

LabVIEW™ FPGA Course Exercises

Course Software Version 2010
October 2010 Edition
Part Number 323662D-01

Copyright

© 2003–2010 National Instruments Corporation. 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.

For components used in USI (Xerces C++, ICU, HDF5, b64, Stingray, and STLport), the following copyright stipulations apply. For a listing of the conditions and disclaimers, refer to either the `USICopyrights.chm` or the *Copyrights* topic in your software.

Xerces C++. This product includes software that was developed by the Apache Software Foundation (<http://www.apache.org/>). Copyright 1999 The Apache Software Foundation. All rights reserved.

ICU. Copyright 1995–2010 International Business Machines Corporation and others. All rights reserved.

HDF5. NCSA HDF5 (Hierarchical Data Format 5) Software Library and Utilities

Copyright 1998, 1999, 2000, 2001, 2003 by the Board of Trustees of the University of Illinois. All rights reserved.

b64. Copyright © 2004–2006, Matthew Wilson and Synesis Software. All Rights Reserved.

Stingray. This software includes Stingray software developed by the Rogue Wave Software division of Quovadx, Inc. Copyright 1995–2006, Quovadx, Inc. All Rights Reserved.

STLport. Copyright 1999–2003 Boris Fomitchev

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

National Instruments Corporate Headquarters

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

Worldwide Offices

Australia 1800 300 800, Austria 43 662 457990-0, Belgium 32 (0) 2 757 0020, Brazil 55 11 3262 3599, Canada 800 433 3488, China 86 21 5050 9800, Czech Republic 420 224 235 774, Denmark 45 45 76 26 00, Finland 358 (0) 9 725 72511, France 01 57 66 24 24, Germany 49 89 7413130, India 91 80 41190000, Israel 972 3 6393737, Italy 39 02 41309277, Japan 0120-527196, Korea 82 02 3451 3400, Lebanon 961 (0) 1 33 28 28, Malaysia 1800 887710, Mexico 01 800 010 0793, Netherlands 31 (0) 348 433 466, New Zealand 0800 553 322, Norway 47 (0) 66 90 76 60, Poland 48 22 328 90 10, Portugal 351 210 311 210, Russia 7 495 783 6851, Singapore 1800 226 5886, Slovenia 386 3 425 42 00, South Africa 27 0 11 805 8197, Spain 34 91 640 0085, Sweden 46 (0) 8 587 895 00, Switzerland 41 56 2005151, Taiwan 886 02 2377 2222, Thailand 662 278 6777, Turkey 90 212 279 3031, United Kingdom 44 (0) 1635 523545

For further support information, refer to the *Additional Information and Resources* appendix. 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	v
B. Course Description	vi
C. What You Need to Get Started	vi
D. Installing the Course Software.....	vii
E. Course Goals	vii
F. Course Conventions	viii

Lesson 2

LabVIEW FPGA Basics

Exercise 2-1	Connecting and Configuring CompactRIO	2-1
Exercise 2-2	Configure the CompactRIO System	2-5
Exercise 2-3	Creating a LabVIEW FPGA Project for an R Series Device	2-10
Exercise 2-4	Creating a LabVIEW FPGA Project for CompactRIO	2-12

Lesson 3

FPGA Programming Basics

Exercise 3-1	VI Execution on the Development Computer	3-1
Exercise 3-2	VI Execution on the FPGA Target	3-5

Lesson 4

FPGA I/O

Exercise 4-1	R Series I/O	4-1
Exercise 4-2	CompactRIO I/O	4-5

Lesson 5

Timing an FPGA VI

Exercise 5-1	While Loop Timing	5-1
Exercise 5-2	While Loop Benchmarking	5-6

Lesson 6

Data Sharing on FPGA

Exercise 6-1	Accelerometer Threshold	6-1
--------------	-------------------------------	-----

Lesson 7

Single-Cycle Timed Loops

Exercise 7-1	While Loop Versus Single-Cycle Timed Loop	7-1
Exercise 7-2	Fixing SCTL Errors	7-4

Lesson 8

Basic Host Integration – PC/Real-Time

Exercise 8-1	Windows Host Integration.....	8-1
Exercise 8-2	RT Host and Windows Integration.....	8-8

Lesson 9

DMA Data Transfers

Exercise 9-1	Custom Triggering.....	9-1
Exercise 9-2	AI Interleaved DMA.....	9-13

Lesson 10

Modular Programming

Exercise 10-1	Creating an FPGA subVI.....	10-1
---------------	-----------------------------	------

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 two-day, hands-on *LabVIEW FPGA* course.

