If you stop the PC VI first, you must stop the CompactRIO from the RT Host Interactive Front Panel Communication.
- Open the data file with an ASCII compatible program such as Microsoft Excel. Each time the Temperature Monitor PC VI runs, it appends data to a file. Consider changing the file name for multiple tests.

2. Close all VIs.
3. Close the project.

End of Exercise 11-2

Exercise 11-3 Transferring Buffered Data Using DMA FIFO

Goal

Transfer buffered data from the FPGA VI to the Windows host VI.

Scenario

You have to develop an application that acquires analog data from a PCIe-7852R and transfers every acquired value to the host computer. You use DMA FIFOs to transfer large amounts of buffered data between an FPGA target and the host computer. You develop this application using simulated hardware since you do not have a PCIe-7852R for development/testing.

Design

FPGA VI Design

Name the DMA FIFO AI DMA. It is a stack of 1023 I16 values and should be configured as a target-to-host DMA. The FPGA VI will write data to the DMA FIFO and notify the host VI if an overflow occurs.

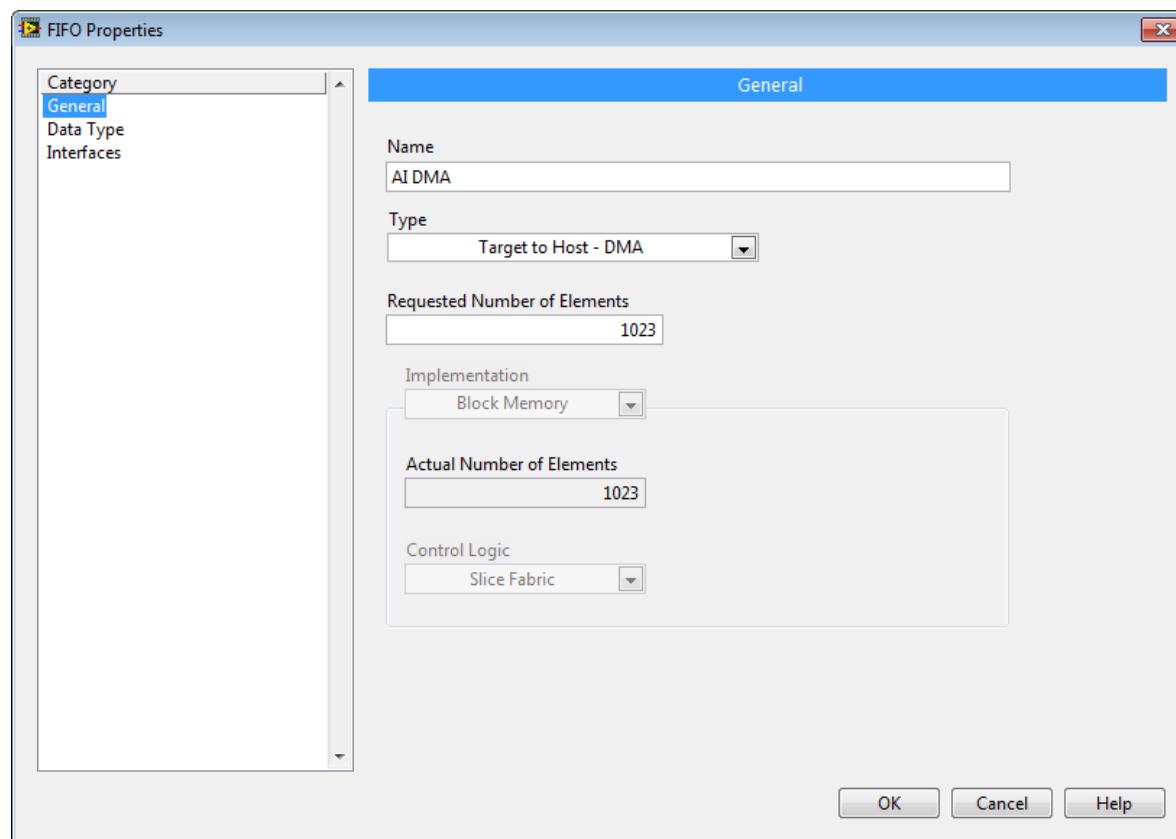
Windows Host VI Design

This VI acts as the user interface for the application. This VI loads and runs the FPGA VI and displays the buffered data acquired from the FPGA in a waveform graph. If the Windows Host VI is notified that an overflow occurred in the DMA FIFO, the Windows Host VI clears the DMA FIFO. The Windows Host VI also keeps track of the total number of overflows and display flushed data each time it clears the DMA FIFO.

Implementation

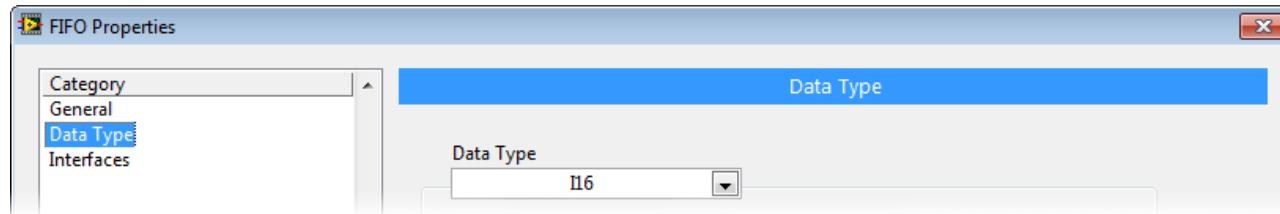
1. Open DMA FIFO.lvproj in the <Exercises>\LabVIEW FPGA\DMA FIFO directory.
2. Create AI DMA to act as your DMA channel for transferring analog input data from the FPGA target to a host VI.
 - Right-click the FPGA Target and select New»FIFO.
 - Configure the FPGA FIFO Properties window as shown in Figure 11-10.

Figure 11-10. AI DMA Properties—General



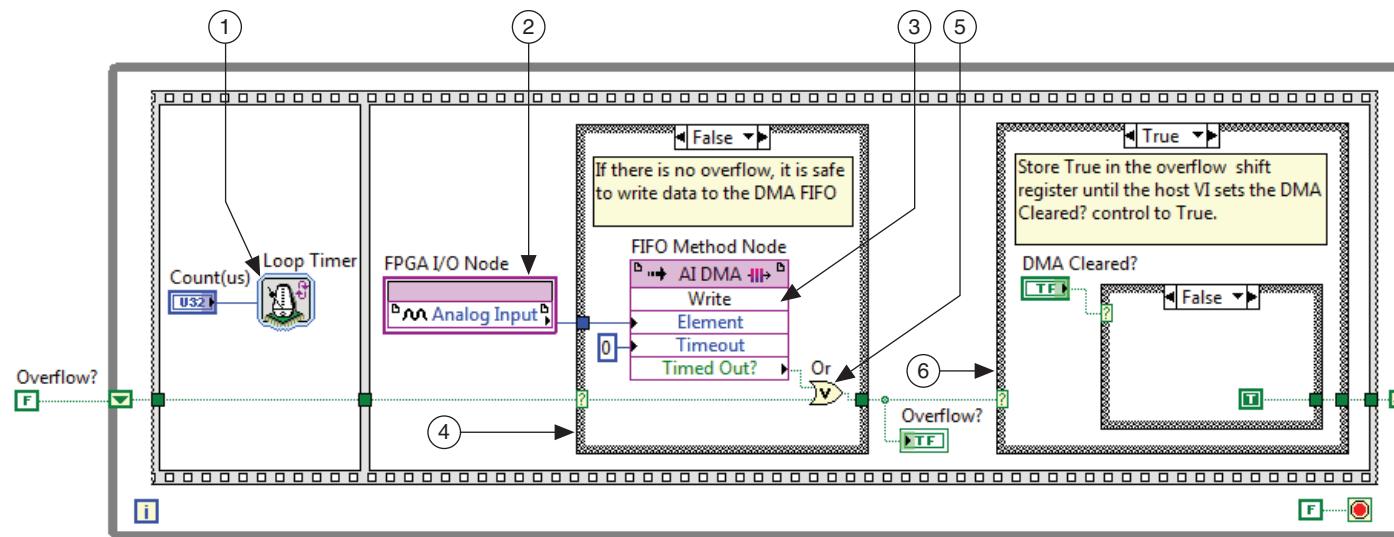
- Select the **Data Type** category and verify the settings as shown in Figure 11-11 and click **OK**.

Figure 11-11. AI DMA Properties—Data Type



3. From the Project Explorer window, open **FPGA Target»DMA Write-FPGA.vi**.
4. Modify the DMA Write-FPGA VI, as shown in Figure 11-12, to write elements to the DMA FIFO.

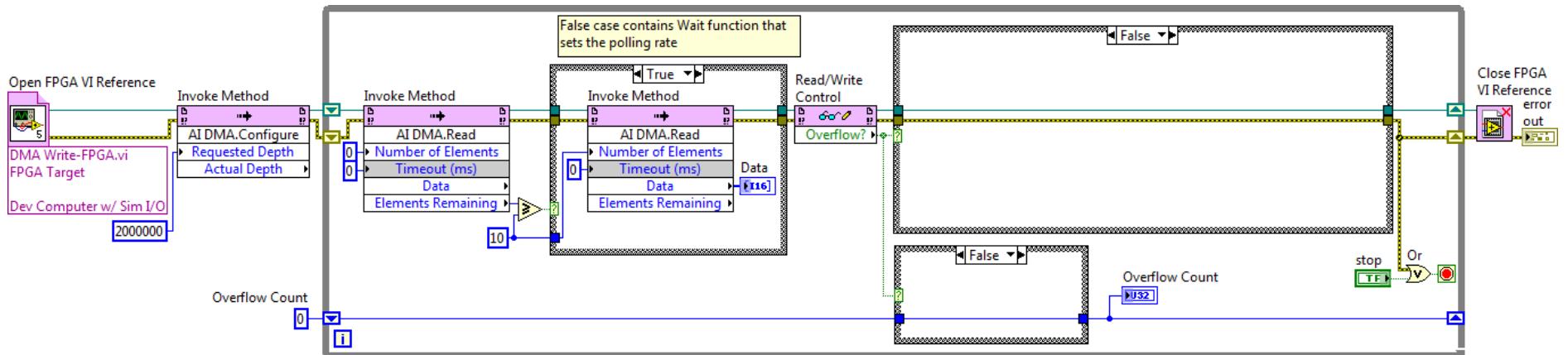
Figure 11-12. DMA Write-FPGA.vi Block Diagram



- 1 Loop Timer Express VI—This VI determines the sample period of the acquisition. In the configuration window, set Counter units to **uSec** and set Size of internal counter to **32 Bit**.
- 2 FPGA I/O Node—This node acquires data that will be written to the DMA FIFO. Configure this node to acquire data from the Analog Input on Connector0.
- 3 FIFO Method Node—Drag the AI DMA item from the Project Explorer window to the block diagram to create the AI DMA Write method.
- 4 Case structure for DMA FIFO—If an overflow does not occur, the Case structure writes data to the DMA FIFO. If an overflow occurs, the Case structure does not write any data to the DMA FIFO and just passes the overflow Boolean wire through.
- 5 Or function—This function is used with the shift register to latch the result of the Timed Out? output of the FIFO Method node. If the FIFO Method node times out, an overflow has occurred.
- 6 Case structure for overflow—If an overflow does not occur, the Case structure just passes the overflow Boolean wire through to the shift register. If an overflow occurs, the VI will store a TRUE in the overflow shift register and the VI will not write any data to the DMA FIFO until the host VI sets the DMA Cleared? control to a value of TRUE.
- 5 Save and close the VI.
- 6 Select **My Computer»DMA Read-Windows Host.vi** from the Project Explorer window.

7. Modify the block diagram, as shown in Figure 11-13. This VI runs on the Windows host computer and serves as the user interface to the FPGA VI. This VI passes data to the controls that you created on the front panel of DMA Write-FPGA.vi and reads the data that was written to the AI DMA FIFO.

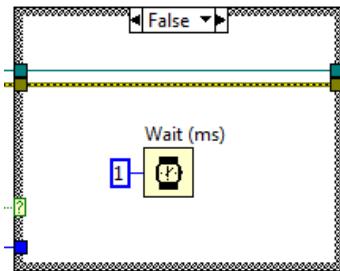
Figure 11-13. DMA Read-Windows Host VI Block Diagram



- Open FPGA VI Reference—This function is located on the FPGA Interface Palette. This function opens a reference to DMA Write-FPGA.vi.
 - Place this function on the block diagram. Right-click the function and select **Configure Open FPGA VI Reference**.
 - In the Open category, select **VI»DMA Write-FPGA.vi**. Click **OK**.
 - Verify that the **Run the FPGA VI** checkbox is enabled. Enabling this checkbox specifies to run the FPGA VI if it is not already running when the Open FPGA VI Reference function executes.
 - Disable the **Dynamic Mode** checkbox.
 - Click **OK**.
- Invoke Method (DMA.Configure)—The Configure method configures the number of elements in the host buffer part of the AI DMA FIFO. Select **AI DMA» Configure**.
- Invoke Method (DMA.Read)—Use this method to return the number of elements remaining that are available for reading.
- Greater or Equal? function—Use this function to check if a fixed number of elements are ready to read from the DMA FIFO.

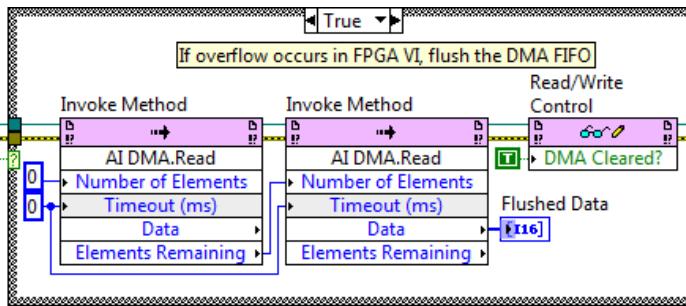
- 1st Case structure—If 10 elements are available to read from the DMA FIFO, this Case structure will use a AI DMA.Read method to read 10 elements. If 10 elements are not yet available to read from the DMA FIFO, this Case structure will wait to allow the processor some time to sleep.
 - Place an Invoke Method (DMA.Read) in the True case and wire as shown in Figure 11-13.
 - Place a Wait (ms) function in the False case and wire as shown in Figure 11-14.

Figure 11-14. False case of 1st Case Structure



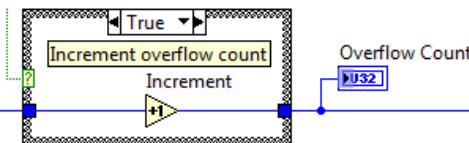
- Read/Write Control function—Use this function to check whether or not an overflow occurred in the DMA FIFO by reading the current value of the Overflow? control on the FPGA VI.
- 2nd Case structure—If the overflow does not occur in the DMA FIFO, this Case structure does not change the Overflow Count. If overflow does occur in the DMA FIFO, then this Case structure clears the DMA FIFO by reading the remaining elements in the DMA FIFO and then notifies the FPGA VI that the DMA is cleared by setting the Overflow? control in the FPGA VI to TRUE.
 - Wire the False case as shown in Figure 11-13.
 - Place two Invoke Methods (DMA.Read) and a Read/Write Control function in the True case as shown in Figure 11-15.

Figure 11-15. True case of 2nd Case Structure



- 3rd Case structure—If the overflow does not occur in the DMA FIFO, this Case structure will not add any additional logic. If overflow does occur in the DMA FIFO, then this Case structure increments the Overflow Count.
 - Wire the False case as shown in Figure 11-13.
 - Place an Increment function in the True case as shown in Figure 11-16.

Figure 11-16. True case of 3rd Case Structure



- Close FPGA VI Reference—This function closes the reference that you opened with Open FPGA VI Reference.
 - Or function and Stop Button control—This function and control are used to stop the While Loop if an error occurs or if the user clicks the Stop Button on the front panel.
 - Wire the diagram as shown.
8. Save the VI.

Testing

1. Notice in the Project Explorer that the FPGA Target is set to execute on the development computer with simulated I/O.
2. Run DMA Read-Windows Host.vi on the host computer.
3. Verify that the DMA Read-Windows Host VI receives buffered data transferred from the FPGA VI, which is running in simulation mode.
4. Save and close the VI and project when finished.

End of Exercise 11-3

Exercise 11-4 Interleaving a DMA FIFO

Goal

Create an FPGA VI that uses a single interleaved DMA FIFO to transfer multiple channels of analog data from the FPGA target to the RT host computer.

Scenario

You have to develop an application that acquires multiple channels of analog data from an NI 9234 and must transfer every acquired value to the host computer. You decide to use a DMA FIFO to transfer the data. Because your cRIO-9074 RT target has only three DMA channels available, you want to conserve the amount of DMA channels you use for additions and modifications to the application in the future. You decide to interleave the data from multiple analog input channels into a single DMA FIFO on the FPGA VI prior to transfer and de-interleave the data on the RT host VI.

Design

FPGA VI Design

Name the DMA FIFO Accelerometer Data. It is a stack of 1023 fixed-point values, configured as $\langle\pm,24,4\rangle$. The FPGA VI acquires accelerometer data from two channels on the NI 9234. The accelerometer data is interleaved and written to the Accelerometer Data DMA FIFO. The FPGA VI will notify the host VI if an overflow occurs.

RT Host VI Design

This VI loads and runs the FPGA VI and de-interleaves the Accelerometer Data DMA FIFO into accelerometer data for each channel. This VI displays multiple channels of the acquired accelerometer data in a waveform graph. If the Windows Host VI is notified that an overflow occurred in the DMA FIFO, the Windows Host VI clears the DMA FIFO. The Windows Host VI also keeps track of the total number of overflows and displays flushed data each time it clears the DMA FIFO.

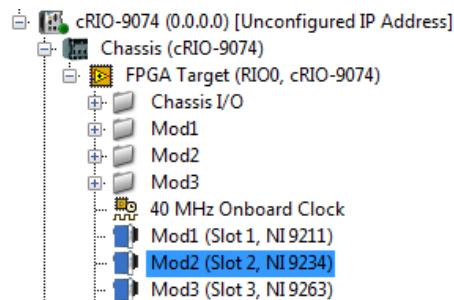
Implementation

Creating the FPGA VI

1. Open Interleaved DMA.lvproj from the <Exercises>\LabVIEW FPGA\Interleaved DMA directory.
2. Configure the CompactRIO target.
 - Right-click the CompactRIO target in the Project Explorer window and select **Properties**.
 - In the General category, set the IP Address to the IP Address of your CompactRIO target.

3. Notice that the CompactRIO target in the Project Explorer uses a NI 9234 module, as shown in Figure 11-17.

Figure 11-17. NI 9234 Module in Project Explorer



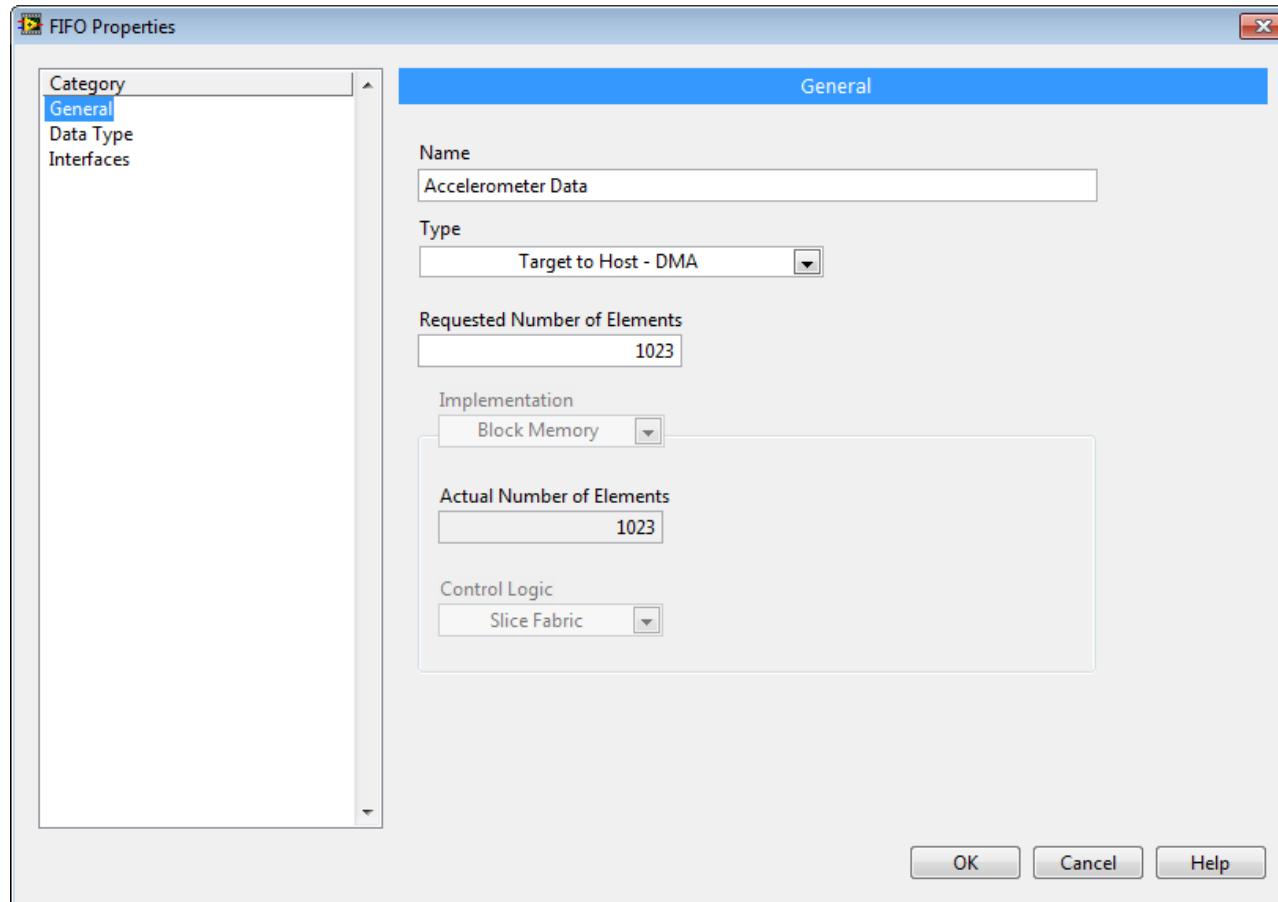
If you are using a NI 9233 module instead of a NI 9234, complete this step to modify the project:

- In the Project Explorer window, right-click the **Mod2 (Slot 2, NI 9234)** module item and select **Remove from Project**.
- Right-click the **cRIO-9074»Chassis»FPGA Target** item and select **New»C Series Modules**.
- In the Add Targets and Devices on FPGA Target dialog box, select **New target or device**. Select **C Series Module** and click **OK**.
- Set Name to **Mod2**, Type to **NI 9233**, and Location to **Slot 2**, and click **OK**.

4. Modify the target-to-host DMA FIFO to transfer accelerometer data acquired from the NI 9234.

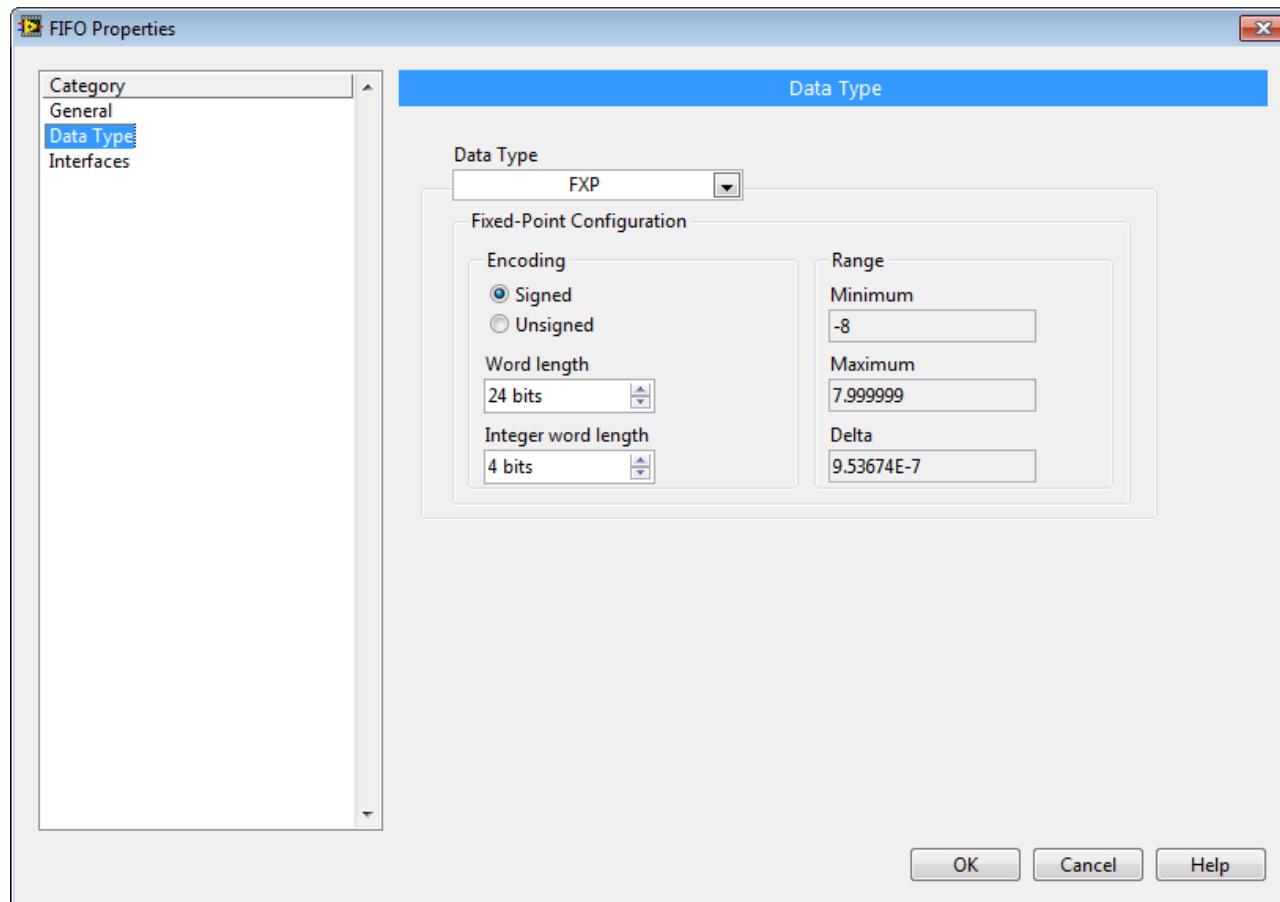
- Double-click **Accelerometer Data** in the Project Explorer window.
- Verify the FPGA FIFO Properties, as shown in Figure 11-18.

Figure 11-18. Accelerometer Data Properties—General



- Select the **Data Type** category. Configure the settings as shown in Figure 11-19.

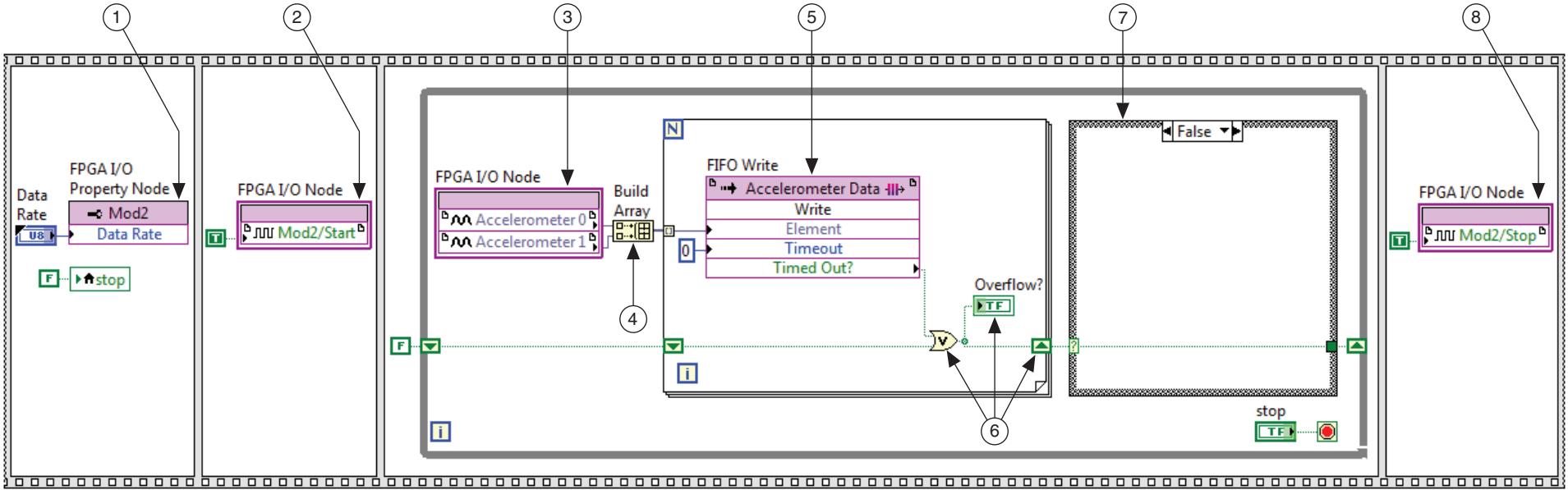
Figure 11-19. Accelerometer Data Properties—Data Type



5. Open `FPGA_Interleaving.vi`.

6. Using the following items, build the block diagram of the FPGA Interleaving VI to match Figure 11-20:

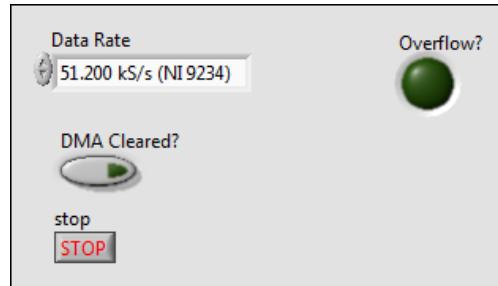
Figure 11-20. FPGA Interleaving VI Block Diagram



- 1 FPGA I/O Property Node—Right-click the node and select **Select Item»FPGA Target»Mod2**. Click **Property** and select **Data Rate**.
- 2 FPGA I/O Node—This node starts data acquisition of the NI 9234. Configure the node to write to Mod2/Start.
- 3 FPGA I/O Node—This node acquires data from two analog input channels. In the Project Explorer window, click **Mod2»Accelerometer 0** and drag it to the block diagram. Expand the node to display two I/O channels. In addition to Accelerometer 0, configure the I/O node to read from Accelerometer 1.
- 4 Build Array—Build an array consisting of data from two accelerometer channels.
- 5 FIFO Write—This FIFO Write method alternates between the two accelerometer channels in the For Loop to interleave the data in a single DMA FIFO. In the Project Explorer window, click and drag Accelerometer Data to the block diagram.
- 6 Or function, Overflow indicator, and Shift Register—The Or function, Overflow indicator, and shift register are used to latch if an overflow condition occurs.
- 7 Case structure for overflow—if an overflow does not occur, the Case structure just passes the overflow Boolean wire through to the shift register. If an overflow occurs, the VI will store a TRUE in the overflow shift register and the VI will not write any data to the DMA FIFO until the host VI sets the DMA Cleared? control to a value of TRUE.
- 8 FPGA I/O Node—This node stops the data acquisition of the NI 9234. Configure the node to write to Mod2/Stop.

7. Arrange the front panel so that it resembles Figure 11-21.

Figure 11-21. FPGA Interleaving Front Panel



8. Save the VI.

9. Test the application on the development computer.

- Set the FPGA target to execute the VI on the development computer with simulated I/O.
- Create an indicator for the output of the Build Array function. This is the data that will be written to the interleaved Accelerometer Data DMA FIFO. You use this indicator to examine the output of the Build Array function. You will delete this indicator later in the exercise.
- Run the VI with simulated I/O.
- Check the values stored in the array indicator that you created. There are two elements in the array, since two channels of data are being acquired and written to the DMA FIFO for each iteration of the While Loop.
- If you allow the VI to continue running, the Overflow? indicator eventually becomes lit. Because the VI is continually writing elements to the DMA FIFO and there is currently nothing reading elements from the DMA FIFO, the DMA FIFO eventually overflows. You will create a RT Host VI to read elements from the DMA FIFO in the next section.
- Stop the VI and remove the array indicator that you created. This indicator was only used for debugging and is no longer needed in the FPGA VI.

10. Save the VI.

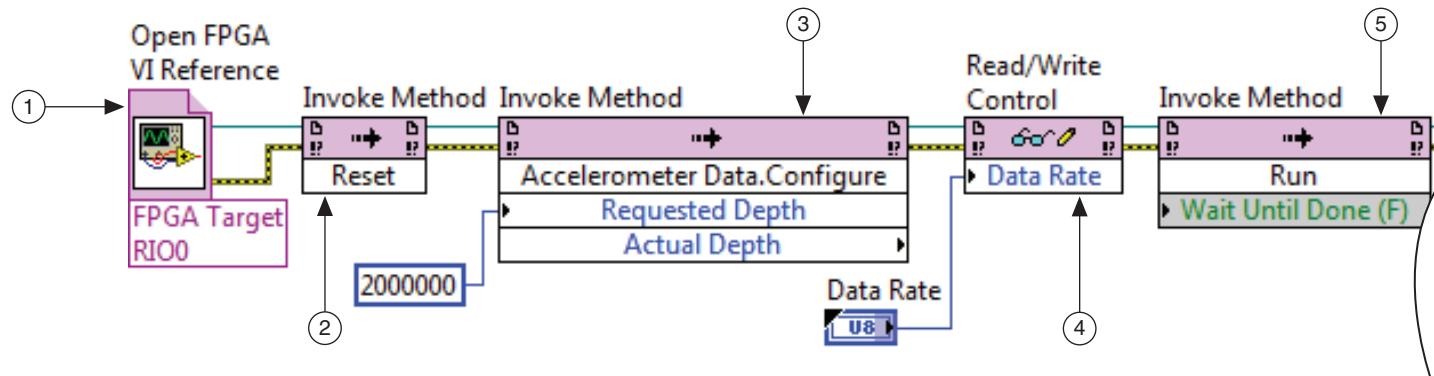
11. Start the compilation of this VI.

- Set the FPGA target to execute the VI on the FPGA.
- Run the VI.

Creating the RT Host VI

1. While the FPGA Interleaving VI compiles, open RT Host Interleaving.vi.
2. Create the left-side of the RT Host Interleaving VI block diagram, as shown in Figure 11-22.

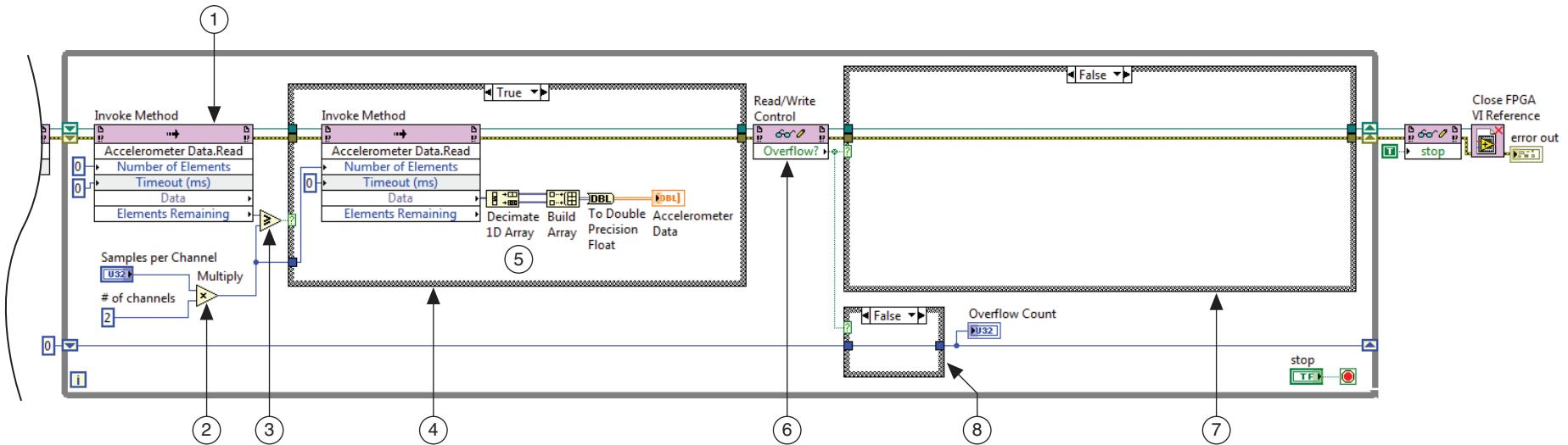
Figure 11-22. RT Host Interleaving VI Block Diagram—Left-Side



- 1 Open FPGA VI Reference—This function loads and opens a reference to the FPGA Interleaving VI.
- 2 Reset method—This method aborts and resets the FPGA VI on the FPGA target to the default state of the VI. To ensure that the DMA FIFOs are empty before running an FPGA VI, you should use the Reset method to clear all DMA FIFOs.
- 3 Accelerometer Data.Configure method—This method configures the number of elements in the host buffer part of the Accelerometer Data DMA FIFO.
- 4 Read/Write Control function—This function sets the value of the Data Rate control on the FPGA VI. Right-click the input and select **Create»Control** to create the Data Rate control. This should create a Data Rate typedef ring control that displays a drop-down list of data rates on the front panel. If this creates a numeric control on the front panel instead, you might need to wait until the FPGA VI has finished compiling before creating the Data Rate control from the input.
- 5 Run method—After the RT Host Interleaving VI calls the Reset method to abort and reset the FPGA VI, configures the host buffer, and sets the Data Rate value, this method runs the FPGA VI on the FPGA target.

3. Modify the right-side of the RT Host Interleaving block diagram to match Figure 11-23:

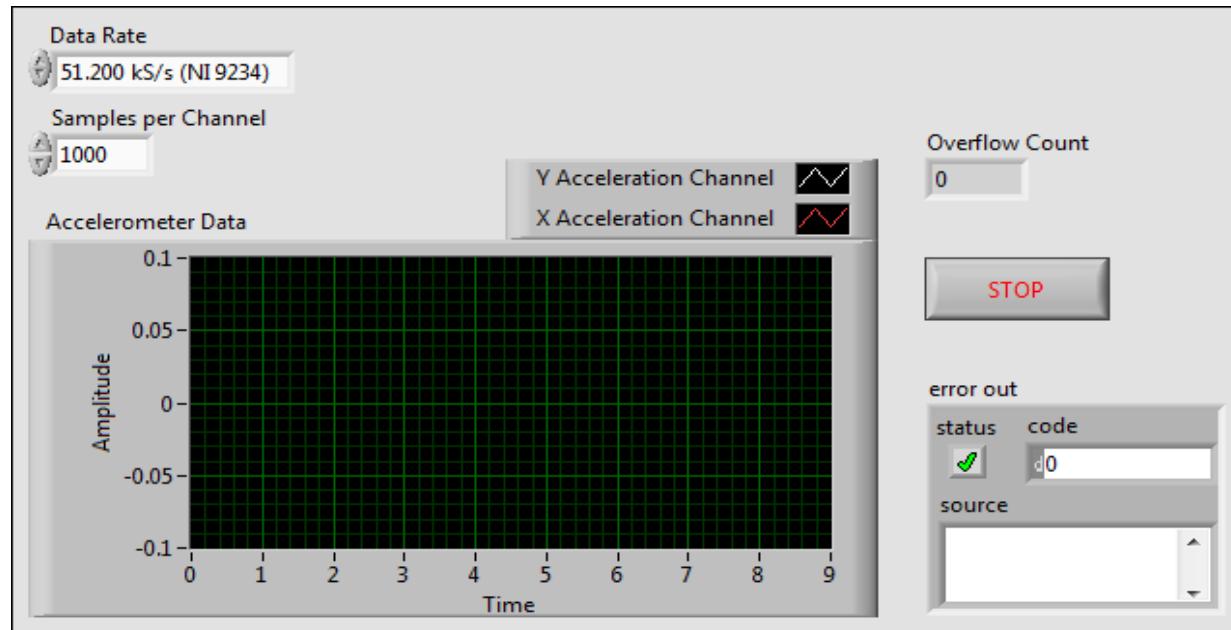
Figure 11-23. RT Host Interleaving Block Diagram—Right-Side



- 1 **Invoke Method (Accelerometer Data.Read)**—Use this method to return the number of elements remaining that are available for reading.
- 2 **Multiply function**—Use this function to determine how many elements to read from the DMA FIFO. Because the DMA FIFO contains two channels of data interleaved together, you must multiply the desired Samples per Channel by 2.
- 3 **Greater or Equal? function**—Use this function to check if desired number of elements are ready to read from the DMA FIFO.
- 4 **Case structure**—If n elements are available to read from the DMA FIFO, this Case structure will use read n elements and de-interleave the data. If n elements are not yet available to read from the DMA FIFO, this Case structure will wait to allow the processor some time to sleep.
- 5 **True case**—Uses the Invoke Method (Accelerometer Data.Read) to read n elements from the DMA FIFO, Decimate 1D Array function to de-interleave the data into two separate arrays that correspond to original Accelerometer 0 and 1 FPGA channels, Build Array function to create a 2D array containing two channels of data to plot, and To Double Precision function to convert the data to double-precision data type. Right-click the output of the To Double Precision function and select **Create»Indicator** and rename it to create the Accelerometer Data indicator to display two channels of data.
- 6 **Read/Write Control function**—Checks if overflow occurred in the DMA FIFO by reading the current value of the Overflow? control on the FPGA VI.
- 7 **Case structure**—Examine the contents. If the overflow does not occur in the DMA FIFO, this Case structure will not add any additional logic. If overflow does occur in the DMA FIFO, then this Case structure clears the DMA FIFO by reading the remaining elements in the DMA FIFO and then notifies the FPGA VI that the DMA is cleared by setting the Overflow? control in the FPGA VI to True.
- 8 **Case structure**—If the overflow does not occur in the DMA FIFO, this Case structure does not change the Overflow Count. If overflow does occur in the DMA FIFO, then this Case structure increments the Overflow Count.

4. Modify the front panel so that it matches Figure 11-24. Develop the front panel using the following items:

Figure 11-24. RT Host Interleaving Front Panel



- Waveform Graph—Replace Accelerometer Data with a waveform graph to better display the data transferred.
 - Right-click on Accelerometer Data and select **Replace»Graph»Waveform Graph**.
 - Resize the Plot Legend to show two plots. Rename the plots as Y Acceleration Channel and X Acceleration Channel.
 - Right-click the waveform graph and deselect **Y Scale»AutoScale Y**. Double-click the minimum and maximum numbers on the y-axis and set them to -0.1 and 0.1, as shown in Figure 11-24.

5. Save the VI.

Testing

1. Test the application on the FPGA target.
 - In the RT Host Interleaving VI, enter the following values in the front panel controls:
 - Data Rate: If using NI 9234, select 51.200 kS/s. If using NI 9233, select 50.000 kS/s.
 - Samples per Channel: 1000
 - Run the RT Host Interleaving VI.
 - On the Sound & Vibration Signal Simulator, set the switch to UNBALANCED FAN and set the Fan Speed Control switch to DIAL. Turn the dial to increase the fan speed, and you should see the graph update with data from two accelerometer channels. You used an interleaved DMA FIFO to transfer this data from the FPGA VI to the host VI.
 - Click the stop button on the front panel. In this application, this stops the host VI and FPGA VI.
2. Test overflow behavior.
 - On the block diagram of the RT Host Interleaving VI, change the constant wired to the Accelerometer Data.Configure method from 2000000 to 5. This changes the host buffer part of the DMA FIFO to 5 elements. Do you expect an overflow? Why or why not?

This changes the host buffer part of the DMA FIFO to 5 elements. Do you expect an overflow? Why or why not?
 - Run the RT Host Interleaving VI.

You should see the Overflow Count indicator start incrementing. The DMA FIFO keeps getting an overflow because the FPGA VI is writing to the DMA FIFO at a rate of 51,200 elements per second (or 50,000 elements/s) and the host buffer of 5 elements is not large enough to help the host VI read elements before the DMA FIFO overflows.
 - Click the stop button on the front panel.
 - Change the constant wired to the Accelerometer Data.Configure method back to 2000000.
3. Save and close the VI when finished.

End of Exercise 11-4

Alternate CompactRIO Controller Instructions

This appendix contains complementary instructions that you use to modify course exercises to use an alternate CompactRIO controller and chassis.

Topics

- A. Using an Alternate CompactRIO Controller/Chassis
- B. Hardware Setup
- C. Hardware Configuration
- D. Creating a New LabVIEW FPGA Project
- E. Modifying an Existing LabVIEW FPGA Project

A. Using an Alternate CompactRIO Controller/Chassis

Many of the course exercises were designed for use with the CompactRIO 9074 integrated controller. However, depending on hardware availability, your course might use an alternate CompactRIO controller and chassis.

Course exercises that use CompactRIO hardware require a chassis or integrated controller with a Spartan 3 or Virtex 5 FPGA target. Use the instructions in the following sections to modify course exercises for alternate hardware.

B. Hardware Setup

The hardware setup steps are the same for your alternate controller.



Note There might be slight differences the CompactRIO controller connectors, but the connections are to the same connectors as noted in the instructions.

C. Hardware Configuration

The hardware configuration steps are the same except for the name you specify for your Network Settings in MAX. Provide a name that is appropriate for your model of controller. For example, instead of naming your controller cRIO-9074, as noted in the instructions, use cRIO-9012, or MyController.



Note The name cannot include spaces.

D. Creating a New LabVIEW FPGA Project

The instructions for creating a new LabVIEW FPGA project for your CompactRIO controller are the same with the following exceptions:

- Select your controller name in the Add Targets and Devices dialog box. For example, instead of **cRIO-9074**, you should select **cRIO-9012**.

After LabVIEW discovers and creates a Real-Time target with your C Series modules, your project hierarchy has the following differences:

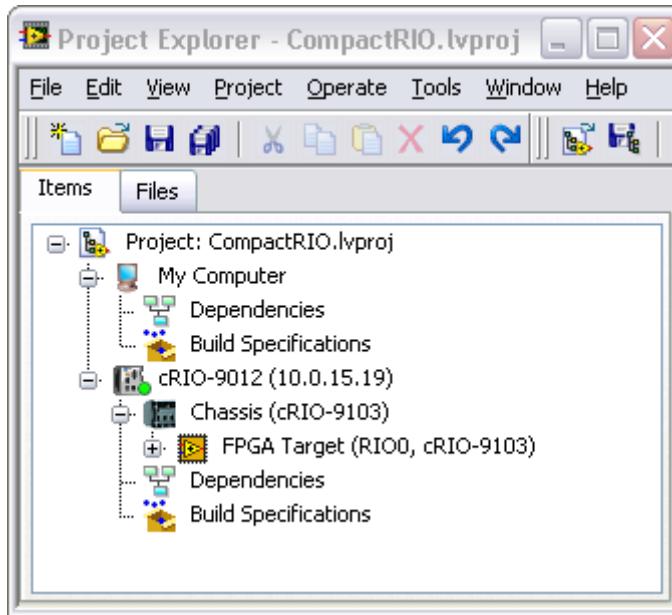
- Real-Time target name is the name you configured in MAX. For example, instead of cRIO-9074, the name might be cRIO-9012.
- Chassis name is the model name of your chassis. For example, instead of cRIO-9074, the name might be cRIO-9103.
- FPGA target name includes the model name of your chassis. For example, instead of cRIO-9074, the name might be cRIO-9103.
- Support of FPGA items varies by FPGA target. For example, some targets support a scan clock.



Note If you have an integrated controller, the chassis name will be the same as your controller.

Refer to Figure A-1 to see a LabVIEW FPGA project created for a cRIO-9012 controller and a cRIO-9103 chassis.

Figure A-1. LabVIEW FPGA Project for cRIO-9012 and cRIO-9103



E. Modifying an Existing LabVIEW FPGA Project

Several of the exercises begin with a pre-built project. Similarly, solution projects are also pre-built. These pre-built projects were created for the cRIO-9074. You must modify these projects for use with your alternate hardware.



Note You must follow similar steps to modify shipping CompactRIO examples.

1. Open the solution or exercise project.
2. Create a new target for your alternate controller. LabVIEW auto-discovers your chassis and modules.
3. Expand both the original pre-built FPGA target and your new FPGA target.

4. Rename any I/O nodes modified under the pre-built FPGA target.

- Identify renamed I/O nodes under the pre-built FPGA target. For example, if Mod2/AI0 was renamed to be Accelerometer 0, it appears in the project explorer as Accelerometer 0 (Mod2/AI0).
- Rename nodes under new FPGA target by right-clicking the node and selecting **Rename**. Enter the same name used in the pre-built FPGA target.



Note You can also drag nodes from the pre-built FPGA target to the new target. After copying the nodes into the new project, you will see a red exclamation mark indicating that there are duplicate nodes. Remove duplicate unnamed nodes by right-clicking the unnamed node and selecting **Remove from Project**.

5. Select the following items under the original pre-built FPGA target and drag them to your new FPGA target:

- FIFOs, memory, derived clocks and other non-I/O nodes
- FPGA VI and any subVIs

6. If the project includes a host VI, you must update the FPGA VI reference. To update the reference, right-click the Open FPGA VI Reference node, and select **Configure Open FPGA VI Reference** to open the Configure Open FPGA VI Reference dialog. Click the Browse button and select the FPGA VI under the new target to update the path. Click **OK**.

7. In the Project Explorer, make note of the pre-built FPGA target name.

8. Remove the unused FPGA target by right-clicking and selecting **Remove from Project**.



Note If the VIs are copied before renaming I/O nodes, the reference to the I/O connector is no longer valid and will result in a broken Run arrow. You must manually re-establish this connection with the I/O node.

9. Rename the new FPGA Target to match the name of the pre-built FPGA Target.

Course Slides

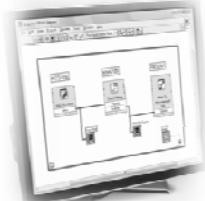
This appendix contains the Course Slides.

Topics

1. Introduction to LabVIEW FPGA Programming Basics
2. LV FPGA Basics
3. FPGA Programming Basics
4. FPGA I/O
5. Timing an FPGA VI
6. Single-Cycle Timed Loops
7. Signal Processing
8. Data Sharing on FPGA
9. Synchronizing FPGA Loops and I/O
10. Communicating Between the FPGA and Host
11. Modular Programming



What You Need To Get Started



- LabVIEW FPGA Course Manual
- LabVIEW FPGA Exercise Manual
- LabVIEW FPGA Course CD
- NI Sound and Vibration Signal Simulator
- Two J-type thermocouples
- cRIO-9074 Integrated chassis and controller
- NI 9211 thermocouple module
- NI 9234 analog input module
- NI 9263 analog output module

Computer running Windows 7/Vista/XP with the following software installed:
▪ LabVIEW 2012 or later
▪ LabVIEW Real-Time 2012 or later
▪ LabVIEW FPGA 2012 or later
▪ NI-RIO 12.0 or later

NATIONAL INSTRUMENTS | ni.com/training

File Locations

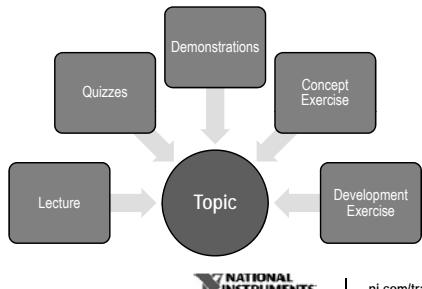


The course installer places the course files in the following location:

```
graph LR; Root[Root Directory] --> Exercises[Exercises <br> <or> Solutions]; Exercises --> LabViewFPGA[LabVIEW FPGA]
```

NATIONAL INSTRUMENTS | ni.com/training

Instructional Methods



NATIONAL
INSTRUMENTS

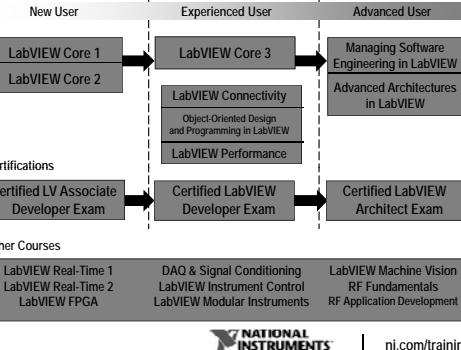
| ni.com/training

Getting The Most Out Of This Course

- Ask questions!
- Experiment with hands-on exercises to understand the methods used
- Explore solutions
- Implementations explore a possible solution—you may find a better one

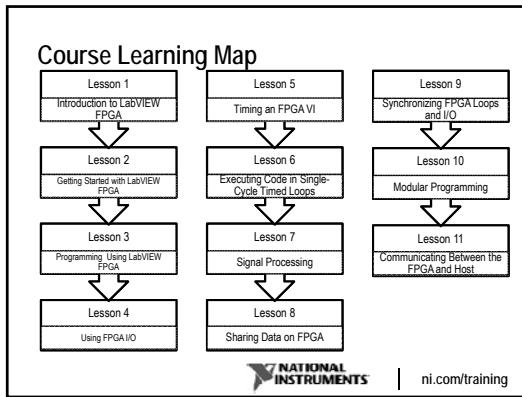
NATIONAL
INSTRUMENTS

| ni.com/training



NATIONAL
INSTRUMENTS

| ni.com/training



Course Goals

This course prepares you to:

- Select and configure NI Reconfigurable I/O (RIO) hardware
- Create, compile, download, and execute a LabVIEW FPGA VI and use NI RIO hardware
- Perform measurements using analog and digital input and output channels
- Create host computer programs that interact with FPGA VIs
- Control timing, synchronize operations, and implement signal processing on the FPGA
- Design and implement applications using the LabVIEW FPGA module



ni.com/training

Configuring Your LabVIEW Environment

- Options Dialog Box
 - Controls/Functions Palettes page
 - Select Load palettes during launch to make Search Palettes immediately usable after launch
 - Set Palette to Category (Icons and Text)
 - Block Diagram page
 - Uncheck Place front panel terminals as icons to place control and indicator terminals in a compact format
 - Configure Block Diagram Cleanup to customize your block diagram



ni.com/training

Configuring Your LabVIEW Environment

- Functions Palette
 - Tack the Functions palette and select View»Change Visible Categories then click Select All
- Controls Palette
 - Tack the Controls palette and select View»Change Visible Categories then click Select All



ni.com/training

Lesson 1 Introduction to LabVIEW FPGA

TOPICS

- A. Introduction to FPGA Technology
- B. Components of a LabVIEW FPGA System
- C. Comparison with a DAQmx System
- D. Examples of LabVIEW FPGA Applications



ni.com/training

A. Introduction to FPGA Technology

What is an FPGA?

- Field programmable gate array (FPGA)
- A silicon chip with unconnected gates and other hardware resources
- Enables user to define and re-define functionality

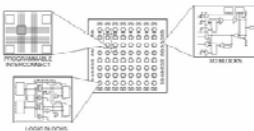


ni.com/training

Introduction to FPGA Technology

How does an FPGA work?

- Circuit behavior is defined using software
- Circuit specification (gate connection, etc.) is loaded into the hardware
- No operating system is needed for execution of logic



ni.com/training

Introduction to FPGA Technology

When is an FPGA used?

- Fast I/O response times and specialized functionality
- Custom hardware, where it does not make sense to pay the high price of developing an ASIC
- Rapid prototyping and verification without the fabrication process of custom ASIC design
- Implementing custom functionality with the reliability of dedicated deterministic hardware
- Field-upgradable after deployment



ni.com/training

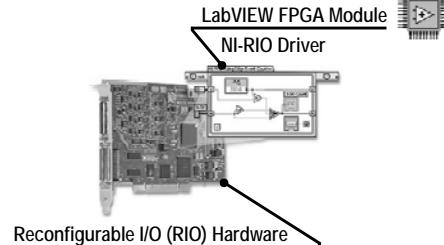
Benefits of FPGA

- Flexibility
 - Reconfigurable through software
- True parallel processing
 - Simultaneous parallel circuits
 - No CPU time sharing
- High Performance
- Reliability
- Offload processing
- Cost



ni.com/training

B. Components of a LabVIEW FPGA System



NATIONAL
INSTRUMENTS

| ni.com/training

LabVIEW FPGA Module

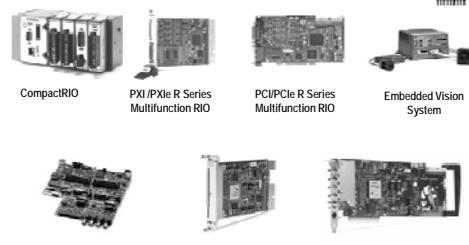


- Add-on module for LabVIEW
- Develop VIs for FPGA target
- Develop VIs for host PC or Real-Time interaction with FPGA

NATIONAL
INSTRUMENTS

| ni.com/training

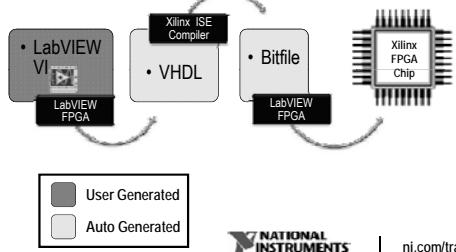
LabVIEW FPGA Targets



NATIONAL
INSTRUMENTS

| ni.com/training

LabVIEW FPGA: How does it work?