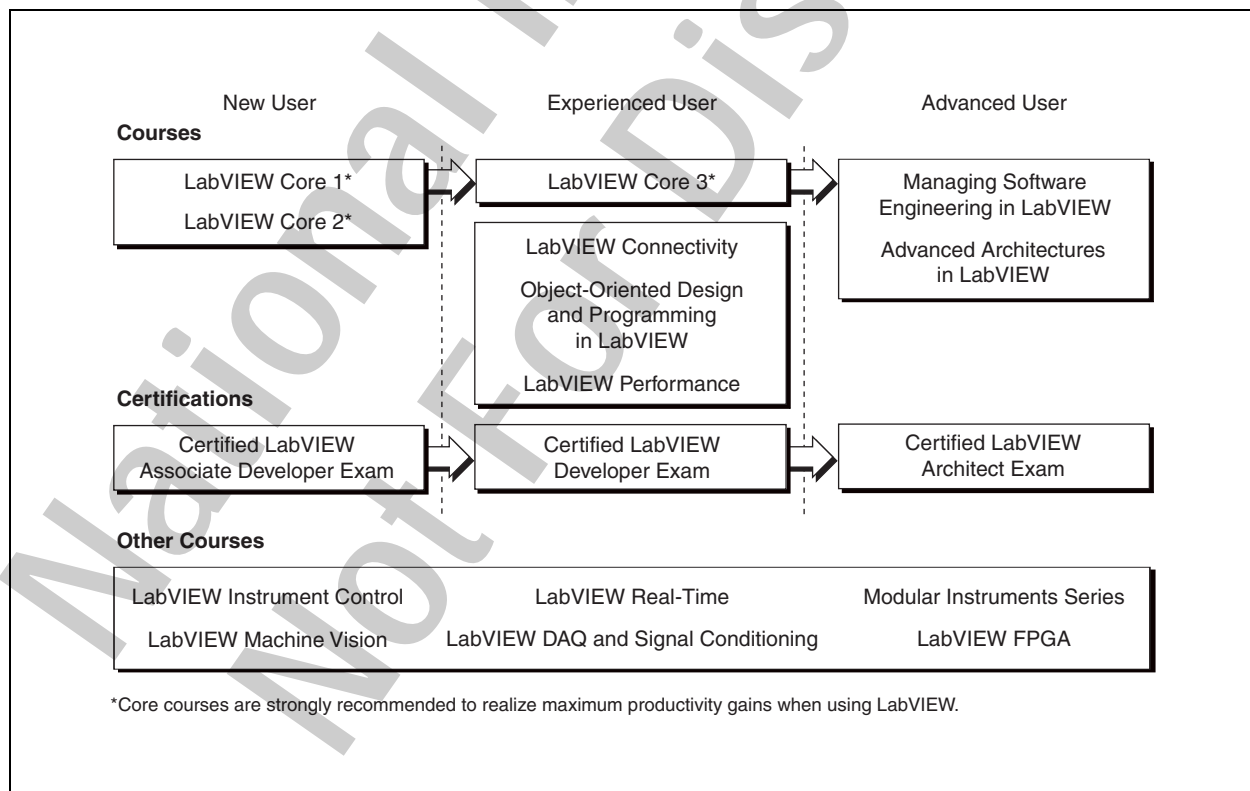
You can apply the full purchase price of this course kit toward the corresponding course registration fee if you register within 90 days of purchasing the kit. Visit ni.com/training to register for a course and to access course schedules, syllabi, and training center location information.



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 part of a series of courses designed to build your proficiency with LabVIEW and help you prepare for exams to become an NI Certified LabVIEW Developer and NI Certified LabVIEW Architect. The following illustration shows the courses that are part of the LabVIEW training series. Refer to ni.com/training for more information about NI Certification.



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 2010 or later
- ☐ Compatible versions of the FPGA Module, Real-Time Module, and NI-RIO software
- ☐ Reconfigurable I/O hardware
- ☐ *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
LabVIEW FPGA.pdf	PDF of course manual

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.
- italic* 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.

LabVIEW FPGA Basics

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 9233 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 9233 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.