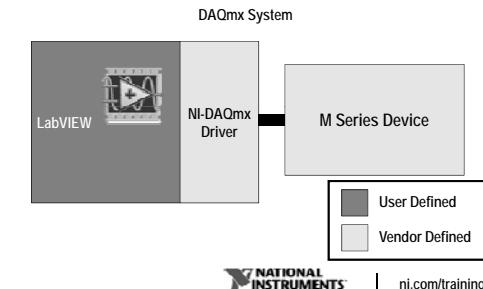
Benefits of a LabVIEW FPGA System

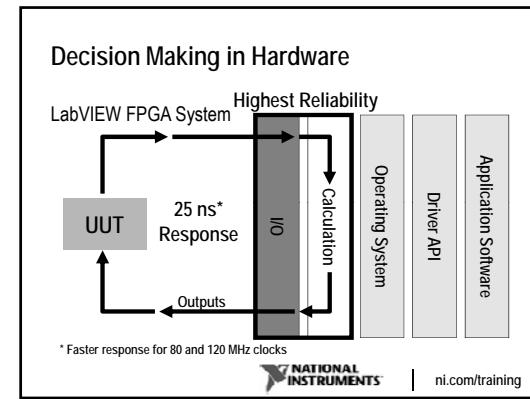
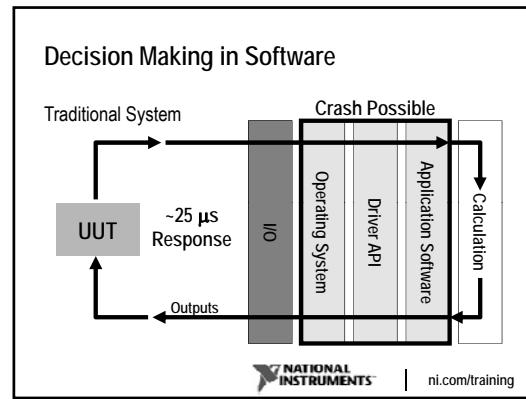
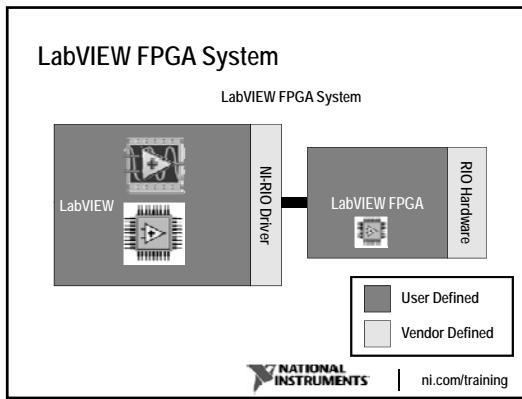
- FPGA technology
- No Verilog/VHDL coding or board design
- Onboard decision making
- Direct access to hardware terminals
- Extensive library of built-in functions
- Integration with 3rd party IP
- Tools to communicate, monitor and control the FPGA from Windows PC or Real-Time controller



ni.com/training

C. Comparison with NI-DAQmx System





D. Examples of LabVIEW FPGA Applications

Typical LabVIEW FPGA applications

- Intelligent DAQ
 - Custom Timing and Synchronization
 - Custom Counters
 - Multiple Scan Rates
- Ultra-high speed control
- Specialized communication protocols
- Off-load CPU – Processing
- Complex timing and synchronization
- Hardware-in-the-Loop (HIL) testing



| ni.com/training

Summary—Quiz

1. Which of the following are required software components of a LabVIEW FPGA system?
 - a. NI-RIO driver
 - b. DAQmx driver
 - c. LabVIEW Development System
 - d. LabVIEW Real-Time Module
 - e. LabVIEW FPGA Module



| ni.com/training

Summary—Quiz Answer

1. Which of the following are required software components of a LabVIEW FPGA system?
 - a. NI-RIO driver
 - b. DAQmx driver
 - c. LabVIEW Development System
 - d. LabVIEW Real-Time Module
 - e. LabVIEW FPGA Module



| ni.com/training

Summary—Quiz

2. What OS runs on the FPGA?
 - a. LabVIEW FPGA
 - b. MicroLinux
 - c. Unix
 - d. VxWorks
 - e. None of the above



| ni.com/training

Summary—Quiz Answer

2. What OS runs on the FPGA?
 - a. LabVIEW FPGA
 - b. MicroLinux
 - c. Unix
 - d. VxWorks
 - e. None of the above



| ni.com/training

Summary—Quiz

3. LabVIEW FPGA allows you to program an FPGA without knowing VHDL or HDL.
 - a. True
 - b. False



| ni.com/training

Summary—Quiz Answer

3. LabVIEW FPGA allows you to program an FPGA without knowing VHDL or HDL.
 - a. True
 - b. False



| ni.com/training

Summary—Quiz

4. Which of the following might be typical reasons for using R Series Multifunction RIO device over a DAQ board programmed with DAQmx?
 - a. Need more counters than available on a DAQ board
 - b. Need custom trigger logic
 - c. Need more analog input channels than available on a DAQ board
 - d. Need to simultaneously acquire data on different channels at different rates
 - e. Need a high-level/easy-to-use driver



| ni.com/training

Summary—Quiz Answers

4. Which of the following might be typical reasons for using R Series Multifunction RIO device over a DAQ board programmed with DAQmx?
 - a. Need more counters than available on a DAQ board
 - b. Need custom trigger logic
 - c. Need more analog input channels than available on a DAQ board
 - d. Need to simultaneously acquire data on different channels at different rates
 - e. Need a high-level/easy-to-use driver



| ni.com/training

Lesson 2 Getting Started with LabVIEW FPGA

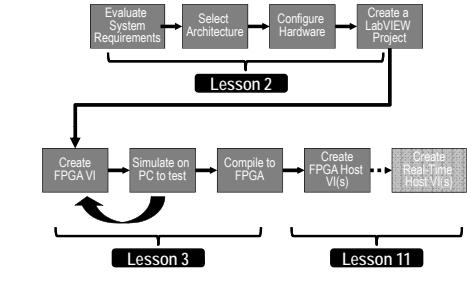
TOPICS

- A. Evaluating System Requirements
- B. Configuring Your System
- C. Creating a LabVIEW FPGA Project



ni.com/training

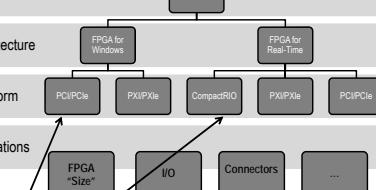
FPGA Development Flow



ni.com/training

A. Evaluating System Requirements

Need FPGA?



ni.com/training

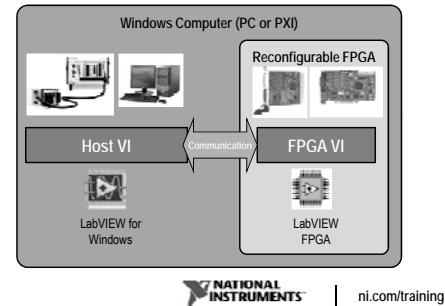
FPGA System Architectures

- LabVIEW FPGA Architecture on Windows
- LabVIEW FPGA Architecture with LabVIEW Real-Time



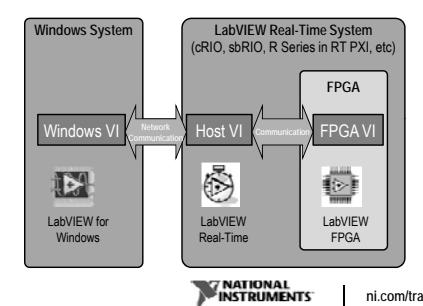
ni.com/training

FPGA – Windows



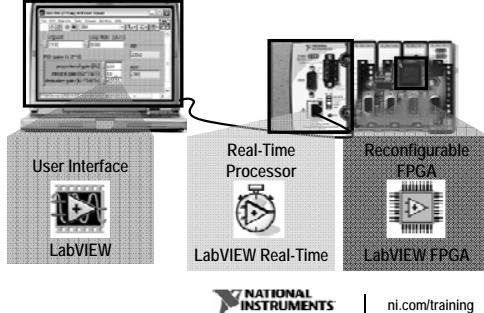
ni.com/training

FPGA – Real-Time



ni.com/training

LabVIEW Programming & CompactRIO



Common Reconfigurable I/O (RIO) Platforms

- PCI, PCI Express
- PXI, PXI Express
- CompactRIO



ni.com/training

R Series Multifunction RIO

- PCI/PCIe/PXI/PXIe form factor
- Define custom data acquisition
- Integrate FPGA with analog and digital I/O
- High channel count

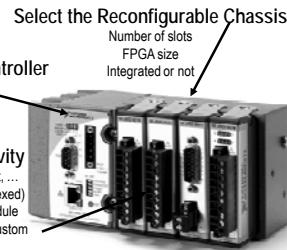


ni.com/training

Building a CompactRIO System

Select the Real-Time Controller

Processor speed
Flash memory size



Select the I/O and Connectivity
Analog Input/Output, Digital Input/Output, ...
(Low / high speed, simultaneous / multiplexed)
Standard connector depends on the module
Options: Strain Relief Connector Block, Custom
Cable, etc.



ni.com/training

Other CompactRIO Form Factors



NI CompactRIO
Integrated Controller



NI Single-Board RIO



ni.com/training

FPGA "Size"

- Historically, FPGA size was measured by "gate count"
 - The "larger" FPGA had more resources than a "smaller" FPGA
- Now, FPGA logic blocks and resources differ so logic capacity and performance comparisons are tricky
 - FPGAs include a variety of resources and logic blocks
 - RAM, Multipliers, Flip-Flops, Lookup Tables (LUTs)
 - Logic blocks differ in capabilities
 - 4-input LUT vs. 6-input LUT
- Difficult to map LabVIEW VIs into FPGA resources
 - Xilinx compiler optimizes code based on FPGA architecture
 - Verify that code fits by compiling for offline hardware targets
 - Resource estimation takes significantly less time than compilation



ni.com/training

FPGA Resource Table

Example Xilinx FPGA Chips	Virtex II 1000	Spartan 3 1000	Virtex 5 LX30	Virtex II 3000	Virtex 5 LX50	Spartan 3 2000	Virtex 5 LX85	Virtex 5 LX10
Equivalent Logic Cells	11,520	17,280	30,720	32,256	46,080	46,080	82,944	110,592
LUTs/FFs	10,240	15,360	19,200	28,672	28,800	40,960	51,840	69,120
Multiplicators	40	24	32	96	48	40	48	64
Block RAM (Kb)	720	432	1,152	1,728	1,728	720	3,456	4,608
Sample NI Products	cRIO-9101	cRIO-9072	cRIO-9111 PXI-7851R	cRIO-9104 PXI-7833R	cRIO-9113 PXI-7852R	cRIO-9074 PXI-7853R	cRIO-9116 PXI-7854R	

Refer to <http://zone.ni.com/devzone/cda/tut/p/id/7440> for more information.



ni.com/training

Exercise 2-1: Set up a CompactRIO System

Connect and power up a CompactRIO system.

GOAL

Exercise 2-1: Set up a CompactRIO System

- If you need to swap modules, do you first need to power off the CompactRIO system?
- Both the FPGA and USER1 LEDs are user-configurable. What do you think are their differences?

DISCUSSION

B. Configuring Your System

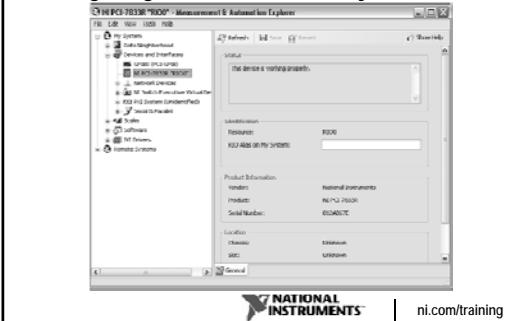
Before installing any device, install the following on the development computer:

1. Install LabVIEW
2. If using a Real-Time controller, install LabVIEW Real-Time
3. Install LabVIEW FPGA
4. Install NI-RIO driver or other target-specific software



ni.com/training

Configuring a Windows FPGA System



Configuring a Real-Time FPGA System

Select Host-Target Network Setup

Detect the Remote Target

Configure Target Network Settings

View Devices and Interfaces

Add/Remove Software



ni.com/training

Configuring a Real-Time FPGA System

Select Host-Target Network Setup



Detect the Remote Target



Configure Target Network Settings



View Devices and Interfaces



Add/Remove Software



| ni.com/training

Host-Target Network Setup

- Options:

- Host and Target connected to Local Area Network
 - Typically uses DHCP
- Host and Target connected directly using a crossover cable
- Host and Target need an IP Address
- To configure remote system
 - Host and Target need to be on the same subnet
 - Might need to temporarily disable your Firewall



| ni.com/training

Configure Network Settings

- IP Address – the unique address of a device
 - a set of four one- to three-digit numbers in the range 0–255
 - dotted decimal notation.
 - example IP address: 224.102.13.24
- Subnet Mask – a code that helps the network device determine whether another device is on the same network.
 - 255.255.255.0 is the most common subnet mask
- Gateway (Optional) – address of a gateway server (a connection between two networks)
- DNS Address (Optional) – address of a device that stores DNS host names and translates them into IP addresses



| ni.com/training

IP Addressing Options

Option	Description
DHCP	<ul style="list-style-type: none">A network that has a DHCP server may automatically assign an IP to the target each time the target boots upCommon network protocol for administering IP addresses
Link-local	<ul style="list-style-type: none">Link-local addresses are network addresses intended for use in a local network onlyAutoIP attempts to assign an address using DHCP first, then link-local if DHCP failsLink-local address range: 169.254.x.x
Static IP	<ul style="list-style-type: none">Manually assigned IP address



ni.com/training

IP Addressing Options

Option	Windows Host	Real-Time Controller
DHCP	Default	Default
Link-local	Default if fail to connect to DHCP	Assigned automatically using AutoIP* or configurable in MAX.
Static IP	Configurable in Windows	Configurable in MAX

* AutoIP available with some controllers and/or versions of Real-Time.



ni.com/training

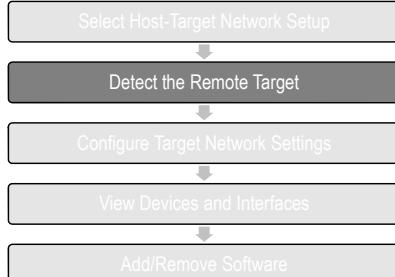
Ethernet Cabling Options

Option	Use when...
Standard	Host and target are connected with hub, router, or network switch
Crossover	Directly connecting host and target



ni.com/training

Configuring a Real-Time FPGA System



ni.com/training

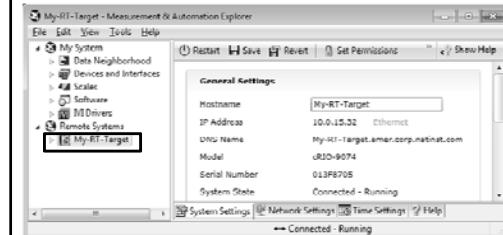
Detect Remote Target - CompactRIO

1. Connect CompactRIO to host PC through a network
2. Power up CompactRIO
3. Not configured CompactRIO – Status light blinks slowly
4. Open MAX (Start»Programs»National Instruments)
5. Remote Systems in MAX– Devices connected over the Ethernet

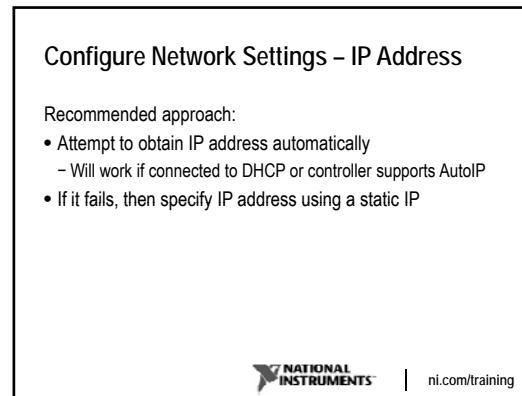
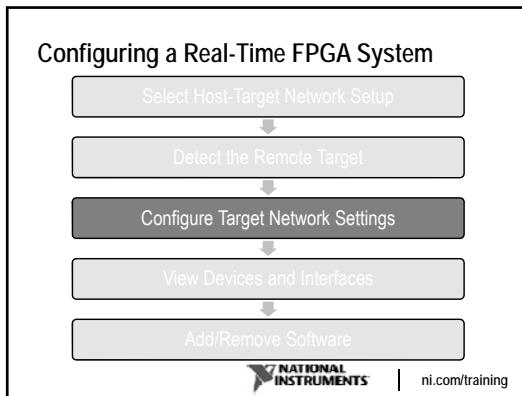


ni.com/training

Detect the Remote Target



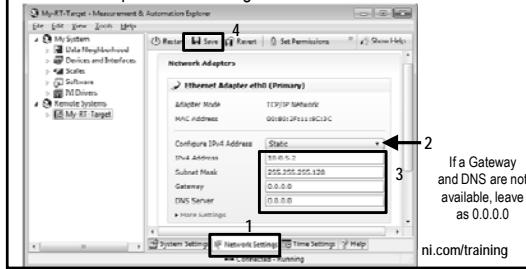
ni.com/training



Manually Specify an IP Address – Static IP

Select Static IP Address Configuration for RT Target

- Host computer and RT target must be in the same subnet



Configuring a Real-Time FPGA System

Select Host-Target Network Setup

Detect the Remote Target

Configure Target Network Settings

View Devices and Interfaces

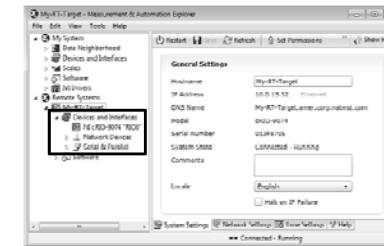
Add/Remove Software



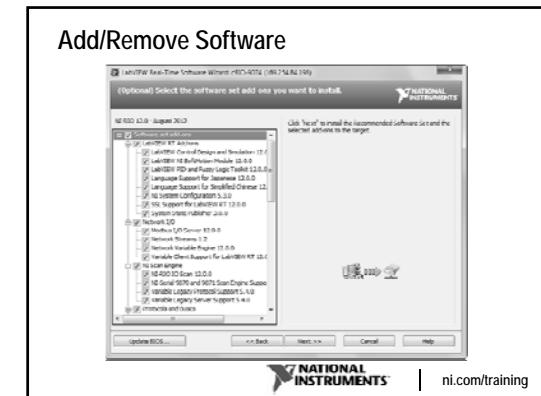
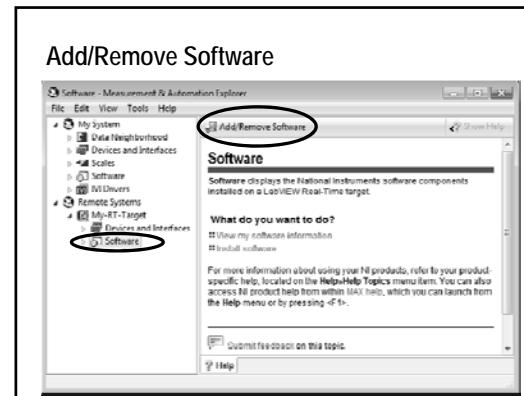
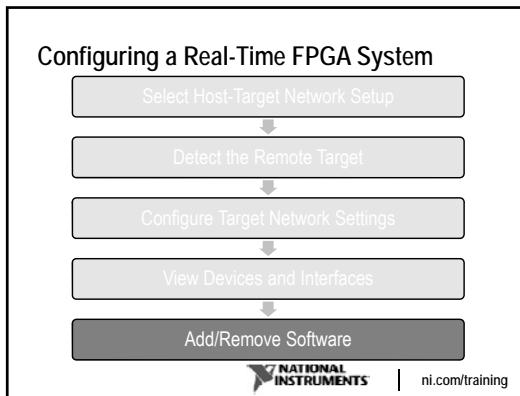
ni.com/training

View Devices and Interfaces

Expand Devices and Interfaces to view the chassis



ni.com/training



Exercise 2-2: Configure a CompactRIO System

Use MAX to configure the CompactRIO.

GOAL

Exercise 2-2: Configure a CompactRIO System

- Why is it important to keep software synchronized between host PC and CompactRIO controller?

DISCUSSION

C. Creating a LabVIEW FPGA Project

Steps to create a LabVIEW FPGA Project:

1. Create a new project
2. Add a target
 - Add under "My Computer" if device is local
 - Add under "Project" if device is networked
3. Discover existing device or create a new device
 - For online devices, LabVIEW can also discover module and I/O nodes
 - For offline devices, add modules and I/O nodes
4. If device is a networked device, connect to the target



| ni.com/training

New FPGA Target – Windows

- For local targets:
- Right-click My Computer and select New>Target and Devices



ni.com/training

Add Targets and Devices – Windows

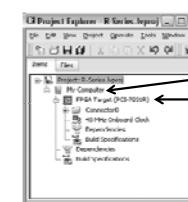
- Existing target or device
 - Use when your device can be accessed from your development computer.
- New target or device
 - Use when you are not connected with a physical target or device is not present.



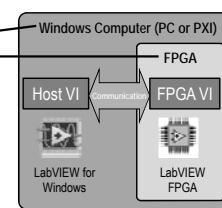
ni.com/training

Windows Project

FPGA Target is under My Computer

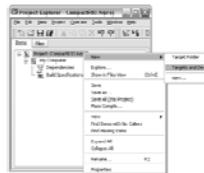


ni.com/training



New Target – RT Target

- For remote targets:
- Right-click on Project and select New>Target and Devices



ni.com/training

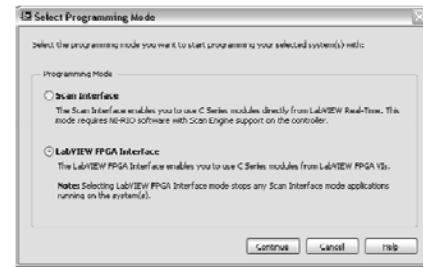
Add Targets and Devices – RT Target

- Existing target or device
 - Use when your device can be accessed from your development computer.
- New target or device
 - Use when you are not connected with a physical target or device is not present.



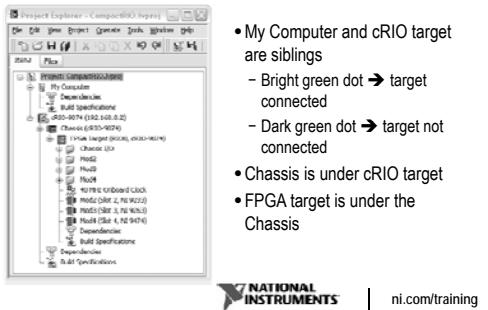
ni.com/training

Select Programming Mode - CompactRIO



ni.com/training

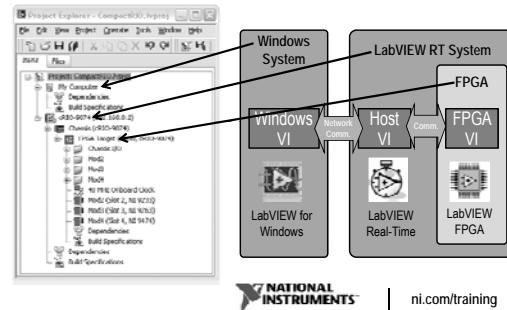
Project with CompactRIO Controller



NATIONAL INSTRUMENTS

ni.com/training

Project with CompactRIO Controller



NATIONAL INSTRUMENTS

ni.com/training

Exercise 2-3 & 2-4: Create Two LabVIEW FPGA Projects

Create LabVIEW FPGA projects for the following two devices:

1. An offline PCIe-7852R
2. An online CompactRIO 9074

GOAL

Exercise 2-3 & 2-4:
Create Two LabVIEW FPGA Projects

- Is it possible to create a simulated project for CompactRIO?

DISCUSSION

LabVIEW FPGA Project Template

DEMONSTRATION

Summary—Quiz

1. Which of the following operations are performed in MAX?
 - a. Install software on an R Series board
 - b. Install software on a CompactRIO target
 - c. Verify connection between host and a real-time target
 - d. Assign the IP address of your host computer



| ni.com/training

Summary—Quiz Answer

1. Which of the following operations are performed in MAX?
 - a. Install software on an R Series board
 - b. Install software on a CompactRIO target
 - c. Verify connection between the host and a real-time target
 - d. Assign the IP address of your host computer

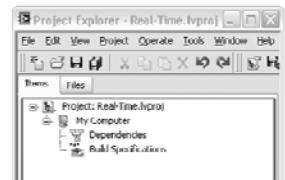


ni.com/training

Summary—Quiz

2. Under which item do you add a CompactRIO target?

- a. Real-Time.lvproj
- b. My Computer
- c. Dependencies
- d. Build Specifications

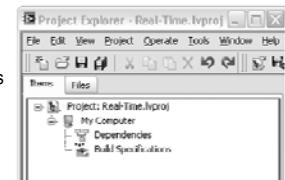


ni.com/training

Summary—Quiz Answer

2. Under which item do you add a CompactRIO target?

- a. Real-Time.lvproj
- b. My Computer
- c. Dependencies
- d. Build Specifications



ni.com/training

Summary—Quiz

3. When would you select New target or device?

- a. You are evaluating an FPGA target to see if your application can run on the target
- b. You are developing your application while traveling and you don't have an FPGA target connected to your laptop
- c. You are curious to see which NI-RIO targets and devices are supported under My Computer
- d. All of the above



ni.com/training

Summary—Quiz Answer

3. When would you select New target or device?

- a. You are evaluating an FPGA target to see if your application can run on the target
- b. You are developing your application while traveling and you don't have an FPGA target connected to your laptop
- c. You are curious to see which NI-RIO targets and devices are supported under My Computer
- d. All of the above



ni.com/training

Lesson 3
Programming Using LabVIEW FPGA

TOPICS

- A. Introduction
- B. Developing the FPGA VI
- C. Interactive Front Panel Communication
- D. Selecting an Execution Mode
- E. Compiling the FPGA VI
- F. Basic Optimizations

NATIONAL INSTRUMENTS | ni.com/training

A. Introduction

FPGA Layout and Components

The diagram illustrates the internal structure of an FPGA. It shows a central 'Configurable Logic Block (CLB)' containing various logic elements and interconnects. This is connected to an 'IO Block' on the right, which provides interface pins. A 'Programmable Interconnect' network routes signals between the CLB and the IO Block. Labels point to each component: 'PROGRAMMABLE INTERCONNECT', 'IO BLOCK', and 'CONFIGURABLE LOGIC BLOCK (CLB)'. Below the diagram, the National Instruments logo and the URL 'ni.com/training' are present.

How an FPGA Works

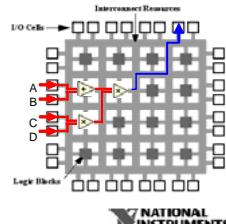
- Programmable interconnect switches and wires route signals between various hardware resources in an FPGA
- Hardware resources include logic gates, flip-flops, and block memory

For a more detailed description of how FPGA works, see http://www.ni.com/fpga_technology/.

NATIONAL INSTRUMENTS | ni.com/training

How FPGA Works - Example

Implements a VI that calculates a value for F from inputs A, B, C, and D, where $F = (A + B) \times C \times D$



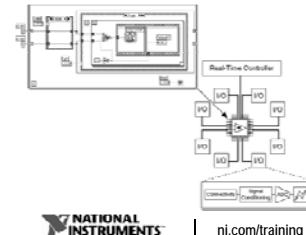
NATIONAL INSTRUMENTS

ni.com/training

Programming FPGA with LabVIEW

LabVIEW FPGA

- Do not have to learn VHDL or Verilog
- True parallel execution
- Deterministic

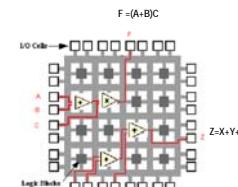


NATIONAL INSTRUMENTS

ni.com/training

True Parallel Execution

$F = (A+B)C$ and $Z = X+Y+M$ in separate gates on an FPGA

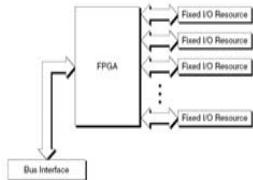


NATIONAL INSTRUMENTS

ni.com/training

FPGA Communication with I/O and Host

- FPGA provides:
 - Timing
 - Triggering
 - Processing
 - Custom I/O
- Each fixed I/O uses a portion of the FPGA hardware resources
- The bus interface to the host computer also uses a portion of the FPGA hardware resources



ni.com/training

Timing Benefits of FPGA

- Multi-loop analog PID loop rates exceed 100 kHz on embedded RIO FPGA hardware whereas they run at 30 kHz in real-time without FPGA hardware
- Digital control loop rates can execute up to 1 MS/s or more depending on the target
- Single-cycle timed loops execute up to 200 MHz or more depending on the target and clock configuration
- Due to parallel processing ability, adding additional computation does not necessarily reduce the speed of the FPGA application



ni.com/training

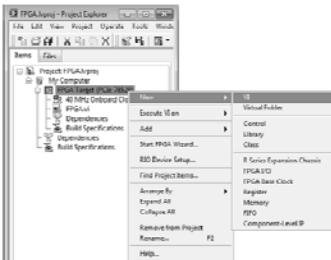
B. Developing the FPGA VI

- There is no operating system on the FPGA
- Download and run only one top-level VI at a time
- FPGA can run independently of the host
- FPGA can store data in memory
- Edit VI under an FPGA Target to use the FPGA palette
- Usually use integer or fixed-point math to perform calculations



ni.com/training

Add a VI Under the FPGA Target



NATIONAL INSTRUMENTS

ni.com/training

FPGA Palettes

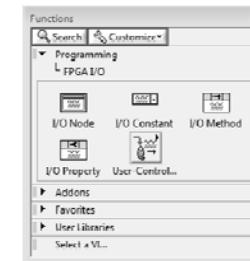
- VIs under the FPGA target inherit the FPGA Function Palette
- Many palettes are similar to LabVIEW for Windows
- FPGA-specific palettes
 - FPGA I/O
 - Memory & FIFO
 - Synchronization
 - FPGA Math & Analysis
 - IP Integration



NATIONAL INSTRUMENTS

ni.com/training

FPGA I/O Palette



NATIONAL INSTRUMENTS

ni.com/training

FPGA Math & Analysis Palette

- Keep calculations as simple as possible to preserve FPGA resources
- Functions perform point-by-point calculations



ni.com/training

Demonstration 3-1

Create an FPGA VI and explore the Functions palette supported under FPGA.

DEMONSTRATION

Top-Level FPGA VI Front Panel

- Hardware resources inside an FPGA are limited
- Use the minimum necessary controls and indicators for programmatic front panel communication to a host
- Add temporary controls and indicators for debugging if necessary, but remove them afterwards



ni.com/training

C. Interactive Front Panel Communication

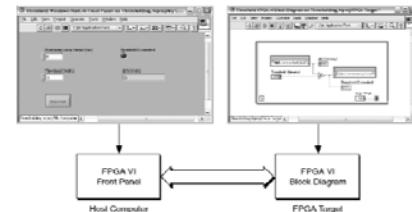
FPGA has no user interface

- Must communicate data from FPGA to Host PC or rely exclusively on I/O
- Front panel displayed on Host PC
- Block diagram executes on FPGA as compiled
- Communication layer shares all control and indicator values
- Cannot use debugging tools when running FPGA VI
 - Test with development computer first, or add indicators as probes



| ni.com/training

Interactive Front Panel Communication



| ni.com/training

D. Selecting an Execution Mode

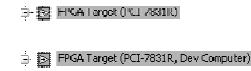
- Right-click the FPGA Target in the Project Explorer window and select Properties to launch FPGA Target Properties dialog box
- Or
- Right-click the FPGA Target and select Execute VI on to select mode directly



| ni.com/training

Selecting an Execution Mode – Options

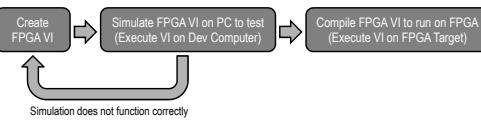
- Execute VI on FPGA target
- Execute VI on Development Computer with Simulated I/O
 - Use Random Data for FPGA I/O Read
 - Use Custom VI for FPGA I/O
- Execute VI on Development Computer with Real I/O
 - Not supported by all FPGA targets



ni.com/training

Testing with the Development Computer

- Compiling to run on the FPGA can require a few minutes to several hours
- Verify logic before compiling by executing the VI on the development computer (Windows PC)
- Traditional debugging tools are available



ni.com/training

Exercise 3-1: VI Execution on the Development Computer

Create an FPGA VI and verify the functionality of the VI using the development computer.

GOAL

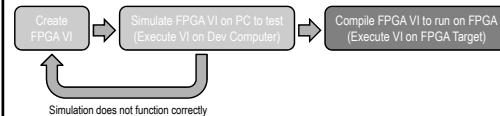
Exercise 3-1: VI Execution on the Development Computer

- Note how quickly you were able to progress from development to testing.
- Have you ever tried troubleshooting a VI without using the debugging tools?

DISCUSSION

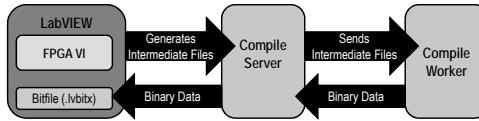
E. Compiling the FPGA VI

- After verifying logic by executing the VI on the development computer, you are ready to compile the FPGA VI and run it on the FPGA target



ni.com/training

What Happens When You Compile an FPGA VI?



- LabVIEW converts FPGA VI into intermediate files
- LabVIEW creates bitfile containing the binary data describing how to configure FPGA circuit
- FPGA VI has reference to corresponding bitfile
- Receives compilation requests from LabVIEW
- Sends compilation jobs to available compile worker
- Sends binary data to LabVIEW
- Transforms intermediate files into binary data that describes how to configure the FPGA circuit of the FPGA target
- Uses Xilinx tools to compile

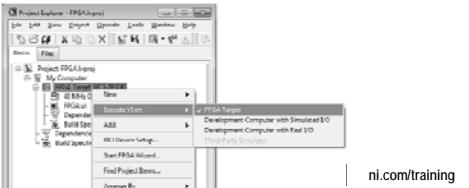


ni.com/training

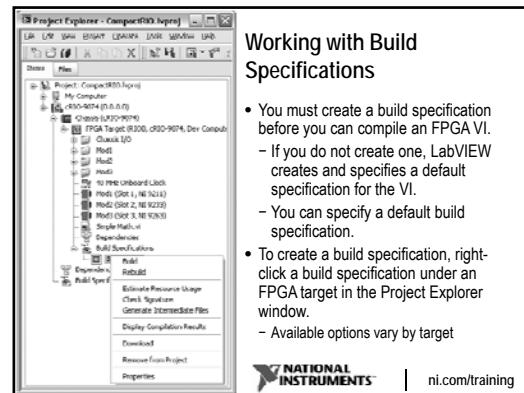
Running an FPGA VI on the FPGA Target

Configure your FPGA target to Execute VI on FPGA Target and click Run on the FPGA VI

- LabVIEW automatically compiles, downloads, and runs an FPGA VI on the FPGA target



ni.com/training



ni.com/training

Working with Build Specifications

- You must create a build specification before you can compile an FPGA VI.
 - If you do not create one, LabVIEW creates and specifies a default specification for the VI.
 - You can specify a default build specification.
- To create a build specification, right-click a build specification under an FPGA target in the Project Explorer window.
 - Available options vary by target



ni.com/training

Common Build Specification Actions

- Build—This command compiles the VI.
- Estimate Resource Usage—Estimates FPGA resource usages without compiling.
- Generate Intermediate Files—Generates intermediate files without compiling the VI. Useful for catching certain code generation errors.
- Display Compilation Results—Displays the Compilation Status window. You must build the build specification once to display the Compilation Status window.



ni.com/training

Stages of the Compilation Process

1. Generation of Intermediate Files (HDL code)
2. Queuing
3. HDL Compilation, Analysis and Synthesis
4. Mapping
5. Placing and Routing
6. Generating Bitstream
7. Bitfile Creation



ni.com/training

Generate Intermediate Files (HDL Code)

- Click Run
- Generating Intermediate Files window launches
 - Converts graphical code to VHDL
 - Generates intermediate files
- Once compilation starts, do not edit the VI
- Create a compilation queue by starting another compilation while the first is still running



ni.com/training

Exercise 3-2, Part A: VI Execution on the FPGA Target

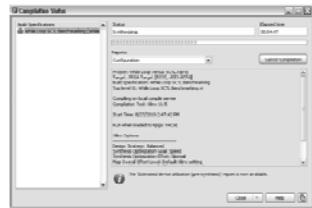
Begin Exercise 3-2 to start compilation of a working VI to run on an FPGA target.

Do only Part A.

GOAL

Compilation Status Window

- Reports
 - Select the report that is displayed
- Close Option
 - Close window
 - Disconnect All
 - Cancel All Compilations
- Help

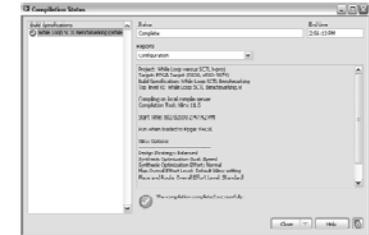


NATIONAL INSTRUMENTS

ni.com/training

Compilation Status Window – Configuration Report

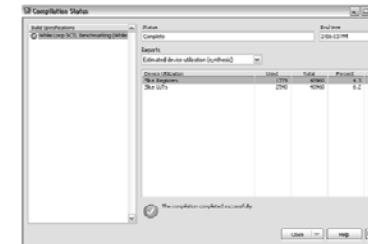
- Project information
- Xilinx compiler configuration for the FPGA VI



ni.com/training

Compilation Status Window – Device Utilization Report

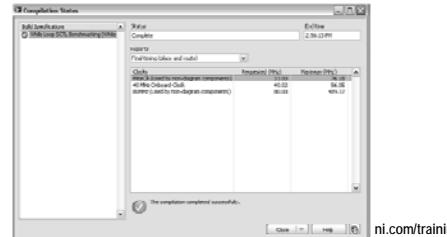
Separate reports generated at pre-synthesis and synthesis stages of compilation



ni.com/training

Compilation Status Window – Timing Report

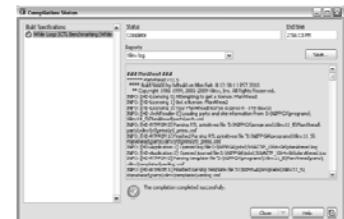
Separate reports generated at mapping (estimated values) and placing and routing (actual values) stages of compilation



ni.com/training

Compilation Status Window – Xilinx Log Report

- Includes XilinxLog.txt and .twr files
- Available after compilation is complete
- Can be saved to a file



ni.com/training

Compile Worker Window

View information about the compile worker:

- Server connection status
- Current compilation jobs



Windows Taskbar



ni.com/training

Exercise 3-2, Part B: VI Execution on the FPGA Target

Complete Exercise 3-2 to view reports generated after compiling a working VI to run on an FPGA target.

GOAL**Exercise 3-2, Part B: VI Execution on the FPGA Target**

- What is the benefit of having reports generated at different points of the compilation?
- What if you had not tested on the development computer and the VI did not function as intended?
- What if your compilation fails?

DISCUSSION**FPGA Compilation Considerations**

- What causes a recompile?
- Compiling an FPGA VI remotely
- Causes of compilation failure
- Xilinx compiler options



| ni.com/training

What Causes a Recompile?

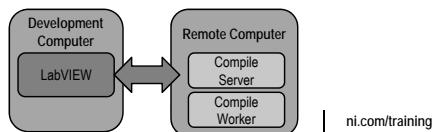
- When you run a FPGA VI that LabVIEW has already compiled, LabVIEW does not need to recompile if the VI does not have any non-cosmetic changes
- If the FPGA VI has any non-cosmetic changes, then LabVIEW will recompile your VI on the next run.
Example:
 - Change algorithm on block diagram
 - Add or delete functions on block diagram
 - Renamed control or indicator
 - Added control or indicator



| ni.com/training

Compiling an FPGA VI Remotely

- To free resources on the local computer, install the LabVIEW FPGA Compile Server and Compile Worker on a remote computer and compile the FPGA VI remotely
- Refer to *Compiling an FPGA VI Remotely (FPGA Module)* in the *LabVIEW Help* for more information about compiling FPGA code remotely



| ni.com/training

Causes of Compilation Failure

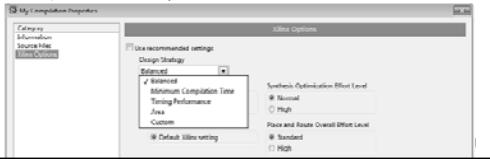
- Timing
 - Delays in the designed circuit exceed the period of the specified clock
- Resource utilization
 - The design requires more FPGA resources than are available on the FPGA



| ni.com/training

Xilinx Compiler Options

- In your Build Specification, you can customize Xilinx compiler options
- Use recommended settings is the default and recommended Xilinx compiler option
- In general, only choose a specific Design Strategy for Xilinx compiler if necessary



F. Basic Optimizations

These types of optimizations are relatively easy to implement

- Require no major changes in code architecture
- Should be basic programming practice for all FPGA VIs
- Basic optimizations in this section primarily affect FPGA size



ni.com/training

Types of Basic Optimizations

- Limit front panel objects on top-level VI
- Use small data types
- Avoid large functions



ni.com/training

Limit Front Panel Objects on Top-Level VI

- Each front panel object on the top-level VI must have logic to interact with the host VI
- Each read and write from the host to the FPGA is broken down into 32-bit packets to transfer across the bus
- Arrays/Clusters with more than 32 bits require extra copy on the FPGA to guarantee all the data is read or written coherently



| ni.com/training

Limit Front Panel Arrays

- Avoid using arrays on the front panel
 - All arrays must be of fixed size
 - Compile fails if array size is larger than available FPGA resources
 - Can quickly use large amounts of FPGA resources
 - Each bit in the array uses its own flip-flop on the FPGA
- This is a recommended optimization



| ni.com/training

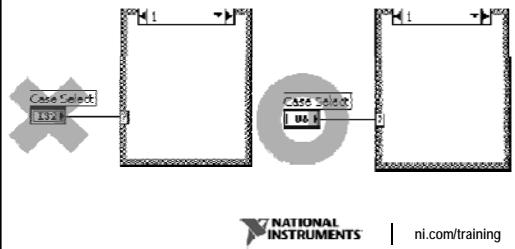
Bitpack Boolean Logic

- Each control has overhead in addition to the size of the data type.
- Maintain the same information using fewer controls
 - Display binary data as an integer
 - Use a Boolean array or cluster control/indicator



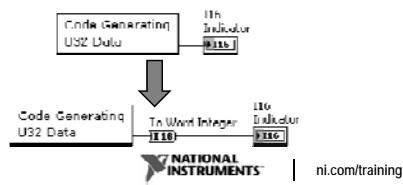
| ni.com/training

Use Small Data Types



Use Small Data Types - Coercion

- Eliminate coercion dots
 - Determine necessary input format
 - Insert conversion function
 - Intentional coercion results in more efficient implementation



Avoid Large Functions

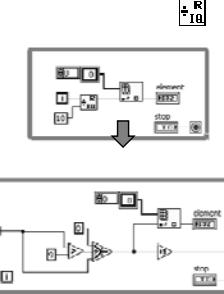
Not all functions are equal



Avoid Large Functions – Quotient & Remainder

This function consumes significant space on the FPGA

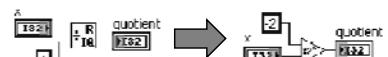
- Quotient & Remainder often used to increment based on iteration number
- Consider replacing with actual increment function and shift register



ni.com/training

Avoid Large Functions – Scale By Power of 2

- Uses significant FPGA space if input for power is a control
- However, if you wire a constant to the input, the function consumes no space on the FPGA
- Use negative powers to replace Quotient & Remainder function whenever possible



ni.com/training

Summary—Quiz

1. You developed a VI and set the project to execute the VI on the FPGA Target. You compile the code and run the VI. Which of the following statements is true?
 - a. The block diagram and the front panel both execute on the FPGA
 - b. The block diagram executes on the FPGA and the front panel executes on the host computer.
 - c. The block diagram executes on the host computer and the front panel executes on the FPGA.
 - d. The block diagram and front panel both execute on the host computer



ni.com/training

Summary—Quiz Answer

1. You developed a VI and set the project to execute the VI on the FPGA Target. You compile the code and run the VI. Which of the following statements is true?
 - a. The block diagram and the front panel both execute on the FPGA
 - b. The block diagram executes on the FPGA and the front panel executes on the host computer.
 - c. The block diagram executes on the host computer and the front panel executes on the FPGA.
 - d. The block diagram and front panel both execute on the host computer



| ni.com/training

Summary—Quiz

2. Where should you first test your FPGA VI's functionality?
 - a. FPGA target
 - b. Development computer



| ni.com/training

Summary—Quiz Answer

2. Where should you first test your FPGA VI's functionality?
 - a. FPGA target
 - b. Development computer



| ni.com/training

Summary—Quiz

3. A bitfile (.lvbitx) is generated when an FPGA VI compiles successfully.
- a) True
 - b) False



| ni.com/training

Summary—Quiz Answer

3. A bitfile (.lvbitx) is generated when an FPGA VI compiles successfully.
- a) True
 - b) False

The bitfile contains binary data that describes how to configure the FPGA circuit of the FPGA target.



| ni.com/training

Summary—Quiz

4. Which of the following are best practices when developing FPGA VIs?
- a. Avoid large functions such as Quotient & Remainder if possible
 - b. Choose the smallest data types required for your application
 - c. Use as many front panel objects and array indicators as possible on top-level FPGA VIs



| ni.com/training

Summary—Quiz Answers

4. Which of the following are best practices when developing FPGA VIs?
 - a. Avoid large functions such as Quotient & Remainder if possible
 - b. Choose the smallest data types required for your application
 - c. Use as many front panel objects and array indicators as possible on top-level FPGA VIs



| ni.com/training

Lesson 4 Using FPGA I/O

TOPICS

- A. Introduction
- B. Configuring FPGA I/O
- C. I/O Types
- D. Using Integer Math
- E. Using CompactRIO I/O
- F. Handling FPGA I/O Errors



ni.com/training

A. Introduction



- FPGA I/O items connect I/O to the FPGA logic
- Each FPGA I/O item has a type like analog or digital
- FPGA VIs can have multiple types of I/O items
- In the NI Example Finder, navigate to:
Toolkits and Modules>FPGA>CompactRIO>Basic IO



ni.com/training

FPGA I/O Terminology



Terminal – a hardware connection, such as on a CompactRIO module



I/O Resource – a logical representation in LabVIEW FPGA of a hardware terminal



I/O Name – a name assigned in the LabVIEW project to a particular I/O Resource



ni.com/training

B. Configuring FPGA I/O

Adding FPGA I/O to the LabVIEW project

- If using CompactRIO, detect modules when you add your chassis.
 - All FPGA I/O added automatically
- If using R Series, manually add FPGA I/O
 - Select which FPGA I/O channels you want to add



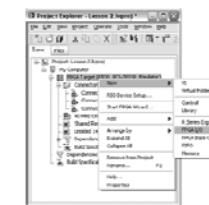
ni.com/training

Adding FPGA I/O to a CompactRIO Project



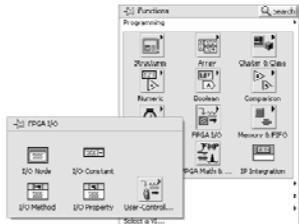
ni.com/training

Adding FPGA I/O to an R Series Project



ni.com/training

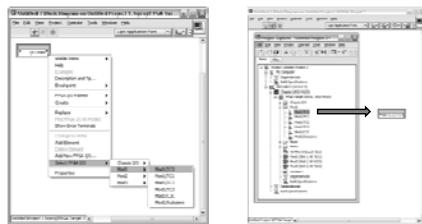
FPGA I/O Palettes



NATIONAL INSTRUMENTS

ni.com/training

Placing I/O Nodes



NATIONAL INSTRUMENTS

ni.com/training

C. I/O Types

- Digital Line – Writes and/or reads Boolean value to/from digital line
- Digital Port – Writes and/or reads unsigned integer value to/from digital port (grouping of digital lines)
- Analog I/O – Writes or reads data to/from an analog channel
 - R Series – Integer values
 - CompactRIO – Fixed-point values (can revert to integer)
- Other
 - Motion
 - CAN



NATIONAL INSTRUMENTS

ni.com/training

Digital I/O



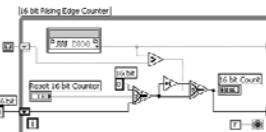
- Individual Lines – Boolean data type
- Ports (collection of lines) – Integer data type (1 bit per line)
- Depending on your hardware, digital I/O can be unidirectional or bidirectional
 - If the digital I/O line or port is bidirectional and you want to change direction of the line or the port from the FPGA VI, use the I/O Method Node, and select Set Output Enable method.



ni.com/training

Creating Counters from Digital I/O

- FPGA does not have built-in counter hardware
- Can be programmed into the FPGA
- Minimum input pulse width detectable depends on loop period and hardware
- On some hardware, you can get improved performance by replacing the While Loop with a Timed Loop



ni.com/training

Analog I/O

- Different devices have different default data types for analog I/O
 - Signed integer (R Series Devices)
 - 16-bit or 32-bit depending on the device
 - Data is raw (calibrated and unscaled)
 - Fixed-point data (CompactRIO modules)
 - Data is calibrated
- Floating-point operations and analysis can be performed on the FPGA or host



ni.com/training

D. Using Integer Math

- Analog Input
 - Converts binary representation of the voltage to a signed integer
 - Can pass this raw, unscaled data to the host VI for conversion to Volts or other units
- Analog Output
 - Convert value in Volts to binary representation in the host VI
 - Write the binary representation of the voltage to the D/A converter



| ni.com/training

Numeric Palette

Most basic numeric functions can be used on the FPGA target



| ni.com/training

Integer Division

Divide function cannot return an integer value, so you must use other functions for integer math.

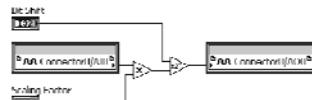
- Scale by Power of 2
- Quotient & Remainder



| ni.com/training

R Series Scaling in LabVIEW FPGA

For variable scaling, you can determine and set the scaling factor and n from the host application.



To multiply the data from A10 by 0.7 we could use:

Scaling Factor: 11500

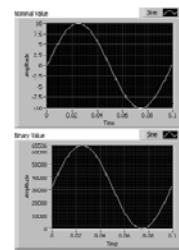
Bit Shift: -14 bits

Resulting Multiplication: $11500 / 2^{14} = 0.7019$



ni.com/training

Converting Binary Representations



R Series FPGA I/O Node:

- Returns a binary representation of the voltage input
- Value is signed integer
- Convert value to a physical quantity with engineering units
- Conversion can occur in the FPGA VI or host VI



ni.com/training

Converting Binary Representations (cont.)

- Conversion is hardware dependent
- Hardware-specific factors used in conversion
 - Voltage Range
 - ADC Resolution
 - Offset
 - Cold-junction compensation (CJC) value
 - Autozero Value



ni.com/training

Exercise 4-1: R Series I/O

Use I/O in LabVIEW FPGA to acquire analog and digital data from a simulated R Series device.

GOAL

Exercise 4-1: R Series I/O

- How many I/O nodes did you have to place on the block diagram?
- What is the benefit of renaming your I/O resources?
- Why would you return data from the digital port as a U8?

DISCUSSION

E. Using CompactRIO I/O

- Analog Input and Analog Output modules use fixed-point data type
- Fixed-point data type
 - Provides some of the flexibility of floating-point data type, but also maintains the size and speed advantages of integer arithmetic
 - Covered in *Lesson 7, Signal Processing*



ni.com/training

CompactRIO I/O

Different modules can have different

- Fixed-point configurations
- FPGA I/O Methods
- FPGA I/O Properties

Refer to the *NI-RIO»C Series Reference and Procedures* section in the *LabVIEW Help* for module-specific information.



ni.com/training

Module Data Types

- Different modules return fixed-point data with different configurations
 - Based on accuracy and range of module values
 - Some I/O nodes return scaled and calibrated fixed-point values



ni.com/training

Module Properties and Methods

Different modules have different properties and methods.
These are two examples:

Properties



Methods



ni.com/training

Properties



Methods



ni.com/training

Exercise 4-2: CompactRIO I/O

Use I/O in LabVIEW FPGA to acquire analog thermocouple data from two channels of an NI 9211 module. Find the difference in the values returned by these two channels.

GOAL

Exercise 4-2: CompactRIO I/O

- If you wanted to convert the thermocouple data into Fahrenheit or Celsius, where can that conversion take place?

DISCUSSION

F. Handling FPGA I/O Errors

- Right-click the I/O Node to show error terminals
- Error information is useful for debugging and validating data, and is essential in some cases, but it uses space on the FPGA and can slow execution



ni.com/training

Error Cluster Optimization

- When FPGA resources are limited
- Use sequence structures for data flow instead
 - Show error terminals only for modules whose functions are critical to system operation
 - Show terminals only once per module if multiple calls are made to the module
 - Do not pass error clusters through program or display the error cluster on the front panel
 - Unbundle the source and/or code items and handle the errors immediately



ni.com/training

Error Clusters for Reliability

- Examples of FPGA I/O errors that need to be handled
- Any safety-critical operations
 - Module-specific error codes for different CompactRIO modules
 - Examples
 - File not found (NI 9802)
 - Timeout (CAN modules)
 - Module has been removed or is not secure



ni.com/training

Summary—Quiz

1. Match each FPGA I/O term to the definition that best fits it.

Terminal	A name assigned by the developer to a particular I/O resource.
I/O Resource	A hardware connection, such as on a CompactRIO module.
I/O Name	A logical representation in LabVIEW FPGA of a terminal.



ni.com/training

Summary—Quiz Answers

1. Match each FPGA I/O term to the definition that best fits it.

Terminal	A name assigned by the developer to a particular I/O resource.
I/O Resource	A hardware connection, such as on a CompactRIO module.
I/O Name	A logical representation in LabVIEW FPGA of a terminal.



ni.com/training

Summary—Quiz

2. Match each analog I/O device to its default data type.

NI PCIe-7852R R Series board	Integer
NI 9234 CompactRIO module	Fixed-point



ni.com/training

Summary—Quiz Answer

2. Match each analog I/O device to its default data type.

NI PCIe-7852R R Series board	→ Integer
NI 9234 CompactRIO module	→ Fixed-point



ni.com/training

Lesson 5
Timing an FPGA VI

TOPICS

- A. Timing Express VIs
- B. Implementing Loop Execution Rates
- C. Creating Delays Between Events
- D. Measuring Time Between Events
- E. Benchmarking Loop Periods

NATIONAL INSTRUMENTS | ni.com/training

A. Timing Express VIs

Loop Timer Express VI

- Implementing Loop Rates
- Wait Express VI

 - Creating Delays Between Events

- Tick Count Express VI

 - Benchmarking

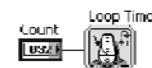


| ni.com/training

B. Implementing Loop Execution Rates

Use the Loop Timer Express VI

- Waits the value you specify in Count between loop iterations
- Controls the period of a loop
- Place the Loop Timer Express VI inside a loop
 - Perform I/O operations at a specific frequency
 - Control loop execution rate



| ni.com/training

Configure Loop Timer

- Counter Units
 - Ticks — clock cycles
 - μ Sec — microseconds
 - mSec — milliseconds
- Size of Internal Counter
 - 32 Bit
 - 16 Bit
 - 8 Bit



Size of Internal Counter determines the maximum time a timer can track.

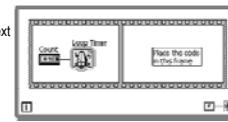
To save space on the FPGA, use the smallest Size of Internal Counter possible.



| ni.com/training

Loop Timer Express VI Functionality

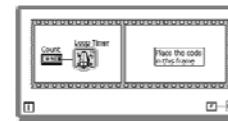
- First iteration
 - Records current time as initial time
 - Immediately executes the code in the next frame without any delay
- Second iteration
 - Adds Count to initial time and waits until Count has elapsed from the initial recorded time
 - Executes the code in the next frame after Count has elapsed
- Subsequent iterations
 - Loop Timer continues to increment the time record it initiated upon the first call
 - Does not reset each time



| ni.com/training

Loop Timer Express VI Caveat

- If an execution instance is missed, such as when the logic in the loop takes longer to execute than the specified Count:
 - Loop Timer returns immediately and establishes a new reference time stamp for subsequent calls
 - No delay



| ni.com/training

Implementing Loop Execution Rates

Watch out for rollover

- Size of Internal Counter determines the maximum time the timer can track

Size of Internal Counter	Ticks	μ s	ms
32-bit	1 to 4,294,967,296 ticks	1 μ s to 71 minutes	1 ms to 49 days
16-bit	1 to 65,536 ticks	1 μ s to 65 ms	1 ms to 65 seconds
8-bit	1 to 256 ticks	1 μ s to 256 μ s	1 ms to 256 ms



ni.com/training

Tick Period

- Tick period is based on the frequency of the FPGA clock
- If using a 40 MHz FPGA clock, the tick period is 25 ns
- Must use ticks for nanosecond timing

Size of Internal Counter	Ticks	μ s	ms
32-bit	1 to 4,294,967,296 ticks	1 μ s to 71 minutes	1 ms to 49 days
16-bit	1 to 65,536 ticks	1 μ s to 65 ms	1 ms to 65 seconds
8-bit	1 to 256 ticks	1 μ s to 256 μ s	1 ms to 256 ms



ni.com/training

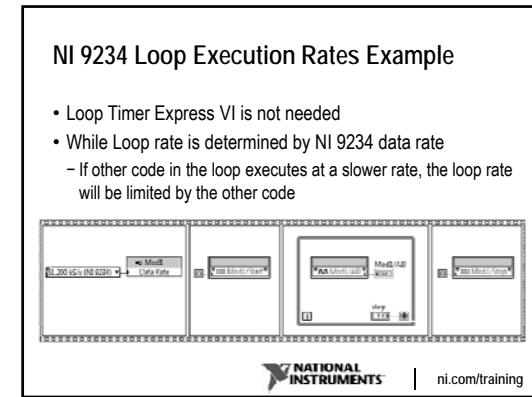
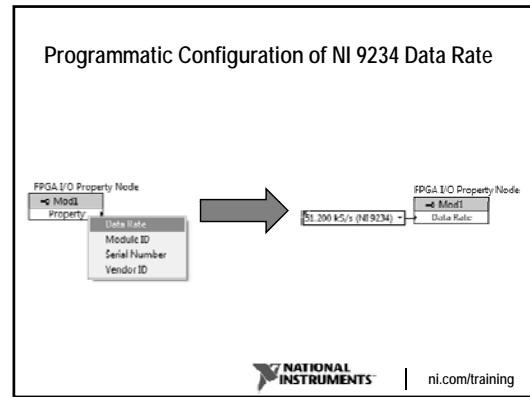
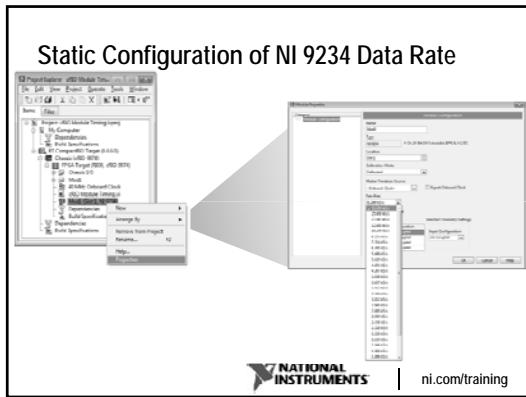
cRIO Module Loop Execution Rates

Some cRIO modules have configurable timing

- Do not need to use Loop Timer Express VI to time loop rate
- Example:
 - NI 9234
 - You can configure the data rate at which the NI 9234 module acquires and returns data
 - Static/Programmatic Configuration
 - Module Properties dialog box
 - FPGA I/O Property Node

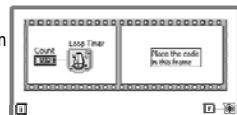


ni.com/training



Loop Condition Terminal

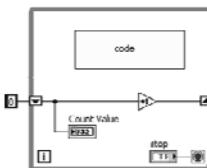
- False constant is acceptable
 - FPGA logic is often meant to run indefinitely on the FPGA
- Controls and application logic are still acceptable



ni.com/training

While Loop Iteration Terminal Maximum

- The iteration terminal count will keep outputting 2,147,483,647 once it reaches this value
- If you need a counter to rollover when it reaches its maximum value, you should implement your own counter



ni.com/training

Exercise 5-1: While Loop Timing

Use the Loop Timer Express VI to time a While Loop running on an FPGA VI.

GOAL

Exercise 5-1: While Loop Timing

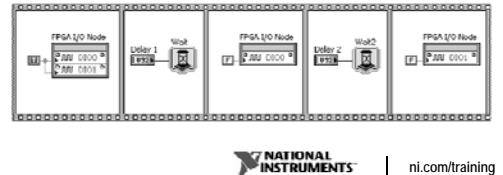
- If the code in the second frame of the Sequence structure takes longer to execute than the value of AI Sample Time control, how long would each iteration of the While loop take?

DISCUSSION

C. Creating Delays Between Events

Use Wait Express VI to create a delay between events

- Examples
 - Create delay between a trigger and subsequent output
 - Create a series of delays



NATIONAL INSTRUMENTS

ni.com/training

Wait Express VI Configuration

- Counter Units
- Size of Internal Counter
 - Determines maximum possible wait time



NATIONAL INSTRUMENTS

ni.com/training

Comparing Loop Timer and Wait VIs

- Code structure is the same
- Loop Timer does not wait the first time it is called
- Wait Express VI waits every iteration of the While Loop



NATIONAL INSTRUMENTS

| ni.com/training

D. Measuring Time Between Events

Tick Count Express VI

- Returns the value of a free-running FPGA counter in the units specified

Use Tick Count Express VI to measure time between events

• Examples

- Measure the time between edges on a digital signal
- Determine the execution time of a section of code



NATIONAL INSTRUMENTS

| ni.com/training

Measuring Time Between Events

- Calculate the difference between the results of two Tick Count Express VIs to determine the execution time
- Subtract one from the result of the calculation to compensate for the execution time of the Tick Count Express VI



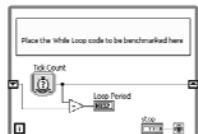
NATIONAL INSTRUMENTS

| ni.com/training

E. Benchmarking Loop Periods

Use Tick Count Express VI to measure loop periods

- Calculate the time difference
- Remove the benchmarking code later
- Benchmarking code executes in parallel
 - Does not add additional time to the loop period



Tick Count Rollover

Watch out for rollover

- If the code you are benchmarking takes more than the maximum value Tick Count can handle, the elapsed time will be much lower than expected due to rollover

Size of Internal Counter	Ticks	μs	ms
32-bit	1 to 4,294,967,296 ticks	1 μs to 71 minutes	1 ms to 49 days
16-bit	1 to 65,536 ticks	1 μs to 65 ms	1 ms to 65 seconds
8-bit	1 to 256 ticks	1 μs to 256 μs	1 ms to 256 ms



ni.com/training

Exercise 5-2: While Loop Benchmarking

Benchmark the loop period of a While Loop containing code.

GOAL

Exercise 5-2: While Loop Benchmarking

- If you configured the Tick Count Express VI to use Counter Units of us or ms instead of Ticks, would the benchmarking results in this exercise still be reasonable?
- If the code you want to benchmark executes within nanoseconds, what Counter Unit should you use?

DISCUSSION

Summary—Matching Quiz Answers

- | | |
|--------------------------|--|
| 1. Loop Timer Express VI | A. Use to create delays between events |
| 2. Wait Express VI | B. Use to benchmark execution speeds |
| 3. Tick Count Express VI | C. Use to control loop execution rate |

NATIONAL INSTRUMENTS | ni.com/training

Summary—Matching Quiz

- | | |
|--------------------------|--|
| 1. Loop Timer Express VI | A. Use to create delays between events |
| 2. Wait Express VI | B. Use to benchmark execution speeds |
| 3. Tick Count Express VI | C. Use to control loop execution rate |

NATIONAL INSTRUMENTS | ni.com/training

Summary—Quiz

2. What is a potential drawback of using an 8-bit counter instead of a 32-bit counter?
 - a. Decreased FPGA resources used
 - b. 32-bit clocks are slower
 - c. Maximum time the timer can track is not large enough for the application



| ni.com/training

Summary—Quiz Answer

2. What is a potential drawback of using an 8-bit counter instead of a 32-bit counter?
 - a. Decreased FPGA resources used
 - b. 32-bit clocks are slower
 - c. Maximum time the timer can track is not large enough for the application



| ni.com/training

Summary—Quiz

3. True or False?

If you place a Loop Timer Express VI inside a While Loop, the Loop Timer Express VI will wait during every iteration of the While Loop.



| ni.com/training

Summary—Quiz Answer**3. True or False?**

If you place a Loop Timer Express VI inside a While Loop, the Loop Timer Express VI will wait during every iteration of the While Loop.

False



ni.com/training

Summary—Quiz**4. True or False?**

Adding benchmarking code to a While Loop will not affect the execution speed of the loop.



ni.com/training

Summary—Quiz Answer**4. True or False?**

Adding benchmarking code to a While Loop will not affect the execution speed of the loop.

True



ni.com/training

Lesson 6 Executing Code in Single-Cycle Timed Loops

TOPICS

- A. Dataflow in FPGA
- B. Single-Cycle Timed Loop
- C. Using FPGA Clocks
- D. Troubleshooting SCTL Errors
- E. Optimizing Code within a While Loop
- F. Pipelining

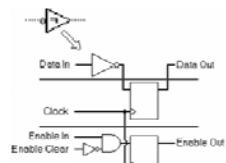


ni.com/training

A. Dataflow in FPGA

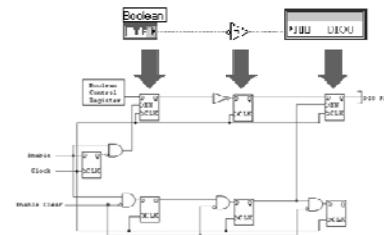
Three components necessary to maintain data flow:

- The corresponding logic function
- Synchronization
- The enable chain



ni.com/training

Dataflow in FPGA



ni.com/training

Dataflow in FPGA

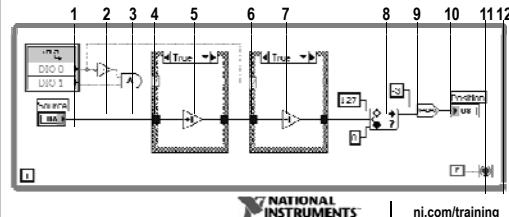
- Each function or VI takes a minimum of 1 clock cycle
- Functions can run in parallel
- Some dependent functions must run in sequence
- Application can only run as quickly as the sum of items in a sequence
- While Loops have a 2 clock cycle overhead



ni.com/training

Execution of a While Loop

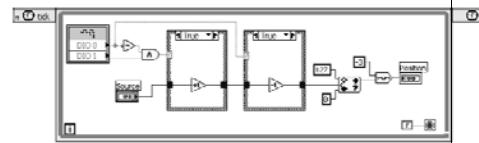
Use a single-cycle Timed Loop to convert this 12 clock-cycle While Loop...



ni.com/training

Execution of a Single-Cycle Timed Loop

...into this 1 clock-cycle Single Cycle Timed Loop



ni.com/training

B. Single-Cycle Timed Loop

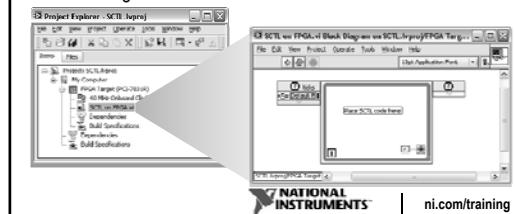
- LabVIEW automatically optimizes code inside an SCTL
 - Removes enable chain registers from code inside the SCTL
 - Executes more quickly
 - Consumes less space on the FPGA
- All code in the SCTL finishes executing within one tick of the specified FPGA clock



ni.com/training

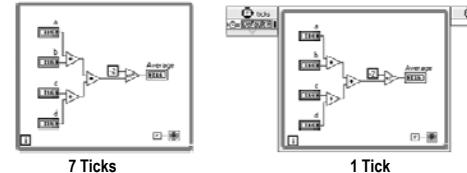
Single-Cycle Timed Loop

- Single-cycle Timed Loop
- Create by placing a Timed Loop on the block diagram of an FPGA target VI



Single-Cycle Timed Loop

Saved 6 ticks by placing this code in a SCTL



ni.com/training

Timed Loop Behavior Based On Target

- FPGA Target
 - Executes as a single-cycle Timed Loop
 - Use to optimize performance, such as resource utilization and throughput, or to simply meet the performance requirements of your application
- RT Target or Windows/My Computer Target
 - Executes as a Timed Loop
 - Implement multirate timing capabilities, precise timing, feedback on loop execution, timing characteristics that change dynamically, or several levels of execution priority
 - Use to separate deterministic from non-deterministic tasks
 - This is covered in the *LabVIEW Real-Time 1* course



ni.com/training

Exercise 6-1: While Loop versus Single-Cycle Timed Loop

Improve loop execution speeds using a single-cycle Timed Loop.

GOAL

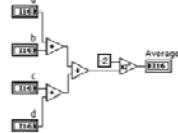
Exercise 6-1: While Loop versus Single-Cycle Timed Loop

- Why does the code require more ticks in a While Loop than in a single-cycle Timed Loop?
- What is the purpose of the Decrement function the right of each Sequence structure?

DISCUSSION

C. Using FPGA Clocks

This FPGA code always requires 5 clock ticks



- Time of clock tick = 1 / clock frequency
- The default FPGA clock frequency depends on the hardware. For example, clock frequency is 40 MHz for R Series boards and cRIO
- Altering clock frequencies alters the execution speed of FPGA code



| ni.com/training

Using FPGA Clocks

FPGA Clocks

- Determine the execution time of the individual VIs and functions on the FPGA VI block diagram

Types of FPGA Clocks

- Base Clock and Derived Clock
- Top-level Clock

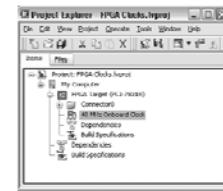


| ni.com/training

Using FPGA Clocks – Base Clock

FPGA Base Clock

- Digital signal existing in hardware that you can use as a clock for an FPGA application
- FPGA targets have an onboard clock

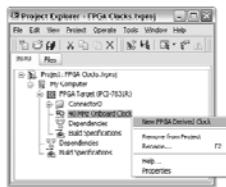


| ni.com/training

Using FPGA Clocks – Derived Clock

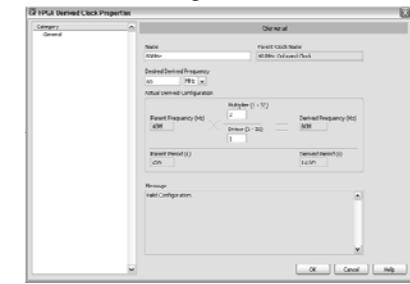
FPGA Derived Clock

- Created from a base clock
- Can derive clock frequencies other than the base clock frequency
- Can use derived clocks as a clock for an FPGA application



ni.com/training

Derived Clock Configuration

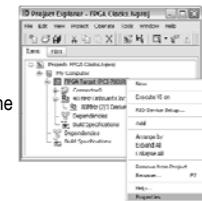


ni.com/training

Using FPGA Clocks – Top-level Clock

FPGA Top-level Clock

- Clock that the FPGA VI uses outside of SCTLs
- Controls the execution rate of the code outside of SCTLs
- Default top-level clock is the onboard FPGA clock



ni.com/training

Using FPGA Clocks – Top-level Clock

Changing the FPGA Top-level Clock Rate

- Most LabVIEW FPGA functions are designed to compile successfully at clock rates of 40 MHz
 - Not all FPGA VIs successfully compile with faster clock rates
 - Compilation Status window will report a failure if the clock rate is too fast for the FPGA VI

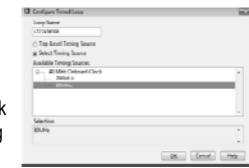


| ni.com/training

Using FPGA Clocks with SCTL

Can configure a SCTL to use a specific FPGA clock as its timing source

- Default timing source is the top-level clock
- Can set a different base clock or derived clock as the timing source



| ni.com/training

Using FPGA Clocks with SCTL

- All code in the SCTL must finish executing within one cycle of the selected timing source for the FPGA VI to compile successfully
- FPGA compilation fails with a timing violation report when the code in the SCTL takes more than the specified time

SCTL code must finish executing within one cycle of the 80 MHz derived clock



| ni.com/training

FPGA Clock Summary

- Base Clock
 - Digital signal existing in hardware that you can use as a clock for an FPGA application
- Derived Clock
 - Created from a base clock that you can use as additional clocks for an FPGA application
- Top-level Clock
 - Clock that the FPGA VI uses outside of SCTLs



ni.com/training

D. Troubleshooting SCTL Errors

- SCTL contents execute in a single clock period
- Some VIs and functions cannot be used in the loop at all
 - Analog input, analog output (Most HW)
 - Nested loops (While Loop, For Loop, Timed Loop)
 - Loop Timer and Wait Express VIs
 - Objects requiring more than one clock cycle to execute (Divide function, Quotient & Remainder function)
 - See the *LabVIEW FPGA Help* for a detailed list of unsupported objects

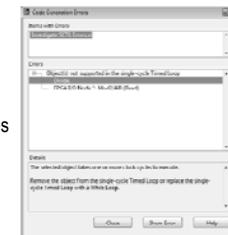


ni.com/training

SCTL Errors – Unsupported Objects

Code Generation Errors window shows unsupported objects

- Appears if errors occur while generating intermediate files
- Xilinx compilation process does not occur

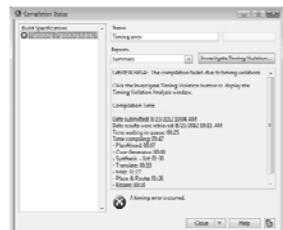


ni.com/training

SCTL Errors – Combinatorial Paths

SCTL limited by logic and routing delays in FPGA circuitry

- If total path propagation takes longer than 1 clock cycle, compilation fails
 - No way to pre-determine path length
 - Try to reduce path length before using SCTL



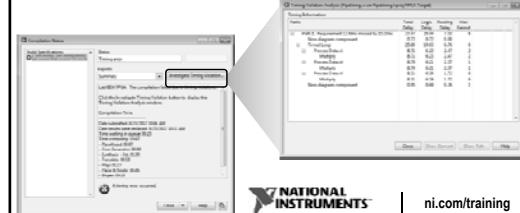
NATIONAL
INSTRUMENTS

ni.com/training

SCTL Errors – Combinatorial Paths

Investigate timing violations with Timing Violation Analysis window

- See logic, routing, and total delay of each item
 - Show path of timing violation on the block diagram



NATIONAL
INSTRUMENTS

ni.com/training

SCTL Errors – Combinatorial Paths

Timing Violation Analysis window

- Fix the timing violations
 - Speed up path by optimizing the code in the SCTL
 - Select a slower clock for the SCTL timing source
 - Speed up path by moving some code out of the SCTL
 - Speed up path by reducing its length further using pipelining

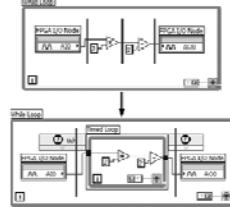


ni.com/training

E. Optimizing Code within a While Loop

Use SCTL within a While Loop to optimize sections of code

- Place section of code to optimize inside a SCTL
 - Wire a TRUE constant to the SCTL conditional terminal
- Place objects unsupported outside of the SCTL



| ni.com/training

Exercise 6-2: Fixing SCTL Errors

Examine and fix errors in a single-cycle Timed Loop caused by unsupported objects and clock rates.

GOAL

Exercise 6-2: Fixing SCTL Errors

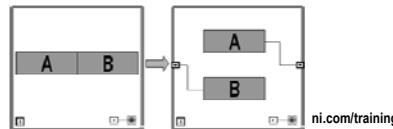
- Why is the NI 9234 Analog Input FPGA I/O Node not supported in a single-cycle Timed Loop?
- How can you estimate the fastest derived clock rate with which the SCTL will successfully compile?

DISCUSSION

F. Pipelining

Pipelining is a technique you can use to increase the throughput or speed of an FPGA loop

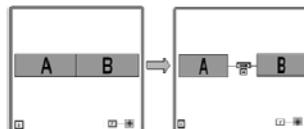
- Takes advantage of the parallel processing capabilities of the FPGA
 - By using parallel processing, increases the efficiency of sequential code
- Must divide code into discrete steps
 - Wire inputs and outputs of each step to Feedback Nodes or shift registers



Pipelining – Feedback Nodes

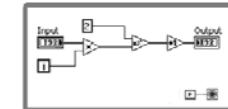
Maintain the look and feel of original application using Feedback Nodes

- Same functionality as a shift register
- Maintains more congruous VI appearance



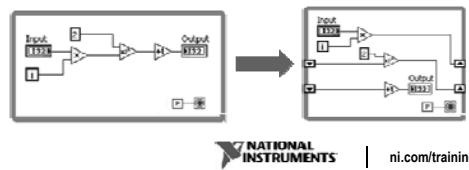
Basic Example of Pipelining

- Simple set of code with inputs and outputs
- Takes 7 clock cycles to execute
 - 5 cycles for the code, 2 for the loop
- If Input = 1 for i = 0..3
 - Output = 1, 5, 9, 13



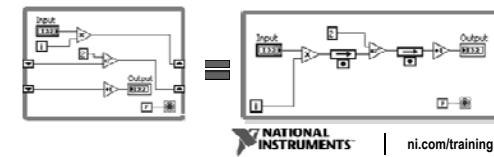
Basic Example of Pipelining

- Pass data to next step of code by using shift registers
- Takes 5 clock cycles to execute
 - 3 cycles for the code, 2 for the loop
- If Input = 1 for i = 0..3
 - Output = ?, ?, 1, 5

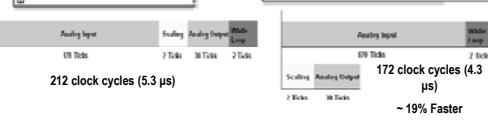
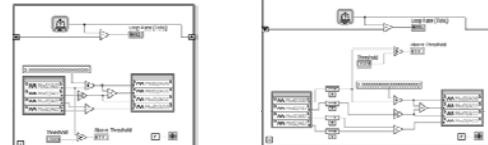


Basic Example of Pipelining

- Maintain look and feel of original application by using Feedback Nodes in sequence with code
- Same functionality as a shift register
 - Maintains more congruous VI appearance



Pipelining

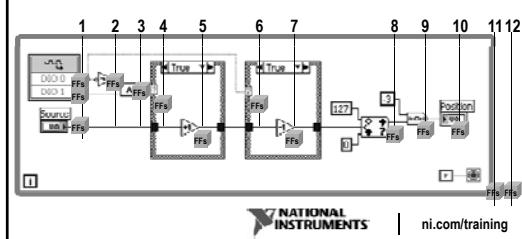


NATIONAL INSTRUMENTS | ni.com/training

Pipelining Optimization

What to do if your diagram executes too slowly?

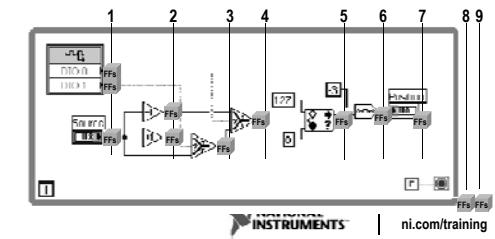
12 cycles



Pipelining Optimization

Shorten the longest path using basic optimizations

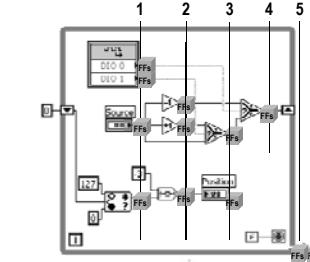
9 cycles



Pipelining Optimization

Shorten the longest path further by implementing pipelining

6 cycles



Using Pipelining in SCTLs

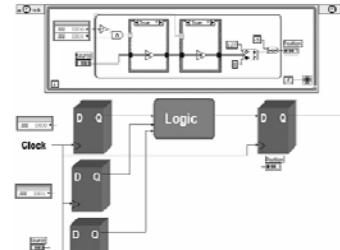
SCTL limited by propagation delays through the FPGA circuitry

- If total combinatorial path propagation takes longer than 1 clock cycle, compile fails
 - First, shorten the longest path using basic optimizations
 - If necessary, reduce the length of the combinatorial path further using pipelining



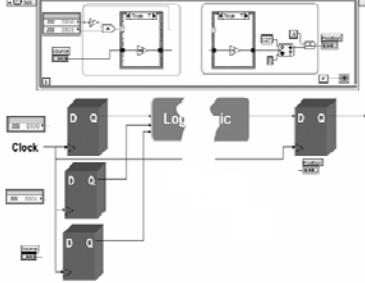
ni.com/training

Using Pipelining in SCTLs



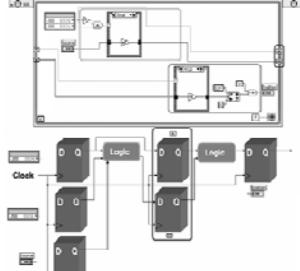
ni.com/training

Using Pipelining in SCTLs



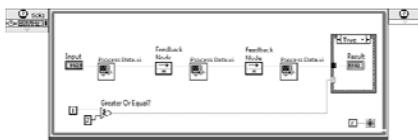
ni.com/training

Using Pipelining in SCTLs



Pipelining Caveats

- Requires code to be restructured
 - Greatly reduced by using Feedback Nodes
- Latency for a pipeline with N steps
 - Output of final step is invalid until the N th iteration
 - Output of final step lags behind the corresponding input by $N-1$ iterations.



Exercise 6-3: Pipelining

- Fix the timing violation in an SCTL by pipelining the code

GOAL

Summary—Matching Quiz

- | | |
|-------------------------|--|
| 1. FPGA base clock | a. Clock that an FPGA VI uses outside of SCLTs |
| 2. FPGA derived clock | b. Clock created from a base clock that you can use as additional clocks for an FPGA application |
| 3. FPGA top-level clock | c. Digital signal existing in hardware that you can use as a clock for an FPGA application |



ni.com/training

Summary—Matching Quiz Answers

- | | |
|-------------------------|--|
| 1. FPGA base clock | a. Clock that an FPGA VI uses outside of SCLTs |
| 2. FPGA derived clock | b. Clock created from a base clock that you can use as additional clocks for an FPGA application |
| 3. FPGA top-level clock | c. Digital signal existing in hardware that you can use as a clock for an FPGA application |



ni.com/training

Summary—Quiz

2. How does the single-cycle Timed Loop create a smaller FPGA footprint and execute within one clock tick?
 - a. By using other VIs logic functions when they are not in use
 - b. By eliminating the enable chain overhead
 - c. By passing the data to the real-time controller to process
 - d. By skipping some functions and having incomplete functionality



ni.com/training

Summary—Quiz Answer

2. How does the single-cycle Timed Loop create a smaller FPGA footprint and execute within one clock tick?
 - a. By using other VIs logic functions when they are not in use
 - b. By eliminating the enable chain overhead**
 - c. By passing the data to the real-time controller to process
 - d. By skipping some functions and having incomplete functionality



ni.com/training

Summary—Quiz

3. Which of the following objects are NOT supported in single-cycle Timed Loops?
 - a. Add
 - b. For Loop
 - c. Loop Timer VI
 - d. NI 9211 Analog Input FPGA I/O Node



ni.com/training

Summary—Quiz Answers

3. Which of the following objects are NOT supported in single-cycle Timed Loops?
 - a. Add
 - b. For Loop**
 - c. Loop Timer VI
 - d. NI 9211 Analog Input FPGA I/O Node



ni.com/training

Summary—Quiz

4. If the code in your SCTL causes a timing violation, which of the following methods can help fix it?
 - a. Optimize the code in the SCTL
 - b. Move some code out of the SCTL
 - c. Select a faster clock for the SCTL timing source
 - d. Select a slower clock for the SCTL timing source
 - e. Speed up path by reducing its length further using pipelining



| ni.com/training

Summary—Quiz Answers

4. If the code in your SCTL causes a timing violation, which of the following methods can help fix it?
 - a. **Optimize the code in the SCTL**
 - b. **Move some code out of the SCTL**
 - c. Select a faster clock for the SCTL timing source
 - d. **Select a slower clock for the SCTL timing source**
 - e. Speed up path by reducing its length further using pipelining



| ni.com/training

Lesson 7

Signal Processing

TOPICS

- A. Using Fixed-Point Data Types
- B. Using Single-Precision Floating Point
- C. Performing FPGA Math & Analysis
- D. Performing High Throughput Math
- E. Integrating Third-Party IP
- F. Additional Resources

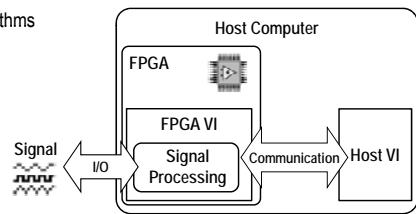


ni.com/training

Signal Processing

After acquiring a signal or before generating a signal, you might need to process it on the FPGA

- Arithmetic
- Algorithms
- Filter
- FFT
- Etc



A. Using Fixed-Point Data Types

- Provides some of the flexibility of floating-point data type, but also maintains the size and speed advantages of integer arithmetic
- Must configure the appropriate range when using fixed-point math
- Limited to a maximum size of 64 bits.



ni.com/training

Fixed-Point Data Types

- Fixed-point is a collection of data types with different ranges of values
 - Range is defined by minimum value, maximum value and delta
- Range of values and resolution are dependent upon three factors
 - Sign Encoding
 - Word Length
 - Integer Word Length



ni.com/training

Fixed-Point Terminology

- Sign Encoding – The option that determines whether the fixed-point data is signed (\pm) or unsigned (+)
- Word Length – The total number of bits used for the fixed-point data
- Integer Word Length – The number of bits used in the integer portion of the fixed-point data



ni.com/training

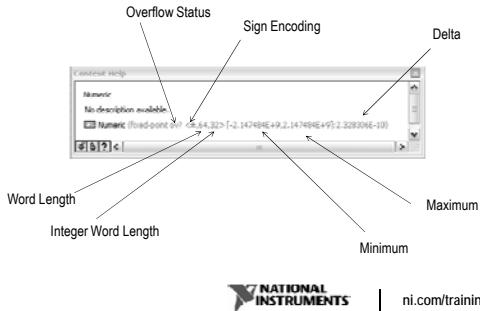
Fixed-Point Data Type

Observe how sign encoding, word length, and integer word length affect the minimum, maximum, and delta of the fixed-point data type

<Exercises>\...\Demonstrations\Fixed-Point Data Type

DEMONSTRATION

Fixed-Point Context Help



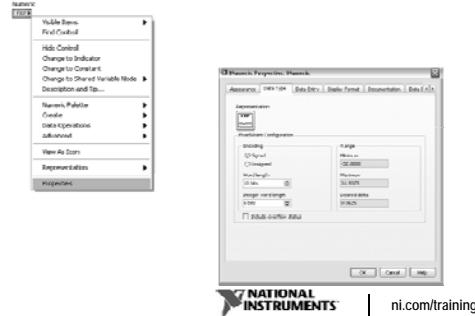
Numeric Representation Examples

Representation	delta	Min Value	Max Value
U8	1	0	255
I8	1	-128	127
FXP <+,8,7>	0.5	0	127.5
FXP <+,8,6>	0.25	0	63.75
FXP <±,8,7>	0.5	-64	63.5
FXP <±,8,6>	0.25	-32	31.75
FXP <+,8,0>	0.0039	0	0.9961
FXP <+,8,-1>	0.0020	0	0.4980



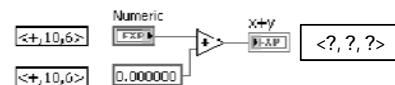
ni.com/training

Fixed-Point Configuration



Fixed-Point Arithmetic

What is the fixed-point configuration of the output?



ni.com/training

Fixed-Point Arithmetic

- Range of output values will be large enough to accommodate largest possible input values, up to 64 bits
- LabVIEW propagates the range of values along each wire. The representation is then calculated to be as small as possible without losing data.
- We will examine the following operations and provide guidelines for representing the output:
 - Addition
 - Subtraction
 - Multiplication
 - Division

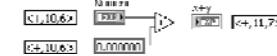


ni.com/training

Fixed-Point Arithmetic

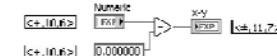
Addition

- $<\pm, A, B> + <\pm, A, B> = <\pm, A+1, B+1>$



Subtraction

- $<+, A, B> - <+, A, B> = <\pm, A+1, B+1>$

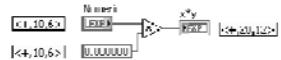


ni.com/training

Fixed-Point Arithmetic

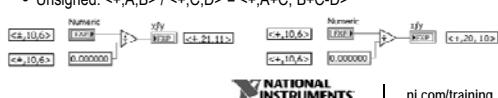
Multiplication

$$\bullet \langle +, A, B \rangle * \langle +, C, D \rangle = \langle +, A+C, B+D \rangle$$



Division

- Signed: $\langle \pm, A, B \rangle / \langle \pm, C, D \rangle = \langle \pm, A+C+1, B+C-D+1 \rangle$
- Unsigned: $\langle +, A, B \rangle / \langle +, C, D \rangle = \langle +, A+C, B+C-D \rangle$



NATIONAL INSTRUMENTS

ni.com/training

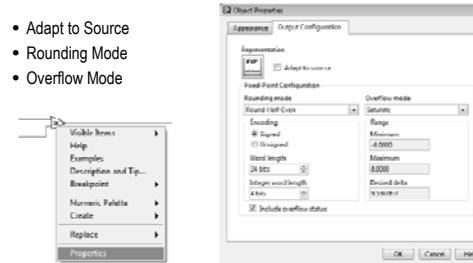
Demonstration – Simple Fixed-Point Math

Observe the impact of addition, subtraction, and multiplication on the fixed-point data type.

DEMONSTRATION

Configuring Fixed Point Functions

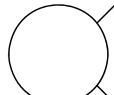
- Adapt to Source
- Rounding Mode
- Overflow Mode



NATIONAL INSTRUMENTS

ni.com/training

Rounding



Rounding – occurs when the precision of the input value or the result of an operation is greater than the precision of the output type.

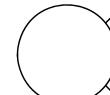
Rounding Mode

- Truncate
- Round Half-Up
- Round Half-Even (default)



ni.com/training

Overflow



Overflow – occurs when the result of an operation is outside the range of values that the output type can represent

Overflow Mode

- Wrap
- Saturate (default)



ni.com/training

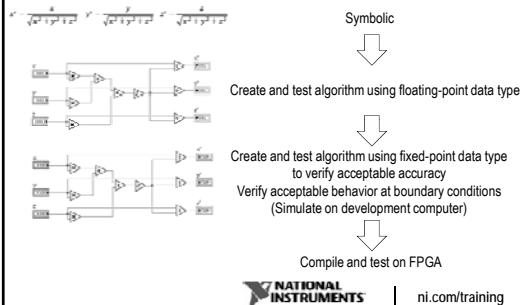
Selecting an Overflow and Rounding Mode

- Use same format for all related functions
- Mixing fixed-point configuration types can cause data loss
- Configure math functions to handle saturation, truncation, and rounding as needed



ni.com/training

Fixed-Point Algorithm Development Process



Exercise 7-1: Fixed-Point Configuration

Use the fixed-point data type and configure rounding and overflow modes if necessary

GOAL

Exercise 7-1: Fixed-Point Configuration

- When should you allow the functions to automatically grow the size of the fixed-point configuration?
- When should you manually configure the fixed-point configuration of the function output?

DISCUSSION

B. Using Single-Precision Floating-Point

- LabVIEW FPGA supports the single-precision floating-point (SGL) data type in FPGA VIs



To Single Precision Float
SGL



| ni.com/training

Using Floating Point in FPGA

- Benefits
 - Easier to rapidly prototype than fixed-point data type
- Caveats
 - Requires significantly higher FPGA resource usage than fixed-point data type for some operations
 - Most functions cannot perform floating-point operations inside SCTL



| ni.com/training

Use Cases for Using Floating Point

- Free the host processor to perform other operations and use single-precision floating-point conversions with I/O
- Operate on data from multiple inputs or outputs that return different fixed-point data types
- You need to represent very large and very small numbers in the same data path, such as with accumulators
- You need to rapidly prototype



| ni.com/training

Use Cases for Using Floating Point

- Free the host processor to perform other operations and use single-precision floating-point conversions with I/O

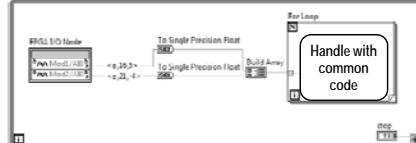


NATIONAL
INSTRUMENTS

| ni.com/training

Use Cases for Using Floating Point

- Operate on data from multiple inputs or outputs that return different fixed-point data types

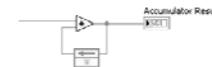


NATIONAL
INSTRUMENTS

| ni.com/training

Use Cases for Using Floating Point

- You need to represent very large and very small numbers in the same data path, such as with accumulators



NATIONAL
INSTRUMENTS

| ni.com/training

Use Cases for Using Floating Point

- You need to rapidly prototype
 - Use floating-point to get functional hardware designs quickly
 - Convert to fixed-point later as necessary to optimize FPGA performance or resource usage



NATIONAL
INSTRUMENTS

| ni.com/training

Using Floating Point in FPGA – Additional Information

Refer to the following topics in the LabVIEW Help

- Deciding Which Data Type to Use in FPGA Designs
- Using the Single-Precision Floating-Point Data Type
- Functions that Support the Single-Precision Floating-Point Data Type in FPGA VIs

NATIONAL
INSTRUMENTS

| ni.com/training

C. Performing FPGA Math & Analysis

Configurable, prebuilt, optimized, tested and documented functions to perform common tasks:

- Mathematics
- Measurements
- Control, PID
- Filtering
- Signal Processing
- High Throughput Math
- And more



NATIONAL
INSTRUMENTS

| ni.com/training

FPGA Math & Analysis VIs

The FPGA Math & Analysis palette contains three types of nodes:

1. Primitive
2. Non-configurable Express VIs
3. Configurable Express VIs

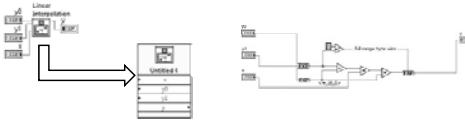


ni.com/training

FPGA Math & Analysis VIs

Non-Configurable Express VI

- No configuration window
- Output data type adapts to input data types



- Can right-click and select Convert to SubVI if necessary
- View or customize code
 - Lose express capabilities (e.g. output data type no longer adapts to input data types)



ni.com/training

FPGA Math & Analysis VIs

Configurable Express VI



ni.com/training

VI Execution Modes

Outside Single-Cycle Timed Loop Inside Single-Cycle Timed Loop

NATIONAL INSTRUMENTS | ni.com/training

VI Throughput and Latency

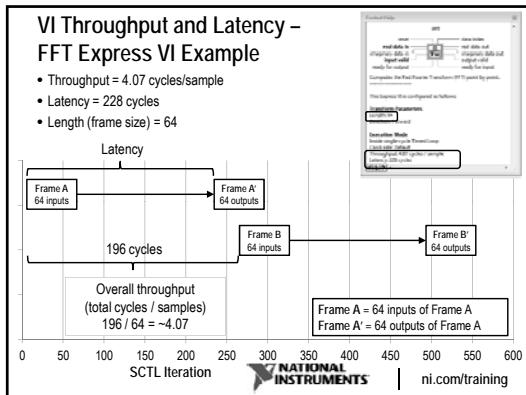
- Throughput is the # of cycles between accepting a new input
- Latency is the # of cycles before producing a valid result for the corresponding input

NATIONAL INSTRUMENTS | ni.com/training

VI Throughput and Latency – Square Root Example

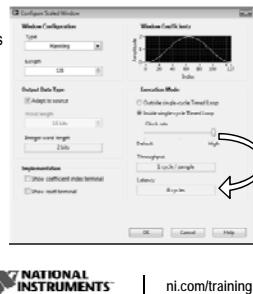
- Throughput = 2 cycles/sample
- Latency = 8 cycles

NATIONAL INSTRUMENTS | ni.com/training



VI Latency and Clock Rate

- Clock rate specifies the maximum clock rate at which this function can compile
- Faster clock rate requires more internal pipelining stages
 - Increases FPGA resource usage
 - Increases latency

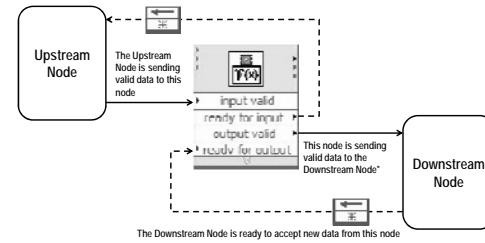


NATIONAL INSTRUMENTS

ni.com/training

VI Handshaking

This node is ready to accept new data from the Upstream Node on the next clock cycle

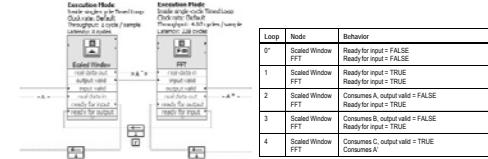


NATIONAL INSTRUMENTS

ni.com/training

VI Handshaking

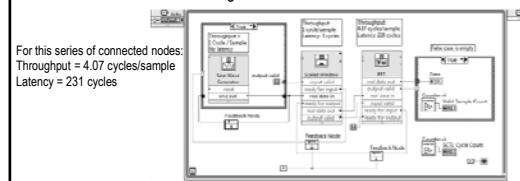
Use the handshaking terminals to let the nodes determine when data is consumed.



ni.com/training

FPGA Loop Throughput and Latency

- Throughput for a series of nodes inside SCTL = Throughput rate of the slowest node (largest)
- Latency for a series of nodes inside SCTL = $\text{Node}_1 \text{ latency} + \text{Node}_2 \text{ latency} + \dots + \text{Node}_n \text{ latency}$
- Use caution when branching wires



Exercise 7-2: Four-Wire Handshaking Protocol

Implement a fast throughput rate for the Scaled Window and FFT Express VIs by using a single-cycle Timed Loop and the four-wire handshaking protocol

GOAL

Exercise 7-2: Four-Wire Handshaking Protocol

What is the purpose of each of these terminals of the FFT Express VI in this exercise?

- ready for input
- input valid
- ready for output
- output valid

DISCUSSION

D. Performing High Throughput Math

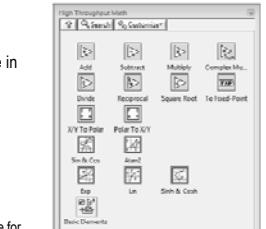
Achieve high FPGA fixed-point math and analysis rates

Perform operations unavailable in basic Numeric palette:

- Trigonometric
- Logarithmic
- Etc.

All operations are supported inside the SCTL

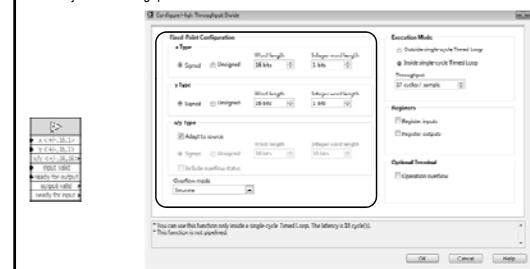
- Divide inside a SCTL!
- Allows you to use SCTL to optimize for size



ni.com/training

FPGA High Throughput Math Nodes

Configurable Node
• Fixed-point configuration options
• Adjustable throughput



Execution Mode, Throughput, and Latency

Outside Single-Cycle Timed Loop Inside Single-Cycle Timed Loop

* You can use this function only outside a single-cycle Timed Loop.
The latency is 17 cycles to return a valid result.
This function is not pipelined.

Throughput: 17 cycles/sample
* You can use this function only inside a single-cycle Timed Loop.
The latency is 4 cycles.
* This function has a 4-stage pipeline.
This function is not pipelined.

NATIONAL INSTRUMENTS | ni.com/training

Basic Elements

Perform low-level FPGA operations:

- Accumulating totals
- Delaying signals
- Binary arithmetic
- Accessing DSP48E slice

Use as building blocks in more complex algorithms

NATIONAL INSTRUMENTS | ni.com/training

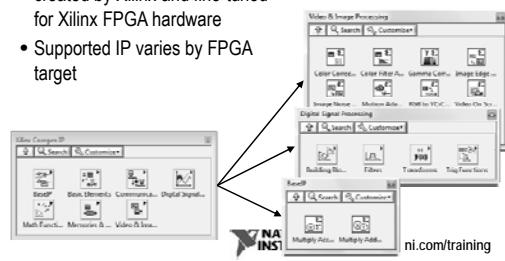
E. Integrating Third-Party Intellectual Property (IP)

- Xilinx CORE Generator IP
- Other Third-Party IP

NATIONAL INSTRUMENTS | ni.com/training

Integrating Xilinx CORE Generator IP into FPGA VIs

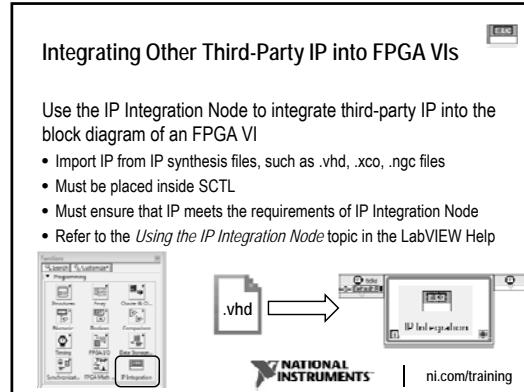
- Access a large collection of IP created by Xilinx and fine-tuned for Xilinx FPGA hardware
- Supported IP varies by FPGA target



Integrating Other Third-Party IP into FPGA VIs

Use the IP Integration Node to integrate third-party IP into the block diagram of an FPGA VI

- Import IP from IP synthesis files, such as .vhd, .xco, .ngc files
- Must be placed inside SCTL
- Must ensure that IP meets the requirements of IP Integration Node
- Refer to the *Using the IP Integration Node* topic in the LabVIEW Help



F. Additional Resources

www.ni.com/ipnet

LabVIEW FPGA IPNet

- Collection of FPGA IP and examples gathered from the LabVIEW FPGA functions palette, internal National Instruments developers, and the LabVIEW FPGA community
- Browse, learn, and download LabVIEW FPGA functions or IP
- Share your LabVIEW FPGA IP
- Save development time!

IPNet Categories		
Control	Digital Buses, Protocols	Signal Processing
Data Acquisition/Generation	Encryption	Simulation
Data Manipulation	Math	Vision
RF	Hardware Integration	

Summary—Quiz

1. Which of the following are parameters used to define the range of values that are represented by a fixed-point number?
 - a. Rounding mode
 - b. Sign Encoding
 - c. Word Length
 - d. Integer Word Length



| ni.com/training

Summary—Quiz Answer

1. Which of the following are parameters used to define the range of values that are represented by a fixed-point number?
 - a. Rounding mode
 - b. Sign Encoding
 - c. Word Length
 - d. Integer Word Length



| ni.com/training

Summary—Quiz

2. Match each term to its definition

- | | |
|------------|---|
| Throughput | a. Number of cycles before producing a valid result for the corresponding input |
| Latency | b. Minimum number of cycles between two successive values or frames of valid input data |



| ni.com/training

Summary—Quiz Answers

2. Match each term to its definition

- | | |
|------------|---|
| Throughput | a. Number of cycles before producing a valid result for the corresponding input |
| Latency | b. Minimum number of cycles between two successive values or frames of valid input data |



ni.com/training

Summary—Quiz

3. Match each handshaking protocol terminal of a node to its purpose

- | | |
|------------------|---|
| Ready for input | a. Indicates whether the node can accept a new data point during the next clock cycle |
| Input valid | b. Indicates that the current data point produced by the node is valid and downstream nodes are ready to use it |
| Ready for output | c. Specifies that the next data point has arrived for processing |
| Output valid | d. Specifies whether the downstream node can accept a new data point |



ni.com/training

Summary—Quiz Answers

3. Match each handshaking protocol terminal of a node to its purpose

- | | |
|------------------|---|
| Ready for input | a. Indicates whether the node can accept a new data point during the next clock cycle |
| Input valid | b. Indicates that the current data point produced by the node is valid and downstream nodes are ready to use it |
| Ready for output | c. Specifies that the next data point has arrived for processing |
| Output valid | d. Specifies whether the downstream node can accept a new data point |



ni.com/training

Summary—Quiz

4. Which of the following third-party FPGA IP can you reuse in your FPGA VIs?
 - a. Xilinx Core Generator IP
 - b. Third-party IP synthesis files that meet the requirements of IP Integration Node
 - c. Third-party IP from LabVIEW FPGA IPNet



| ni.com/training

Summary—Quiz Answers

4. Which of the following third-party FPGA IP can you reuse in your FPGA VIs?
 - a. Xilinx Core Generator IP
 - b. Third-party IP synthesis files that meet the requirements of IP Integration Node
 - c. Third-party IP from LabVIEW FPGA IPNet



| ni.com/training

Lesson 8

Sharing Data on FPGA

TOPICS

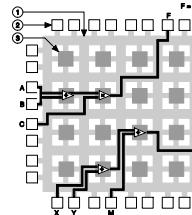
- A. Parallel Loops on FPGA
- B. Shared Resources
- C. Sharing Latest Data
- D. Transferring Buffered Data
- E. Comparing Data Sharing Methods



ni.com/training

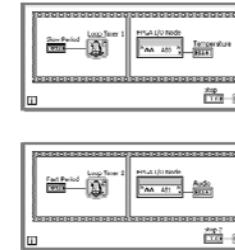
A. Parallel Loops on FPGA

- Graphical programming promotes parallel code architectures
- LabVIEW FPGA implements true parallel execution



ni.com/training

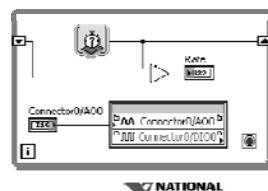
Parallel Loops with Different Sampling Rates



ni.com/training

Loop Rate Limitation – Longest Path

- AO takes ~35 ticks, DI takes 1 tick (HW Specific)
- DI limited by AO when in same loop

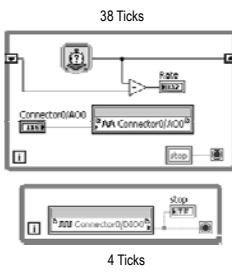


38 Ticks

NI NATIONAL INSTRUMENTS | ni.com/training

Loop Rate Limitation – Longest Path (continued)

- AO takes ~35 ticks, DI takes 1 tick (HW Specific)
- Separate functions to allow DI to run independent of AO
- This allows you to sample DI 10 times faster by using a separate loop



4 Ticks

NI NATIONAL INSTRUMENTS | ni.com/training

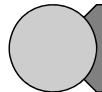
Sharing Data Between Parallel Loops

What if you want to pass data from one loop to another?

- Since loops are executing in parallel, you cannot use a wire to pass data
- Need access to a resource that can be shared by multiple processes

NI NATIONAL INSTRUMENTS | ni.com/training

B. Shared Resources



Shared Resource – Any resource that is accessed by multiple objects in an FPGA VI

- Digital outputs on most targets
- Analog inputs on most targets
- Analog output on most targets
- Digital inputs on some targets
- Non-reentrant subVIs
- Memory items
- Register items
- FIFOs
- Local and global variables



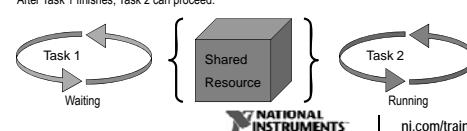
| ni.com/training

Shared Resources

Before a task can begin using a shared resource, it must wait until the resource is free.

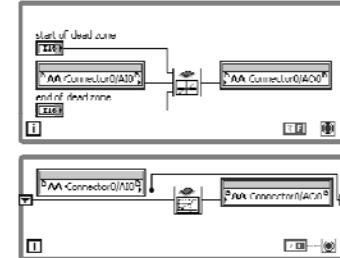


After Task 1 finishes, Task 2 can proceed.



| ni.com/training

Shared Resources Example



| ni.com/training

C. Sharing Latest Data

You can use the following to store and access latest data on the FPGA

- Local and global variables
- Memory items
- Register items



| ni.com/training

Sharing Latest Data – Variables

- Variables are block diagram elements that allow you to access or store data in the flip-flops of the FPGA
- Store only the latest data you write to it
 - Good choice if you don't need every value you acquire
 - Do not need extra code to discard unused values
- Two types of variables on FPGA
 - Local – Accessible only by a single VI
 - Global – Accessible by any VI running on the FPGA target



| ni.com/training

Local Variables

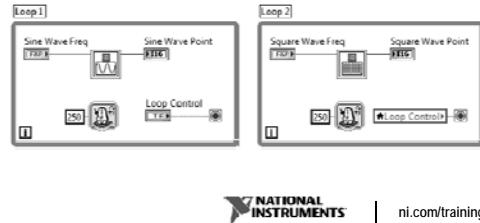
- Store data in front panel controls and indicators
- Accessible only by a single VI
- Use to maintain data separately for separate re-entrant subVIs
- Create by right-clicking on a control or indicator on the block diagram and selecting **Create»Local Variable**
- If you use this approach, you may find yourself creating controls and indicators only for data sharing within the VI they are on. Create a coding convention to help you distinguish such controls and indicators from those that meant for passing data in and out of VIs



| ni.com/training

Local Variable Example

Share data between parallel loops running in the same FPGA VI



ni.com/training

Global Variables

- Accessible across multiple FPGA VIs
- Create from Functions palette by selecting Structures>Global Variable
- If you created a control or indicator only to use a local variable, consider replacing it with a global variable instead to decrease FPGA resource usage



ni.com/training

Sharing Latest Data – Memory Items

- Similar to variables in that only the most recent data is stored
- Data can be stored in FPGA memory
 - Allocate an address or block of addresses
 - Can use all available memory on the FPGA
 - Use of block memory consumes relatively few logic resources
 - Unlike a fixed-size array
- Caveats
 - Does not include extra logic to ensure data integrity across clock domains



ni.com/training

Memory Item Usage

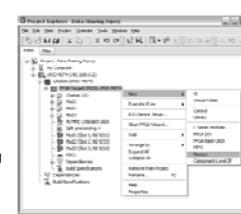
- Useful if you do not need every data value that you acquire
 - If you need every data value, FIFOs are a better option
 - No need to discard old values
- Two options for creation
 - Target-Spaced
 - VI-Defined



ni.com/training

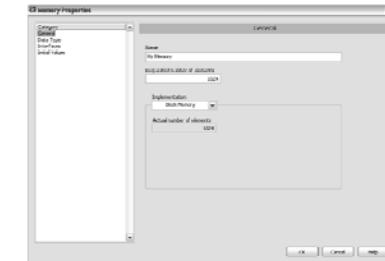
Target-Spaced Memory Item

- Accessible by any VI running on the FPGA target
- Use to store data that is accessed by multiple VIs
- Creation
 - Right-click on FPGA Target
 - Select New>Memory
 - Launches Memory Properties Dialog Box



ni.com/training

Memory Properties Dialog Box - General Page



ni.com/training

Memory Items – Implementation

Use block memory when:

- You need larger amounts of memory
- You do not have enough logic resources available to use look-up tables

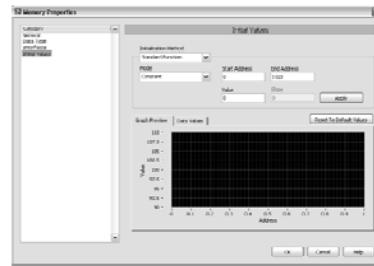
Use look-up tables when:

- Accessing memory in a single-cycle timed loop and need to read data from memory in the same cycle as the one in which you give the address
- The amount of memory needed is smaller than the minimum amount of embedded block memory on the FPGA
- You do not have enough free embedded block memory on the FPGA



ni.com/training

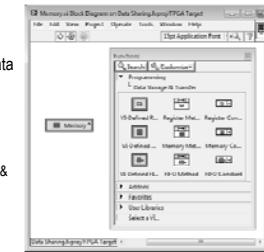
Memory Properties Dialog Box – Initial Values Page



ni.com/training

VI-Defined Memory Item

- Use for data that you only access from a single VI
- Use if maintaining separate data for separate instances of a re-entrant subVI
- Creation
 - Place VI-Defined Memory Configuration Node from Memory & FIFO palette on block diagram
 - Double-click on node to launch Memory Properties Dialog Box



ni.com/training

Memory Write and Read

- Add Memory Method node from Memory & FIFO palette
- Select the memory item to be modified
 - Right-click node and choose Select Memory
- Choose to read from or write to the memory item
 - Right-click the node and choose Select Method

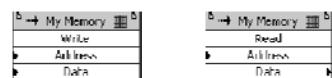


NATIONAL INSTRUMENTS

ni.com/training

Memory Write and Read

- Address – The location of the data in memory on the FPGA target
- Data – The data to be written to or read from the memory on the FPGA target
- Add error terminals to perform error handling



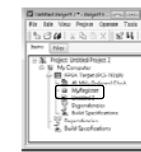
NATIONAL INSTRUMENTS

ni.com/training

Sharing Latest Data – Register Items

- Similar to memory items
 - Only the most recent data is stored
 - Target-scoped and VI-defined options for creation

Target-scoped



NATIONAL INSTRUMENTS

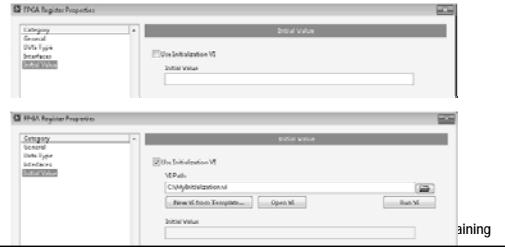
VI-defined



NATIONAL INSTRUMENTS

Register Items

- Similar to memory items
 - Can initialize value through configuration window or an initialization VI



Register Items

- Differences from Memory items:
 - Register items access or store data in the registers of the FPGA
 - Register items do not consume block memory, which is the most limited type of FPGA resource
 - Only share one element (a memory item can share a block of elements)
 - Can share data across multiple clock domains



ni.com/training

Sharing Latest Data – Race Conditions

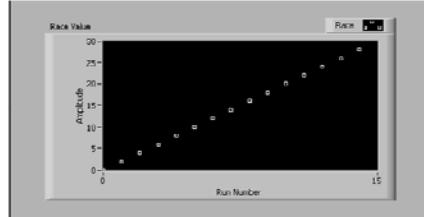
Race conditions can result from accessing a shared resource at multiple locations.

- What is the value of the Race Value after executing this VI the first time?
- After the second?
- After the third?



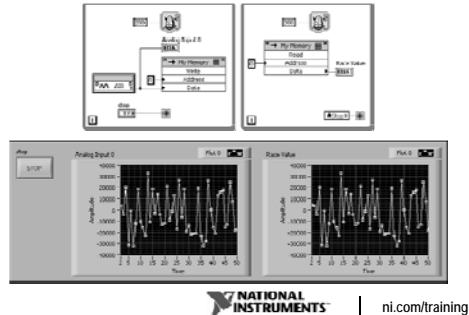
ni.com/training

Race Conditions – Possible Result



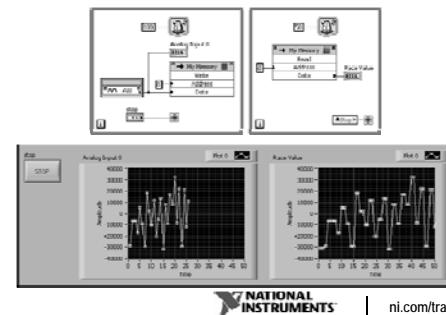
ni.com/training

Parallel Loops – Write and Read at Same Rate



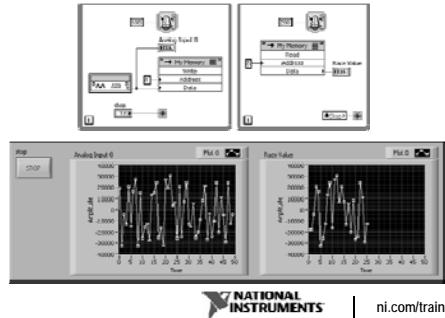
ni.com/training

Parallel Loops – Oversampling



ni.com/training

Parallel Loops - Undersampling



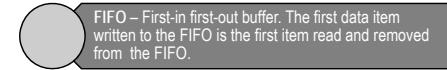
Avoiding Race Conditions

- Control shared resources
- Properly sequence instructions
- Reduce use of variables
- Identify and protect critical sections within your code
 - Pass data between parallel loops by using FIFOs



ni.com/training

D. Transferring Buffered Data – FPGA FIFOs



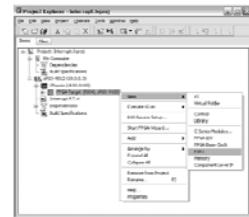
- Target-Scoped – A single FIFO transfers data between loops on multiple VIs running on the same FPGA target
- VI-Defined – A single FIFO transfers data between multiple loops in the same VI



ni.com/training

Target-Scoped FIFOs

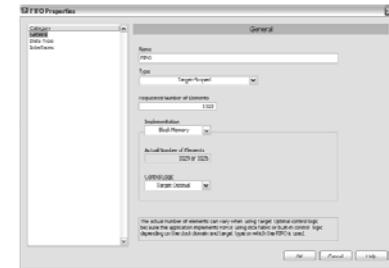
- Appear under the FPGA Target in the Project Explorer Window
- Use to store data that you need to access from multiple VIs
- Create Target-Scoped FIFOs From the Project Explorer window



NATIONAL INSTRUMENTS

ni.com/training

FIFO Properties Dialog Box



NATIONAL INSTRUMENTS

ni.com/training

FIFO – Number of Elements

- Maximum depends on the Implementation that has been selected and the amount of space available on the FPGA for the implementation
- You request the number of elements based on the needs of the application. LabVIEW specifies the actual number of elements. If the actual number is greater than the requested number, the difference is due to technical constraints.

NATIONAL INSTRUMENTS

ni.com/training

FIFO – Implementation

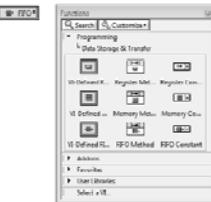
- Flip-Flops – Use gates on the FPGA to provide the fastest performance. Recommended only for very small FIFOs <100 bytes.
- Look-up Table – Store data in look-up tables available on the FPGA. Recommended only for small FIFOs that are 100 to 300 bytes.
- Block Memory – Store data in block memory to preserve FPGA gates and LUTs for your VI. Recommended for FIFOs > 300 bytes.



ni.com/training

VI-Defined FIFOs

- Create VI-Defined FIFOs using the VI-Defined Configuration Node from the Memory & FIFO Palette
- Right-click on the VI-Defined FIFO Configuration Node and select Configure to launch the FIFO Properties dialog box



ni.com/training

FPGA FIFO Write and Read

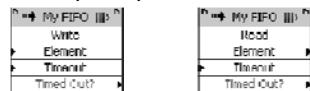
- Place a FIFO Method from the Memory & FIFO palette
- Right-click the object and choose Select FIFO
 - Associates node with a defined FIFO
- Right-click the object and choose Select Method
 - Choose Write or Read



ni.com/training

FPGA FIFO Write and Read

- Element – The data element to write or read
- Timeout – Sets number of ticks that the method waits for available space or data if the FIFO is full or empty
- Timed Out? – True if attempt to write or read failed. Does not overwrite or add new element.
 - FIFO transfer may be lossy if write times out



NATIONAL INSTRUMENTS

INSTRUMENTS

ni.com/training

FIFO Overflow and Underflow

- Overflow – Write loop executing faster than read loop
 - FIFO is filled and the FIFO Write method times out
 - Data is not written to the FIFO until space is available in the FIFO
 - Space can be created by reading data or resetting the FIFO
 - Data is lost until space is made available.
- Underflow – Write loop executing slower than read loop
 - FIFO is empty and the FIFO Read method times out

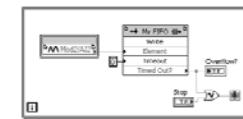
NATIONAL INSTRUMENTS

ni.com/training

Handling FIFO Overflow and Underflow

Method for handling overflow/underflow depends on importance of lossless transfer

- Latch Timed Out? output of read/write method so that operator can see if a timeout ever occurred
- Use Timed Out? as a condition for termination of loop execution



NATIONAL INSTRUMENTS

ni.com/training

Demonstration – FIFO Bucket

Test how a FIFO works by reading and writing data and observing the effect on the FIFO status.

DEMONSTRATION

Get Number of Elements to Read/Write

- Get Number of Elements to Read
 - Returns the number of elements remaining to be read
- Get Number of Elements to Write
 - Returns the number of empty elements remaining in the FIFO

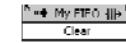
These methods can be used to avoid underflow/overflow conditions by monitoring the status of the FIFO.



ni.com/training

Clear Method

- FIFOs retain data if the FPGA is stopped and restarted
- Use the Clear Method to empty a target- scoped or VI-defined FIFO
 - Place FIFO Method node
 - Right-click on object and choose Select Method»Control»Clear



ni.com/training

Exercise 8-1 Accelerometer Threshold

Modify a VI on the FPGA with parallel loops that pass data using an FPGA FIFO.

GOAL

Exercise 8-1 Accelerometer Threshold

- Why is it important to configure the FIFO data type to match the data that you will be writing to it?
- Could this application have been developed using a VI-defined FIFO?
- What is the purpose of the FIFO Clear method in this exercise?

DISCUSSION

E. Comparing Data Sharing Methods

Transfer Method	FPGA Resource	Lossy?	Common Use
Variables	Logic	Yes	Share latest data
Memory Items	Memory	Yes	Share latest data
Register Items	Logic	Yes	Share latest data
Flip-Flop FIFOs	Logic	No	Transfer buffered data (FIFOs < 100 bytes)
Look-Up Table FIFOs	Logic	No	Transfer buffered data (FIFOs that are 100 to 300 bytes)
Block Memory FIFOs	Memory and Logic	No	Transfer buffered data (FIFOs > 300 bytes)

INSTRUMENTS

ni.com/training

Summary—Quiz

1. Which of the following is NOT a method for passing data between parallel loops on an FPGA?
 - a. Local variable
 - b. Memory item
 - c. Register item
 - d. FIFO
 - e. A wire containing the data



| ni.com/training

Summary—Quiz Answer

1. Which of the following is NOT a method for passing data between parallel loops on an FPGA?
 - a. Local variable
 - b. Memory item
 - c. Register item
 - d. FIFO
 - e. A wire containing the data



| ni.com/training

Summary—Quiz

2. Which of the following may result if an FPGA VI with a FIFO Read or Write does not monitor the Timed Out? output, assuming that the Timeout is not -1 (infinite)? (Select all answers that apply)
 - a. Attempts to write when there is no room in the FIFO may go undetected
 - b. Attempts to read from an empty FIFO may be misinterpreted as having produced valid data
 - c. The FIFO may reset itself automatically
 - d. LabVIEW will report a code generation error
 - e. None of the above



| ni.com/training

Summary—Quiz Answer

2. Which of the following may result if an FPGA VI with a FIFO Read or Write does not monitor the Timed Out? output, assuming that the Timeout is not -1 (infinite)? (Select all answers that apply)
- a. Attempts to write when there is no room in the FIFO may go undetected
 - b. Attempts to read from an empty FIFO may be misinterpreted as having produced valid data
 - c. The FIFO may reset itself automatically
 - d. LabVIEW will report a code generation error
 - e. None of the above



ni.com/training

Summary—Quiz

Match each method of FIFO implementation with its recommended usage.

- | | |
|------------------|--|
| 1. Block Memory | a. For FIFOs < 100 bytes |
| 2. Flip-Flops | b. For FIFOs that are 100 to 300 bytes |
| 3. Look-Up Table | c. For FIFOs > 300 bytes |



ni.com/training

Summary—Quiz

Match each method of FIFO implementation with its recommended usage.

- | | |
|------------------|--|
| 1. Block Memory | a. For FIFOs < 100 bytes |
| 2. Flip-Flops | b. For FIFOs that are 100 to 300 bytes |
| 3. Look-Up Table | c. For FIFOs > 300 bytes |



ni.com/training

Lesson 9

Synchronizing FPGA Loops and I/O

TOPICS

- A. Synchronizing FPGA Loops
- B. Synchronizing C Series I/O Modules



ni.com/training

A. Synchronizing FPGA Loops

Many FPGA applications require synchronizing multiple FPGA loops

- Synchronize acquisition and processing loops
- Synchronize execution between multiple FPGA loops
- Synchronize tasks to a common signal (e.g. sample clock, external clock, etc)
- Synchronize I/O from different C Series modules

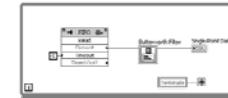
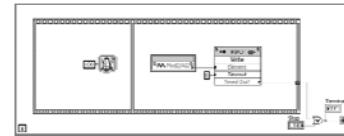


ni.com/training

Synchronize FPGA Loops – FIFOs (Review)

Synchronize acquisition and processing loops

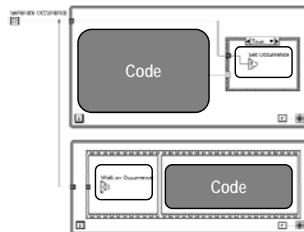
- Top loop places elements into the FIFO
- Bottom loop executes when there are elements in the FIFO



ni.com/training

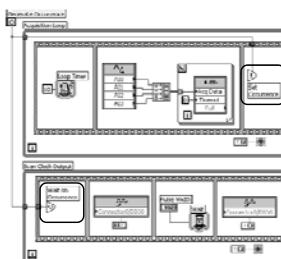
Synchronize FPGA Loops – Occurrence

- You can synchronize execution between multiple FPGA loops by using occurrences



Synchronize FPGA Loops – Occurrence

- Another example of using occurrences



Occurrence

Synchronize the execution of code in multiple FPGA loops using occurrences

<Exercises>\...\Demonstrations\Synchronize FPGA Loops\Synchronize FPGA Loops.lvproj

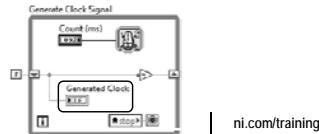
Open Occurrences.vi.

DEMONSTRATION

Synchronize FPGA Loops – Share Common Signal

You can synchronize execution between multiple FPGA loops by generating and sharing a common signal

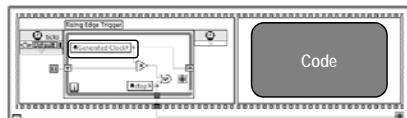
- Generate Boolean signal to provide synchronization among FPGA loops
- Trigger execution in other FPGA loops by monitoring the Boolean signal



Synchronize FPGA Loops – Share Common Signal

- Monitor signal in FPGA loops
– Trigger on rising edge

1. Wait for trigger 2. Execute triggered code



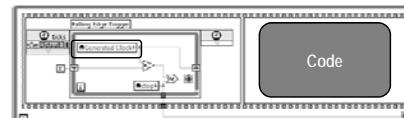
NATIONAL INSTRUMENTS

ni.com/training

Synchronize FPGA Loops – Share Common Signal

- Monitor signal in FPGA loops
– Trigger on falling edge

1. Wait for trigger 2. Execute triggered code



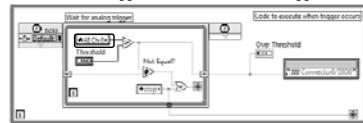
NATIONAL INSTRUMENTS

ni.com/training

Synchronize FPGA Loops – Share Common Signal

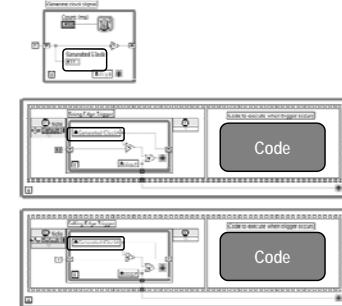
- Monitor signal in FPGA loops
 - Trigger on analog edge

1. Wait for trigger 2. Execute triggered code



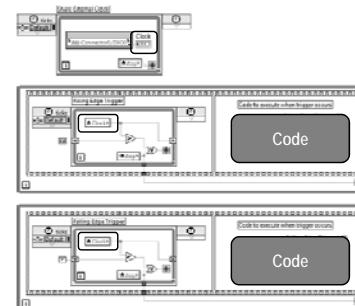
ni.com/training

Synchronize FPGA Loops – Share Common Signal – Generate and Share Clock Example



ni.com/training

Synchronize FPGA Loops – Share Common Signal – Share External Clock Example



ni.com/training

Share Common Signal

Synchronize the execution of code in multiple FPGA loops by sharing a common signal
<Exercises>\..\Demonstrations\Synchronize FPGA Loops\Synchronize FPGA Loops.lvproj
Open Share boolean signal.vi and Share external clock.vi.

DEMONSTRATION

B. Synchronizing C Series Modules

- Some applications using C Series I/O modules require a high level (<100 ns) of synchronization between channels
 - Vibration or sound measurement
 - Multiple types of measurements



ni.com/training

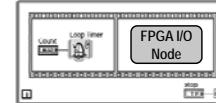
Synchronizing On-Demand C Series Modules



On-demand modules – modules that use the Loop Timer VI to time their loop rate

- Example on-demand modules

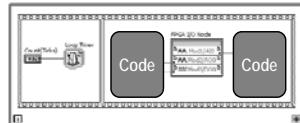
- Digital modules (NI 9474, 9401)
- Analog modules with SAR ADC (NI 9215, 9263)
- Multiplexed modules (NI 9211, 9207)



ni.com/training

Synchronizing On-Demand C Series Modules

- Place all channel reads or updates in the same I/O node
 - Can mix AI, AO, DIO channels with minimal skew

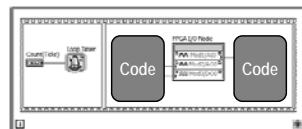


NATIONAL INSTRUMENTS

| ni.com/training

Synchronizing On-Demand C Series Modules

- The maximum loop rate possible is limited by the slowest channel in the FPGA I/O node
 - Digital channels can execute in one clock tick
 - Analog channels require multiple clock ticks
 - Each additional channel in multiplexed modules requires additional clock ticks



| ni.com/training

Synchronizing Delta-Sigma C Series Modules

Delta-sigma modules

- Many modules designed for high-speed, dynamic measurements use a delta-sigma ADC
- Internally timed
- Use Data Rate property to time their loop rates

- Examples of delta-sigma modules

- NI 923x
- NI 9225, 9227, 9229



NATIONAL INSTRUMENTS

| ni.com/training

Synchronizing Delta-Sigma C Series Modules

- You can synchronize multiple delta-sigma modules and different delta-sigma models
- Modules must use the same master timebase source
- Modules must start acquisition at the same time



NATIONAL
INSTRUMENTS

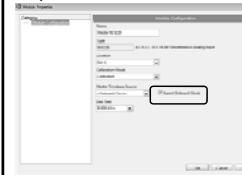
| ni.com/training

Synchronizing Delta-Sigma C Series Modules

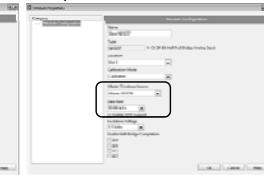
Configure modules in Project Explorer

- Select one of the modules, “the master”, to export its clock
- Set the rest of the modules to import this clock

Export clock



Import clock



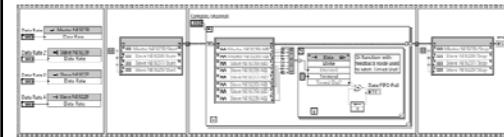
NATIONAL
INSTRUMENTS

| ni.com/training

Synchronizing Delta-Sigma C Series Modules

Synchronize modules on the block diagram

- Set Data Rate property node for each module to same value
- Use the same I/O node to start all modules
- Place all channels in the same I/O node



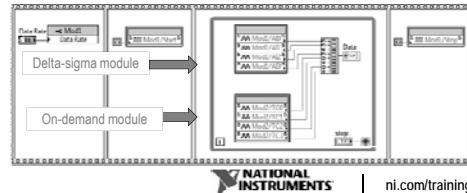
NATIONAL
INSTRUMENTS

| ni.com/training

Synchronizing Delta-Sigma & On-Demand C Series Modules

1. Design your application for delta-sigma module timing using Data Rate property nodes
2. Add on-demand I/O modules without using Loop Timer VI

Note: This synchronizes all modules to a delta-sigma module clock



NATIONAL INSTRUMENTS

ni.com/training

Summary—Quiz

1. Which of the following methods can you use to synchronize execution between multiple FPGA loops?
 - a. Occurrences
 - b. Share common signal

NATIONAL INSTRUMENTS

ni.com/training

Summary—Quiz Answers

1. Which of the following methods can you use to synchronize execution between multiple FPGA loops?
 - a. Occurrences
 - b. Share common signal

NATIONAL INSTRUMENTS

ni.com/training

Summary—Quiz

2. Which of the following do you need to synchronize acquisition from analog input channels from multiple delta-sigma C Series modules?
 - a. Time the acquisition using a Loop Timer Express VI
 - b. Configure modules to use the same master timebase source
 - c. Start acquisition at the same time
 - d. Acquire from all channels in the same FPGA I/O node



| ni.com/training

Summary—Quiz Answers

2. Which of the following do you need to synchronize acquisition from analog input channels from multiple delta-sigma C Series modules?
 - a. Time the acquisition using a Loop Timer Express VI
 - b. Configure modules to use the same master timebase source
 - c. Start acquisition at the same time
 - d. Acquire from all channels in the same FPGA I/O node



| ni.com/training

Lesson 10
Modular Programming

TOPICS

- A. Review of SubVIs
- B. Using SubVIs on the FPGA
- C. Reentrancy and Non-reentrancy in FPGA
- D. Passing FPGA Items to SubVIs
- E. Testing FPGA SubVIs

 ni.com/training

A. Review – Understanding Modularity

- Modularity defines the degree to which a program is composed of discrete modules such that a change to one module has minimal impact on other modules
- Modules in LabVIEW are called subVIs

Review – SubVIs

- A VI within another VI is a subVI
- The upper right corner of the front panel and block diagram displays the icon for the VI
- This icon identifies the VI when you place the VI on the block diagram



ni.com/training



ni.com/training

Review - Icon and Connector Pane



- After you build a VI, build the icon and the connector pane so you can use the VI as a subVI
- The icon and connector pane correspond to the function prototype in text-based programming languages
- Every VI displays an icon in the upper-right corner of the front panel and block diagram windows
- An icon is a graphical representation of a VI
- If you use a VI as a subVI, the icon identifies the subVI on the block diagram of the VI



ni.com/training

Review - Icon and Connector Pane – Setting up the Connector Pane

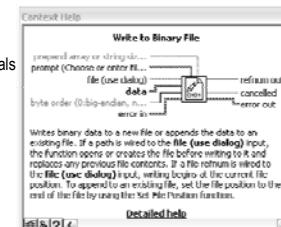
- Right-click the icon in the upper right corner of the front panel and select Show Connector
 - Each rectangle on the connector pane represents a terminal
 - Use the terminals to assign inputs and outputs
- Select a different pattern by right-clicking the connector pane and selecting Patterns from the shortcut menu



ni.com/training

Review - Using SubVIs – Terminal Setting

- Bold:** Required terminal
- Plain:** Recommended terminals
- Dimmed:** Optional terminals

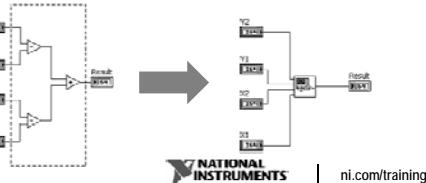


ni.com/training

Review - Using SubVIs – Section to SubVI

To convert a section of a VI into a subVI:

- Use the Positioning tool to select the section of the block diagram you want to reuse
- Select **Edit>Create SubVI**



NATIONAL INSTRUMENTS

ni.com/training

B. Using SubVIs on the FPGA

- You can run only one top-level FPGA VI
- You can use multiple VIs on the FPGA by placing subVIs on the block diagram of the top-level FPGA VI
- Placing large controls and indicators on the front panel of a subVI is acceptable
 - SubVI controls and indicators are not exposed to the host

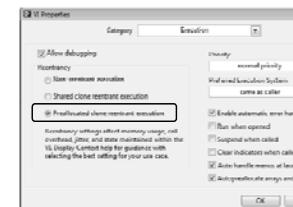
NATIONAL INSTRUMENTS

ni.com/training

C. Reentrancy and Non-reentrancy in FPGA

Reentrant FPGA SubVI configuration

- Default configuration of VIs created under an FPGA target
- Multiple calls to the same subVI run in parallel using separate FPGA resources

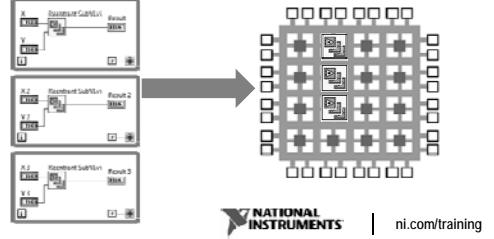


NATIONAL INSTRUMENTS

ni.com/training

Reentrant FPGA SubVI

- Each instance of the reentrant FPGA subVI on the block diagram becomes a separate hardware resource

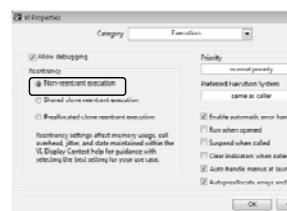


NATIONAL INSTRUMENTS

ni.com/training

Non-reentrant FPGA SubVI

- Single instance shared among multiple callers
- Each call to the subVI waits until the previous call ends
- Does not allow parallel execution

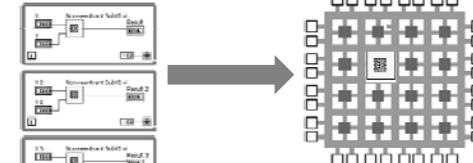


NATIONAL INSTRUMENTS

ni.com/training

Non-reentrant FPGA SubVI

- Only a single copy of the non-reentrant FPGA subVI becomes hardware and all callers share the hardware resource



NATIONAL INSTRUMENTS

ni.com/training

Reentrancy and Non-reentrancy in FPGA

Tradeoffs

- Reentrant is the default VI Type for FPGA VIs

Comparison	Reentrant SubVI	Non-reentrant SubVI
FPGA Resources for VI Logic	Separate for each instantiation	Single set for all instantiations
Parallel Execution	Yes	No
Optimized for...	Speed	Size (Typically)
Use in Single-Cycle Timed Loop	Yes	Only if called from one place



ni.com/training

D. Passing FPGA Items to SubVIs

You can pass I/O, clocks, register items, memory items, and FIFOs to subVIs by using the following FPGA name controls:

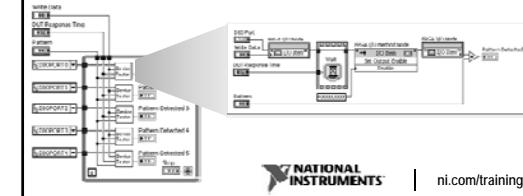
- FPGA I/O
- FPGA Clock
- Memory
- Register
- FIFO



ni.com/training

FPGA I/O Control

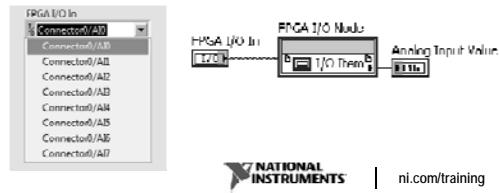
- Passes FPGA I/O items to subVIs
- Create VIs that can be used as reentrant subVIs with configurable I/O items



ni.com/training

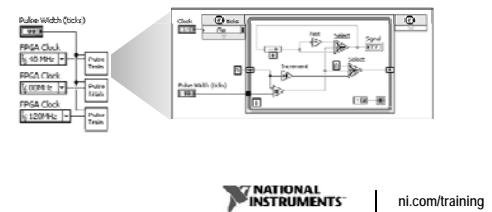
FPGA I/O Control

- Place FPGA I/O Node in subVI
- Right-click FPGA I/O In input and select Create»Control
- Wire the FPGA I/O control to the subVI connector pane



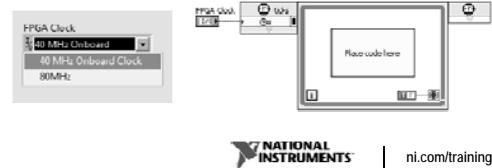
FPGA Clock Control

- Pass FPGA Clocks to subVIs



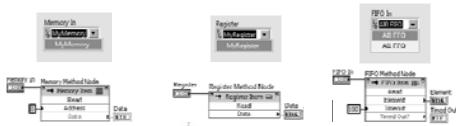
FPGA Clock Control

- Place SCTL in subVI
- Right-click the Source Name input of an SCTL and select Create»Control.
- Wire the clock control to the subVI connector pane



FPGA Memory/Register/FIFO Control

- Pass memory, register, and FIFO items to subVIs to create reusable code
- Right-click the input of a Memory, Register, or FIFO Method node and select Create»Control
- Wire the memory, register, or FIFO control to the subVI connector pane



FPGA Control Restrictions

- You cannot change the value of FPGA controls and indicators while an FPGA VI runs on the development computer or when using Interactive Front Panel communication
- You cannot access FPGA controls and indicators from the FPGA Interface
- You can bundle FPGA name controls into clusters with other FPGA name controls, with occurrence refnums, and with other data types. However, you cannot have such a cluster on the front panel of the top-level FPGA VI. LabVIEW returns an error when you attempt to compile the FPGA VI.



ni.com/training

E. Testing FGPA SubVIs

Test subVIs as top-level VIs

- Execute on development computer
- Execute on FPGA target



ni.com/training

Exercise 10-1: Creating an FPGA SubVI

Create an FPGA subVI and call it from an FPGA main VI.

GOAL

Exercise 10-1: Creating an FPGA SubVI

- If you needed to monitor additional AI channels and control additional DIO lines, how would you change your code?
- How would the behavior of this application change if you changed the subVI from reentrant to non-reentrant?

DISCUSSION

Summary—Quiz

1. True or False? Controls and indicators on an FPGA subVI are exposed to the host VI.



| ni.com/training

Summary—Quiz Answer

1. True or False? Controls and indicators on an FPGA subVI are exposed to the host VI.

False



| ni.com/training

Summary—Quiz

2. Which of the following are true for reentrant subVIs in LabVIEW FPGA?
- a) Default configuration of VIs created under an FPGA target
 - b) Each instance on the block diagram becomes a separate hardware resource
 - c) Optimized for FPGA size
 - d) Each call to the subVI waits until the previous call ends
 - e) Multiple calls to the same subVI run in parallel



| ni.com/training

Summary—Quiz Answers

2. Which of the following are true for reentrant subVIs in LabVIEW FPGA?
- a) Default configuration of VIs created under an FPGA target
 - b) Each instance on the block diagram becomes a separate hardware resource
 - c) Optimized for FPGA size
 - d) Each call to the subVI waits until the previous call ends
 - e) Multiple calls to the same subVI run in parallel



| ni.com/training

Summary—Quiz

3. Which of the following are FPGA name controls that can be passed into FPGA subVIs?
- a) FPGA I/O control
 - b) FPGA Clock control
 - c) FPGA FIFO control
 - d) FPGA Memory control
 - e) FPGA Register control



| ni.com/training

Summary—Quiz Answers

3. Which of the following are FPGA name controls that can be passed into FPGA subVIs?
- a) FPGA I/O control
 - b) FPGA Clock control
 - c) FPGA FIFO control
 - d) FPGA Memory control
 - e) FPGA Register control



| ni.com/training

Lesson 11
Communicating Between the FPGA and Host

TOPICS

- A. Programmatically Communicating with the FPGA VI from the Host
- B. Deploying an FPGA VI
- C. Transferring Latest Data
- D. Transferring Buffered Data
- E. Synchronizing the Host VI and FPGA VI
- F. Implementing an FPGA Watchdog

NATIONAL INSTRUMENTS | ni.com/training

A. Programmatically Communicating with the FPGA VI from the Host

You use a host VI to send information between the host computer and the FPGA target for the following reasons

- Do more data processing than you can fit on the FPGA
- Perform operations not available on the FPGA target, such as logging data to file
- Control the timing and sequencing of data transfer
- Create a test bench for an FPGA VI

Host Computer FPGA

```
graph LR; Host[Host Computer] -- "Communication" --> FPGA[FPGA];
```

ni.com/training

Host Computer – Windows-Based Computer

Example

- R Series in Windows PC
- R Series in Windows PXI

Windows Computer (PC or PXI)

```
graph LR; subgraph HostComputer [Windows Computer (PC or PXI)]; direction TB; subgraph HostVI [Host VI]; end; subgraph LabVIEW [LabVIEW for Windows]; end; subgraph FPGA [FPGA]; direction TB; subgraph FPGAVI [FPGA VI]; end; subgraph LabVIEWFPGA [LabVIEW FPGA]; end; HostVI <-->|Communication| LabVIEWFPGA;
```

NATIONAL INSTRUMENTS | ni.com/training

Developing a Host VI on Windows-Based Host Computer

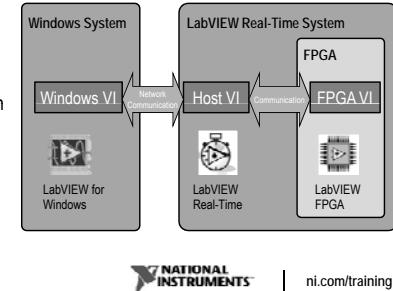
Add a VI under My Computer



ni.com/training

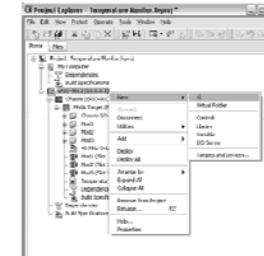
Host Computer – RT Target

- Example
- cRIO
 - sbRIO
 - R Series in RT PXI



Developing Host VI on RT Target Host Computer

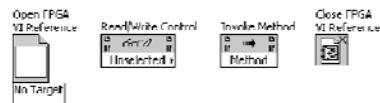
Add a VI under the RT target



ni.com/training

FPGA Interface Functions

- Establish communication with a FPGA VI from a host VI
- Control the execution of the FPGA VI on the FPGA target
- Read and write data to the FPGA VI
- Terminate communication with the FPGA VI

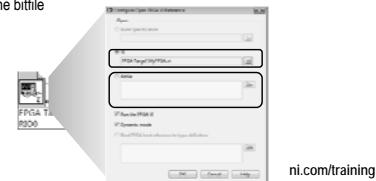


NATIONAL INSTRUMENTS

ni.com/training

Open FPGA VI Interface Function

- Opens a reference to the FPGA so you can communicate between the host and FPGA
- Specify an FPGA VI or bitfile
 - VI: Uses the bitfile associated with the FPGA VI
 - Bitfile: Uses the specified bitfile
- If the bitfile is not loaded on the FPGA target, this function deploys and runs the bitfile and opens a reference to it
- If the bitfile is already running on the FPGA target, this function just opens a reference to the bitfile

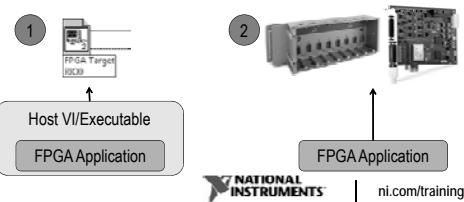


ni.com/training

B. Deploying an FPGA VI

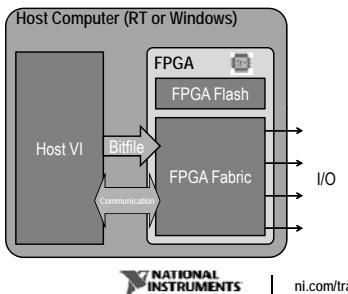
Two methods to deploy an FPGA bitfile:

1. Host loads bitfile with Open FPGA VI Reference VI
2. Bitfile loaded independently from FPGA flash memory



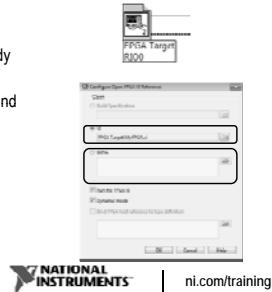
NATIONAL INSTRUMENTS | ni.com/training

Host Reference Deployment Method – Hardware Overview

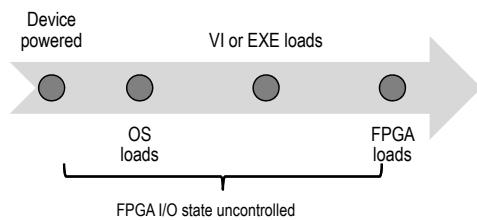


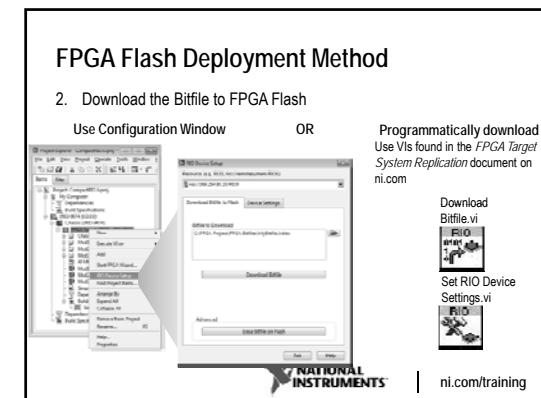
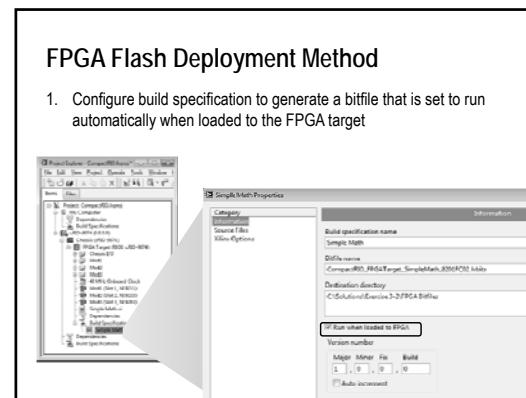
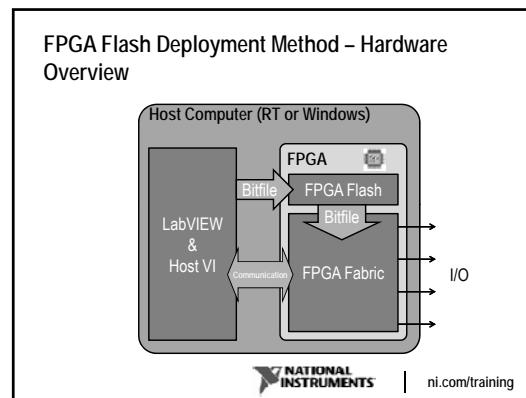
Host Reference Deployment Method

- Open FPGA VI Reference function checks if bitfile is loaded and loads if not already loaded
- Caveat: Bitfile is not loaded and run until after the OS and application have loaded

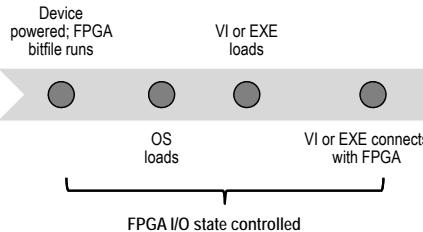


Host Reference Deployment Method – Timing





FPGA Flash Deployment Method – Timing

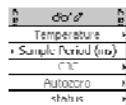


| ni.com/training

C. Transferring Latest Data – Programmatic Front Panel Communication

Read/Write Control function

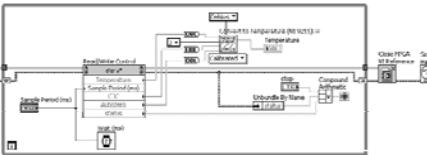
- Use in the Host VI to read from or write to controls and indicators of the top-level FPGA VI
- Only transfers the latest data to and from the controls and indicators of the FPGA VI



| ni.com/training

Transfer Latest Data – Programmatic Front Panel Communication

Host VI



FPGA VI



| ni.com/training

Transfer Latest Data – Programmatic Front Panel Communication

Use cases

- Sending configuration parameters or data from the host to the FPGA
- Reporting status from the FPGA to the host
- Sending commands between the host and the FPGA
- Transferring latest data between the host and the FPGA
- Creating a test bench for an FPGA VI



ni.com/training

FPGA Interface Functions – Invoke Method

- Invokes method or action from a host VI
- The following methods are supported by most targets:
 - Run
 - Abort
 - Get FPGA VI Execution Mode
 - Reset
 - Wait on IRQ
 - Acknowledge IRQ
 - Download
 - DMA FIFOs
- Your target may support additional target-specific methods



ni.com/training

FPGA Interface Functions – Close FPGA VI Reference

- By default, stops and resets the FPGA
- Right-click and select Close from the shortcut menu to close the reference without resetting
- Default is Close and Reset if Last Reference, which closes the reference, stops the FPGA VI, and resets the FPGA
- Use free label to describe functionality



ni.com/training

FPGA Interface Functions

- FPGA Interface functions on the host VI control and communicate with the FPGA VI
- Expose only necessary controls and indicators on the FPGA VI
- Use the host VI to write values to the FPGA VI
 - Do not write values to the FPGA VI using the front panel of the FPGA VI



| ni.com/training

Exercise 11-1: Windows Host Integration

Create a user interface on a Windows host system that interacts with an FPGA VI on a simulated PCIe-7852R R-Series board.

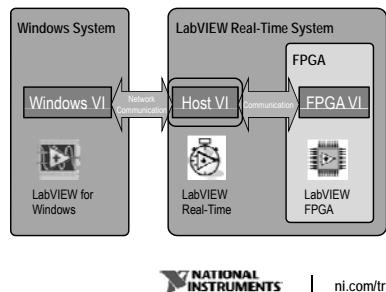
GOAL

Exercise 11-1: Windows Host Integration

- Will the same scaling calculations work for I/O channels that have a different resolution and voltage range?
- What functionality can a Windows host VI accomplish that a FPGA VI could not?

DISCUSSION

LabVIEW FPGA Architecture with LabVIEW Real-Time: RT Host VI



Introduction to Real-Time

Advantages of Using a Real-Time Host

- Code execution is more deterministic than Windows OS
- Ability to set tasks to run at different priorities including a time-critical priority
- Ability to separate deterministic tasks from non-deterministic tasks
- Real-time OS is more stable than Windows OS
- Real-time targets can have 1MHz clocks (Windows has a 1kHz clock)



ni.com/training

RT Host VI Common Tasks

- Data processing
- Perform operations not available on the FPGA target
- Floating-point math
- Sequence multiple FPGA VIs
- Log data
- Run multiple VIs
- Control the timing and sequencing of data transfer
- Coordinate operation of several FPGA targets
- Coordinate operation of FPGA targets with other measurement devices in the computer
- Control which components are visible on the Windows user interface VI if applicable



ni.com/training

LabVIEW Real-Time 1 Course

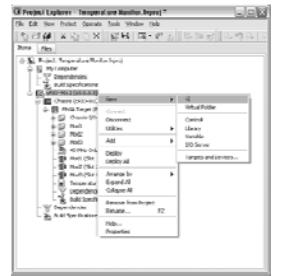
- Configuring Real-Time hardware
- Shared Variable with RT FIFO
- Real-Time architecture and concepts
- Network Streams
- Network-published Shared Variables
- Improving determinism
- And more
- Setting priorities of processes
- Passing data between processes
- Network communication



ni.com/training

Developing an RT Host VI

Add a VI under the RT target

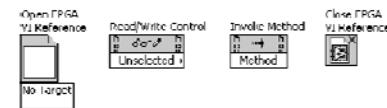


ni.com/training

Developing an RT Host VI

FPGA Interface Functions

- Used the same way whether programming a Windows Host VI or RT Host VI

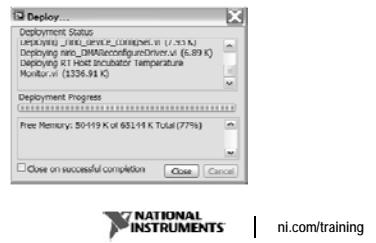


ni.com/training

Developing an RT Host VI

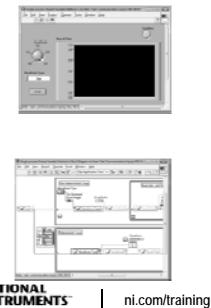
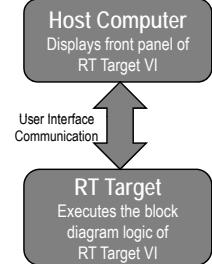
After developing the RT host VI, run it

- Deploying Status Window:



ni.com/training

Front Panel Communication in LabVIEW Real-Time



NI NATIONAL INSTRUMENTS

ni.com/training

Front Panel Communication in LabVIEW Real-Time

Advantages

- Debug VIs
- Quickly monitor VIs running on RT target during development

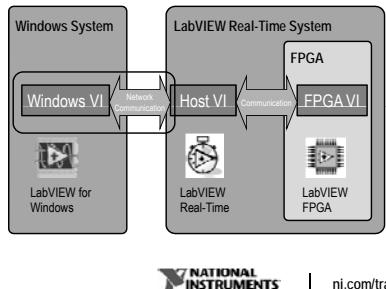
Caveats

- You should not use Front Panel Communication for a final RT application
- Learn how to build a proper final RT application in the *LabVIEW Real-Time 1* course

NI NATIONAL INSTRUMENTS

ni.com/training

LabVIEW FPGA Architecture with LabVIEW Real-Time: Windows VI



Developing a Windows VI

- Windows VI possible tasks:
- Act as a user-interface
 - Send data and commands to RT host VI
 - .NET and other Windows-compatible functionality
 - Database connectivity
 - Log data
 - Data processing



ni.com/training

Developing a Windows VI

- Use network communication to transfer data between a Windows system and an RT target
- Network communication allows you to:
 - Run a VI on the RT target and a VI on the Windows system
 - Communicate between the RT target VI and Windows VI
 - Control timing and sequencing of the data transfer
 - Perform additional data processing or logging



ni.com/training

Shared Variable Network Communication

- Network-published shared variables automatically publish data values over an Ethernet network
- Shared variables communicate between
 - Parallel processes
 - Between VIs
 - Between computers



ni.com/training

Creating Shared Variables

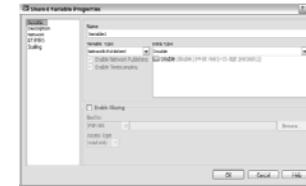
- Create Shared Variables from the Project Explorer window
- Shared Variables must be inside project libraries
- LabVIEW automatically creates the library with the shared variable inside



ni.com/training

Shared Variable Configuration

- Configuring Network-published shared variables
 - Set Variable Type to Network-published



ni.com/training

Shared Variable with RT FIFO

RT FIFO category

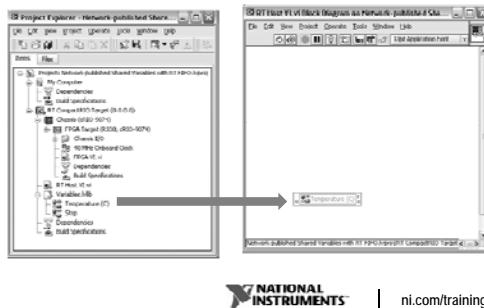
- Enable RT FIFO
 - Shares data across a network without affecting the determinism of the VIs
- FIFO Type
 - Single Element
 - Multi-element
 - Number of elements



NATIONAL INSTRUMENTS

ni.com/training

Shared Variable Programming



NATIONAL INSTRUMENTS

ni.com/training

Network Communication Methods

- There are multiple methods of transferring data between the VI running on the RT target and the VI running on the Windows system
 - Network-published shared variables
 - Network-published shared variables with RT FIFO enabled
 - TCP
 - UDP
 - Network Streams
- This is covered in the *LabVIEW Real-Time 1* course

NATIONAL INSTRUMENTS

ni.com/training

Exercise 11-2: RT Host and Windows Integration

Develop an RT host VI that communicates with the FPGA and Windows VI that serves as a user interface for the Temperature Monitor project.

GOAL

Exercise 11-2: RT Host and Windows Integration

- Why was the temperature conversion to units of Celsius done in the RT Host VI instead of the FPGA VI?
- What are the advantages of writing data to file on a Windows VI, and what would be the advantages of writing data to file on a RT Host VI?

DISCUSSION

Prepare RT Host VI for Final Application

- For final RT application
 - Do not use Front Panel Communication
 - Compile the RT VI into a startup EXE or startup VI
- For more information, see the *LabVIEW Real-Time 1* and *LabVIEW Real-Time 2* courses



| ni.com/training

Transfer Latest Data – User-Defined I/O Variables

User-Defined I/O Variables

- Useful if your Host VI is running on an RT Target already using the NI Scan Engine and FPGA
- Unidirectional
- Requires NI Scan Engine



ni.com/training

Transfer Latest Data – User-Defined I/O Variables

Use cases

- Similar use cases as programmatic front panel communication but with the timing and synchronization of the NI Scan Engine
- Transfer time-coherent sets of data

Disadvantages

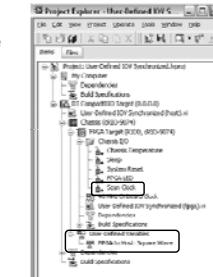
- Including the NI Scan Engine uses up FPGA resources
- Limited by specified Scan Period of the NI Scan Engine



ni.com/training

Transfer Latest Data – User-Defined I/O Variables

- Create User-Defined I/O Variable
 - Right-click Chassis item and select New>User-Defined Variable



ni.com/training

Transfer Latest Data – User-Defined I/O Variables

- Configure User-Defined I/O Variable

The screenshot shows the 'Shared Variable Properties' dialog box for a variable named 'FPGA to Host - Square Wave'. Under the 'Variable' tab, 'Data Type' is set to 'Boolean' and 'Scaling' is set to 'None'. The 'Timing' tab is selected, showing a 'Scan Period' of 100 ms and a 'Scan Interval' of 10 ms. A checkbox for 'Enable Timestamping' is checked. The 'Description' field contains the text 'Boolean' and 'Output device'. At the bottom, there are 'OK', 'Cancel', and 'Help' buttons.

Transfer Latest Data – User-Defined I/O Variables

- Synchronize reads/writes with the Scan Engine scan period using the Scan Clock item

The diagram illustrates the LabVIEW FPGA VI structure. It shows a 'Scan Clock' block connected to a 'Wait on Rising Edge' block. This is followed by a 'Read FIFO' block and a 'Write FIFO' block. The 'Write FIFO' block is connected to a 'Write FIFO to Host - Square Wave' block. The 'Scan Clock' block is also connected to an 'Application Logic' block, which then connects to a 'Scan Clock' block in the Host VI. The Host VI also includes a 'Host Read FIFO' block and a 'Host Read FIFO to Host - Square Wave' block.

Transfer Latest Data – User-Defined I/O Variables

- Synchronize Timed Loop with the Scan Engine

The diagram illustrates the LabVIEW Host VI structure. It shows a 'Host Read FIFO' block connected to a 'Host Read FIFO to Host - Square Wave' block. This block is connected to a 'Scan Engine' block. The 'Scan Engine' block is connected to an 'Application Logic' block, which then connects to a 'Scan Clock' block. The 'Scan Clock' block is connected to a 'Wait on Rising Edge' block, followed by a 'Read FIFO' block and a 'Write FIFO' block. The 'Write FIFO' block is connected to a 'Write FIFO to Host - Square Wave' block. The 'Scan Engine' block is also connected to a 'Host Read FIFO' block and a 'Host Read FIFO to Host - Square Wave' block.

Summary—Quiz

Match the task with the VI in a FPGA/RT architecture

- | | |
|---|------------------------------|
| 1. Logging data to file | A. FPGA VI |
| 2. I/O operations | B. RT Host VI |
| 3. Double-precision floating-point math | C. Windows User Interface VI |
| 4. User interface | |
| 5. 40 MHz logic | |
| 6. Inserting data into an enterprise database | |
| 7. Separating deterministic and non-deterministic tasks | |



| ni.com/training

Summary—Quiz Answers

Match the task with the VI in a FPGA/RT architecture

- | | |
|---|------------------------------|
| 1. Logging data to file (B,C) | A. FPGA VI |
| 2. I/O operations (A) | B. RT Host VI |
| 3. Double-precision floating-point math (B,C) | C. Windows User Interface VI |
| 4. User interface (C) | |
| 5. 40 MHz logic (A) | |
| 6. Inserting data into an enterprise database (C) | |
| 7. Separating deterministic and non-deterministic tasks (B) | |



| ni.com/training

Summary—Quiz

2. Which of the following should you use in the host VI to communicate with the FPGA VI?

- a. Array functions
- b. Network-published shared variables
- c. FPGA Interface Vis
- d. TCP/IP VIs



| ni.com/training

Summary—Quiz Answer

2. Which of the following should you use in the host VI to communicate with the FPGA VI?
 - a. Array functions
 - b. Network-published shared variables
 - c. FPGA Interface VIs
 - d. TCP/IP VIs



ni.com/training

Summary—Quiz

3. Which of the following are necessary in a host VI that reads and writes values of controls and indicators on the FPGA VI?
 - a. Open FPGA VI Reference
 - b. Read/Write Control
 - c. Invoke Method
 - d. Close FPGA VI Reference



ni.com/training

Summary—Quiz Answer

3. Which of the following are necessary in a host VI that reads and writes values of controls and indicators on the FPGA VI?
 - a. Open FPGA VI Reference
 - b. Read/Write Control
 - c. Invoke Method
 - d. Close FPGA VI Reference



ni.com/training

Summary—Quiz

4. True or False? You should use Interactive Front Panel Communication in your final RT application.



| ni.com/training

Summary—Quiz

4. True or False? You should use Interactive Front Panel Communication in your final RT application.

False



| ni.com/training

Summary—Quiz

5. Which method can you use to control the FPGA I/O states at all times including immediately after device power-on or rebooting?
- a. Deploy FPGA VI using Open FPGA VI Reference function on Host VI
 - b. Deploy FPGA VI by downloading the bitfile to the FPGA flash memory



| ni.com/training

Summary—Quiz

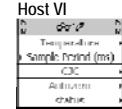
5. Which method can you use to control the FPGA I/O states at all times including immediately after device power-on or rebooting?
 - a. Deploy FPGA VI using Open FPGA VI Reference function on Host VI
 - b. Deploy FPGA VI by downloading the bitfile to the FPGA flash memory



ni.com/training

Limitations of Programmatic Front Panel Communication

- Only the most current data is transferred
- Large controls/indicators use significant logic resources on FPGA
 - Mixed-data clusters
 - Numeric arrays
- Using CPU for host data transfer results in...
 - Transfer speed dependent on CPU processor speed
 - Slower data transfer from target to host
 - Consumption of CPU resources



D. Transferring Buffered Data

Buffer – An area of computer memory that stores multiple data items

- To transfer larger amounts of data at high rates between the FPGA target and the host you must buffer your data
- The best way to transfer buffered data is by using Direct Memory Access (DMA) data transfer methods



ni.com/training

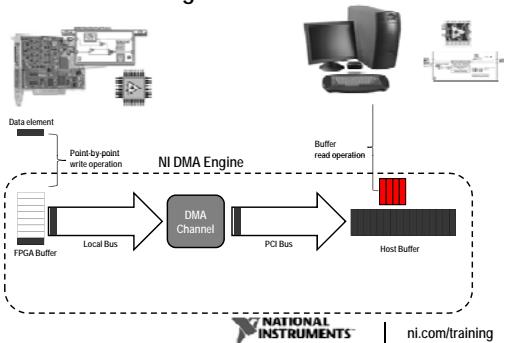
DMA FIFOs

- Direct Memory Access (DMA) – A FIFO-based method of transferring data between an FPGA target and host computer
- Consists of two parts – FPGA and host
- Streams large amounts of data between computer memory and the FPGA
- Unidirectional
- Done mostly without the CPU
- Provides better performance than using the CPU to read and write data to the indicators and controls

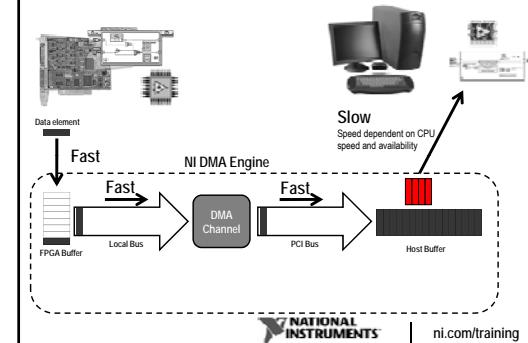


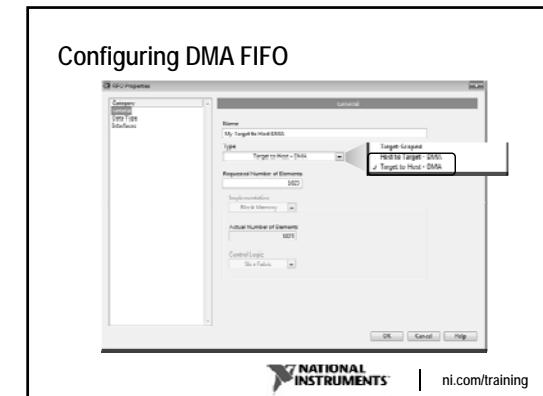
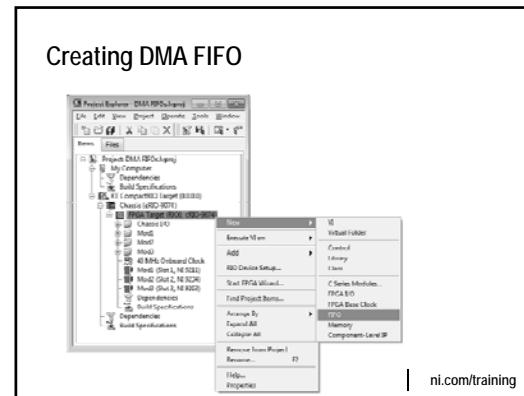
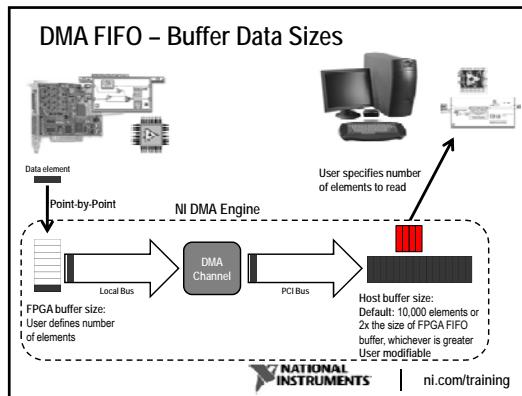
ni.com/training

DMA FIFO – Target to Host Transfer



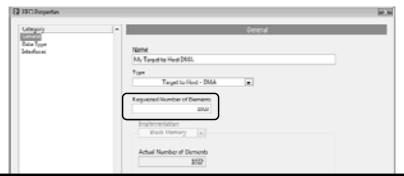
DMA FIFO – Relative Transfer Rates





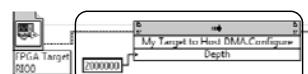
Selecting DMA FIFO Sizes

- FPGA buffer size
 - In the FIFO Properties dialog box of the DMA FIFO, use Number of Elements to set the size of the FPGA buffer size on the FPGA
 - Save resources by choosing the smallest size that is reasonable for your application



Selecting DMA FIFO Sizes

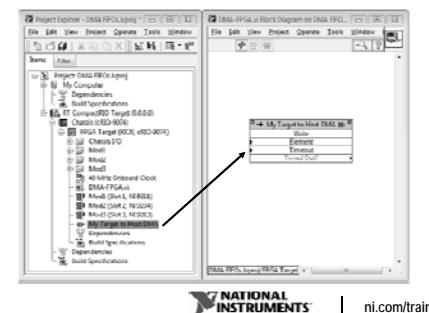
- Host buffer size
 - Default is 10,000 elements or twice the size of FPGA FIFO buffer, whichever is greater
 - Use the Configure Method in the host VI to specify a different size



NATIONAL INSTRUMENTS

ni.com/training

FPGA VI: Using DMA FIFO



FPGA VI: DMA FIFO Write and Read

- Put data into the DMA FIFO with the FIFO Write function
- Retrieve data with the FIFO Read function
- Use Timed Out? to determine the status of the FIFO

NATIONAL
INSTRUMENTS

| ni.com/training

Host VI: Using DMA FIFO

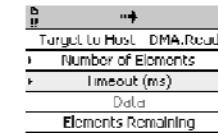
- Open a reference to the FPGA
- Add an Invoke Method function
- For a Host to Target DMA FIFO
 - Choose <DMA FIFO Name>»Read to read data from FPGA
- For a Target to Host DMA FIFO
 - Choose <DMA FIFO Name>»Write to write data to FPGA



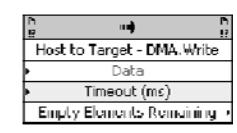
ni.com/training

Host VI: DMA FIFO Read and Write

Target to Host Transfer



Host to Target Transfer

NATIONAL
INSTRUMENTS

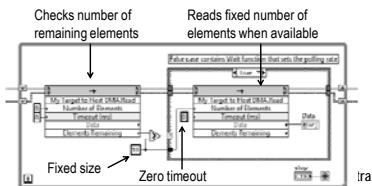
| ni.com/training

Using DMA FIFO Read and Write – Target to Host Transfer

FPGA VI



Host VI



Ensuring Lossless Data Transfer

- Underflow
- Overflow



ni.com/training

Ensuring Lossless Data Transfer – Underflow Error

Underflow Error – If the DMA FIFO Read is configured to read n elements and times out before n or more elements is available, the DMA FIFO Read returns Error -50400

- You will not receive this error if you check if the number of elements you want to read is available before reading



ni.com/training

Ensuring Lossless DMA Transfer – Preventing Overflow

Overflow – FIFO is filled as a result of data being written faster than it is read, data may be lost

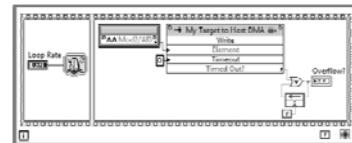
- Reduce the rate at which you write data to the FIFO
- Increase number of elements to read on host
- Increase FPGA and host buffer sizes
- Reduce load on CPU
 - Speed of CPU and competing tasks impact transfer rate from DMA to application memory



ni.com/training

Identifying DMA FIFO Overflow

- Check whether or not the Write Method timed out



ni.com/training

Recovering from a DMA FIFO Overflow

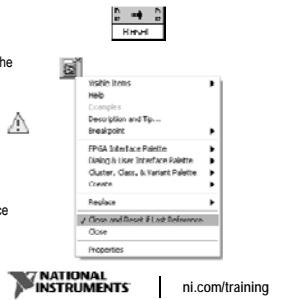
- There are two primary methods for clearing the DMA FIFO when an overflow occurs, depending on whether or not you want to keep the data that is in the DMA FIFO
 - Reset the FPGA VI – Discards the data in the DMA FIFO
 - Flushing the buffer – Reads the remaining data in the DMA FIFO
- Either method can be performed to initialize the DMA FIFO at the beginning of execution



ni.com/training

Clearing DMA FIFO – Reset FPGA VI Method

- Execution on development computer
 - FIFOs automatically reset when the VI stops and restarts
- Execution on FPGA Target
 - FIFOs do not automatically reset
- To reset programmatically:
 - Use the Invoke Method function
Or
 - Use the Close FPGA VI Reference function configured to Close and Reset if Last Reference



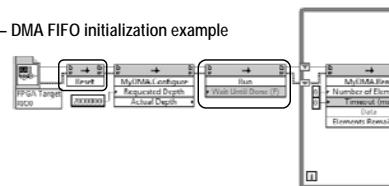
NATIONAL INSTRUMENTS

ni.com/training

Clearing DMA FIFO – Reset FPGA VI Method

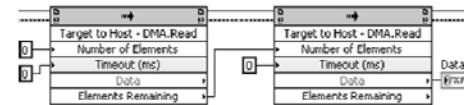
- Reset method will abort and reset the FPGA VI to its default state (clears DMA FIFO)
- Use Run method to run the FPGA VI on the FPGA target again

Host VI – DMA FIFO initialization example



Clearing DMA FIFO – Flush DMA FIFO Buffer Method

- Stop writing to the DMA FIFO
- Find out how much data remains in the DMA FIFO
- Read all of the remaining elements to empty the DMA FIFO



NATIONAL INSTRUMENTS

ni.com/training

Exercise 11-3: Transferring Buffered Data Using DMA FIFO

Transfer buffered data from the FPGA VI to the Windows host VI

GOAL**Exercise 11-3: Transferring Buffered Data Using DMA FIFO**

- What purpose does the shift register and Or function serve in the FPGA VI?
- How does the code handle a DMA FIFO overflow?

DISCUSSION**DMA FIFOs – Interleaving**

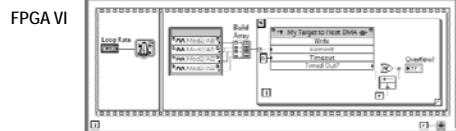
- There is a limited number of DMA FIFO channels available
 - Three channels available for NI 7852R and cRIO-9074
- What if you want to acquire and transfer data from multiple channels on multiple modules?
- You can interleave data of the same type prior to writing to the DMA FIFO



ni.com/training

DMA FIFO Interleaving – FPGA VI

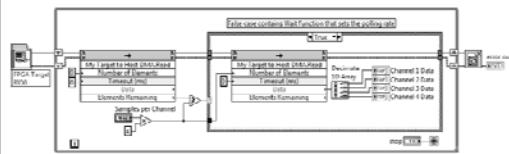
- Combine multiple elements of the same data type into a single array
 - Module 1 Channel 0 → Index 0
 - Module 1 Channel 1 → Index 1
 - Etc...
- Iterate through that array, writing the data one element at a time to the DMA FIFO



DMA FIFO De-Interleaving – Host VI

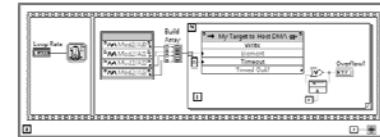
- Read data from the DMA FIFO
- Divide the data into multiple arrays, each representing a single channel of acquisition
- Process the data for each channel

Host VI

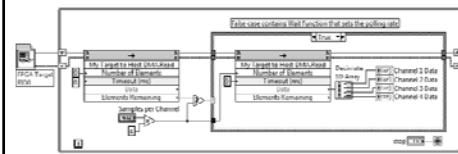


DMA FIFO Interleaving Example

FPGA VI



Host VI



Exercise 11-4: Interleaving a DMA FIFO

Create an FPGA VI that uses a single interleaved FPGA to transfer multiple channels of analog data from the FPGA target to the RT host computer

GOAL

Exercise 11-4: Interleaving a DMA FIFO

- How would you modify this application to interleave and de-interleave data from 4 accelerometer channels using the DMA FIFO?

DISCUSSION

E. Synchronizing the Host VI and FPGA VI

- Boolean Flag – use when the Host VI needs to notify the FPGA VI of an event



- Interrupts – use when the FPGA VI needs to notify the Host VI of a one-time or infrequent event

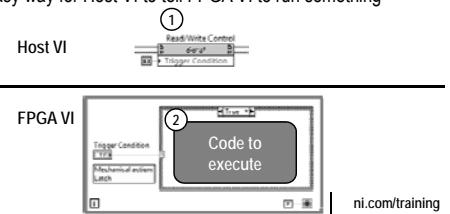


| ni.com/training

Synchronizing Host VI and FPGA VI – Boolean Flag

Use case

- Host VI notifies the FPGA VI of an event
- Easy way for Host VI to tell FPGA VI to run something



ni.com/training

Synchronizing Host VI and FPGA VI – Interrupts

Interrupt – a physical hardware line to the host that the FPGA target asserts

Use case

- FPGA VI notifies the Host VI of one-time or infrequent events
 - FPGA has started or is ready
 - Task has finished
 - Trigger occurred
 - Error occurred
- Generate interrupts from the FPGA VI
- Minimize CPU usage of host processor



NATIONAL INSTRUMENTS

ni.com/training

Interrupts – FPGA VI

Interrupt VI

- IRQ Number – specifies which logical interrupt (0 – 31) to assert. The default is 0.
- Wait Until Cleared – specifies if this VI waits until the host VI acknowledges the logical interrupt. The default is FALSE.
- Interrupt VI becomes a shared resource if you use more than one, and can induce jitter.

FPGA VI



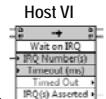
NATIONAL INSTRUMENTS

ni.com/training

Interrupts – Host VI

Host VI can wait on and acknowledge interrupts.

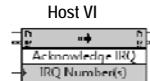
- Wait on IRQ method
 - IRQ Number(s) – specifies the logical interrupt or array of logical interrupts for which the function waits.
 - Timeout (ms) – specifies the number of milliseconds the VI waits before timing out. -1 = infinite timeout.
 - Timed Out – returns TRUE if this method has timed out.
 - IRQ(s) Asserted – returns the asserted interrupts. Empty or -1 array indicates that no interrupts were received.



ni.com/training

Interrupts – Host VI

- Acknowledge IRQ method
 - Acknowledges and resets any interrupts that have occurred to their default values
 - Use this method after the Wait on IRQ method



ni.com/training

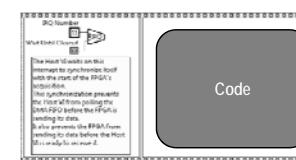
Interrupt Example – Synchronize Start of Code in Host and FPGA

Host VI



Code

FPGA VI



Code

ni.com/training

F. Implementing an FPGA Watchdog

FPGA Watchdog

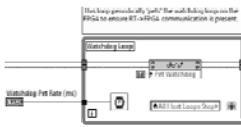
- FPGA VI monitors the status of the Host VI
- FPGA VI can take action if Host VI becomes unresponsive
 - Reboot host computer
 - Set I/O to safe states



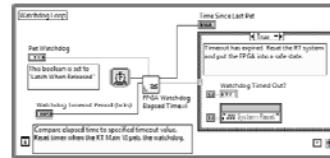
ni.com/training

Implementing an FPGA Watchdog

Host VI



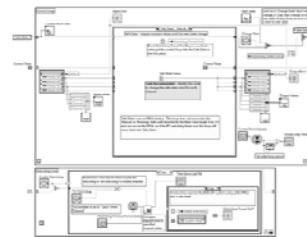
FPGA VI



ni.com/training

Implementing an FPGA Watchdog

Refer to the FPGA Control sample project in LabVIEW 2012 or later for a full-featured example.



ni.com/training

Summary—Quiz

1. Identify whether each of the following statements describes communication using front panel controls/indicators or a DMA FIFO.
 - a. Well-suited to streaming large amounts of data
 - b. Consumes significant CPU resources
 - c. Only the most recent data is transferred
 - d. Suitable for use in asynchronous applications



| ni.com/training

Summary—Quiz Answer

1. Identify whether each of the following statements describes communication using *front panel controls/indicators* or a *DMA FIFO*.
 - a. Well-suited to streaming large amounts of data - DMA FIFO
 - b. Consumes significant CPU resources - front panel controls/indicators
 - c. Only the most recent data is transferred – front panel controls/indicators
 - d. Suitable for use in asynchronous applications – DMA FIFO



| ni.com/training

Summary Quiz

2. You have developed an application that acquires data and uses a DMA FIFO to transfer data to a host VI. However, the DMA FIFO repeatedly fills, resulting in an overflow condition. Which of the following techniques can be used to address this problem? (Multiple Answers)
 - a. Reduce the speed of acquisition on the FPGA VI
 - b. Increase the number of elements to read on the host
 - c. Decrease the rate at which the host reads data
 - d. Increase the rate at which the host reads data



| ni.com/training

Summary Quiz Answer

2. You have developed an application that acquires data and uses a DMA FIFO to transfer data to a host VI. However, the DMA FIFO repeatedly fills, resulting in an overflow condition. Which of the following techniques can be used to address this problem? (Multiple Answers)
- a. Reduce the speed of acquisition on the FPGA VI
 - b. Increase the number of elements to read on the host
 - c. Decrease the rate at which the host reads data
 - d. Increase the rate at which the host reads data



ni.com/training

Summary—Quiz

3. Match the technique with its use case.

- | | |
|------------------|--|
| i. FPGA Watchdog | A. Host VI notifies the FPGA VI of an event |
| ii. Boolean flag | B. FPGA VI notifies the Host VI of one-time or infrequent events |
| iii. Interrupts | C. Monitor the status of the Host VI and take action if Host VI becomes unresponsive |



ni.com/training

Summary—Quiz Answer

3. Match the technique with its use case.

- | | |
|------------------|--|
| i. FPGA Watchdog | A. Host VI notifies the FPGA VI of an event |
| ii. Boolean flag | B. FPGA VI notifies the Host VI of one-time or infrequent events |
| iii. Interrupts | C. Monitor the status of the Host VI and take action if Host VI becomes unresponsive |



ni.com/training

Continuing Your LabVIEW Education

- Instructor Led Training
 - LabVIEW Real-Time 1
 - LabVIEW Real-Time 2
 - LabVIEW Core 2
 - LabVIEW Core 3
 - LabVIEW Performance



| ni.com/training

Continuing Your LabVIEW Education

- Training and Certification Membership upgrade
 - Includes access to all our regional and online classes plus all certifications from 1 yr of purchase
 - Please contact Customer Education at (866) 337-5918 to receive a quote for Training and Certification Membership
 - Apply the cost of this course to your membership if you purchase within 30 days



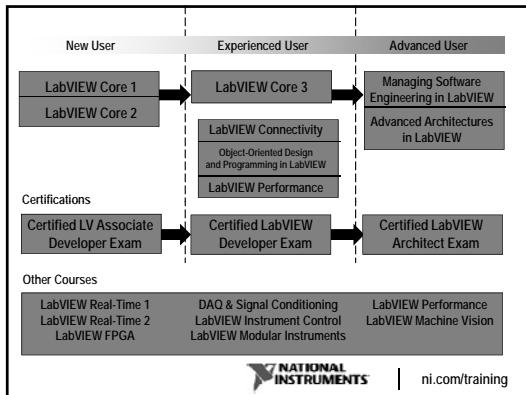
| ni.com/training

Continue Your Learning

- [ni.com/support](#)
 - On Demand training modules: [ni.com/src](#)
 - Access product manuals, KnowledgeBase, example code, tutorials, application notes, and discussion forums
- Info-LabVIEW: [www.info-labview.org](#)
- User Groups: [ni.com/usergroups](#)
- Alliance Program: [ni.com/alliance](#)
- Publications: [ni.com/reference/books/](#)
- Practice!



| ni.com/training



Additional Information and Resources

This appendix contains additional information about National Instruments technical support options and LabVIEW resources.

National Instruments Technical Support Options

Log in to your National Instruments [ni.com](#) User Profile to get personalized access to your services. Visit the following sections of [ni.com](#) for technical support and professional services:

- **Support**—Technical support at [ni.com/support](#) includes the following resources:
 - **Self-Help Technical Resources**—For answers and solutions, visit [ni.com/support](#) for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on. Registered users also receive access to the NI Discussion Forums at [ni.com/forums](#). NI Applications Engineers make sure every question submitted online receives an answer.
 - **Standard Service Program Membership**—This program entitles members to direct access to NI Applications Engineers via phone and email for one-to-one technical support, as well as exclusive access to self-paced online training modules at [ni.com/self-paced-training](#). All customers automatically receive a one-year membership in the Standard Service Program (SSP) with the purchase of most software products and bundles including NI Developer Suite. NI also offers flexible extended contract options that guarantee your SSP benefits are available without interruption for as long as you need them. Visit [ni.com/ssp](#) for more information.
- For information about other technical support options in your area, visit [ni.com/services](#) or contact your local office at [ni.com/contact](#).
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. The NI Alliance Partners joins system integrators, consultants, and hardware vendors to provide comprehensive service and expertise to customers. The program ensures qualified, specialized assistance for application and system development. To learn more, call your local NI office or visit [ni.com/alliance](#).

If you searched [ni.com](#) and could not find the answers you need, contact your local office or NI corporate headquarters. Phone numbers for our worldwide offices are listed at the front of this manual. You also can visit the Worldwide Offices section of [ni.com/niglobal](#) to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

Other National Instruments Training Courses

National Instruments offers several training courses for LabVIEW users. These courses continue the training you received here and expand it to other areas. Visit ni.com/training to purchase course materials or sign up for instructor-led, hands-on courses at locations around the world.

National Instruments Certification

Earning an NI certification acknowledges your expertise in working with NI products and technologies. The measurement and automation industry, your employer, clients, and peers recognize your NI certification credential as a symbol of the skills and knowledge you have gained through experience. Visit ni.com/training for more information about the NI certification program.