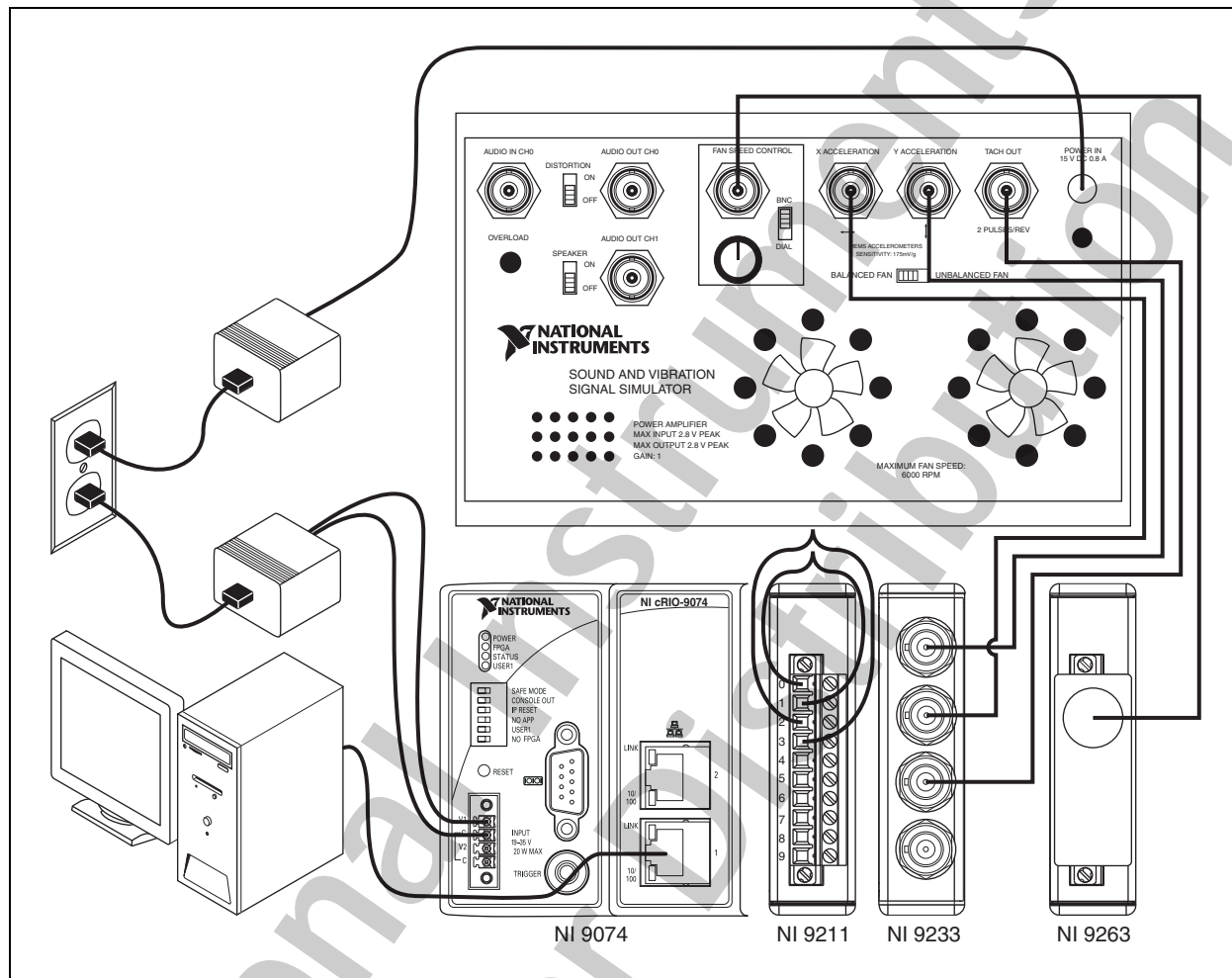


Figure 2-1. System Wiring Diagram for LabVIEW FPGA Course

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 9233

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

National Instruments
Not For Distribution

Exercise 2-2 Configure the CompactRIO System

Goal

Configure the cRIO target using Measurement and 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

IP Address

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.
3. 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.

cRIO Configuration

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 the 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**.

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.

1. Verify that the CompactRIO target is detected.
 - ☐ Expand **Remote Systems** in the configuration tree.
 - ☐ Verify that the CompactRIO target appears in Remote Systems.
 - ☐ If no device is listed, refresh the list by selecting **View»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.
 - ☐ Verify that Configure IPv4 Address is set to **DHCP or Link Local**.
 - ☐ Select the **System Settings** tab.
 - ☐ Verify that the **Halt on IP Failure** checkbox is disabled.



Note When the CompactRIO is configured to use a DHCP or link local IP address and the Halt on IP Failure checkbox is disabled, the CompactRIO uses a link local address if it does not find a DHCP server. If you are attending an instructor-led course, you connect directly to the CompactRIO to your computer using a cross-over cable, so the CompactRIO does not find a DHCP server and uses a link local IP address instead.

- ☐ Click the **Save** button unless it is dimmed.
- ☐ 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.
 - ☐ Press 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.
 - IPv4 Address: _____
 - 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.
- ☐ View the **System Settings** tab and enter cRIO-9074 in the **Name** 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 10.0»NI-RIO 3.5.0** and click **Next**, as shown in Figure 2-2.

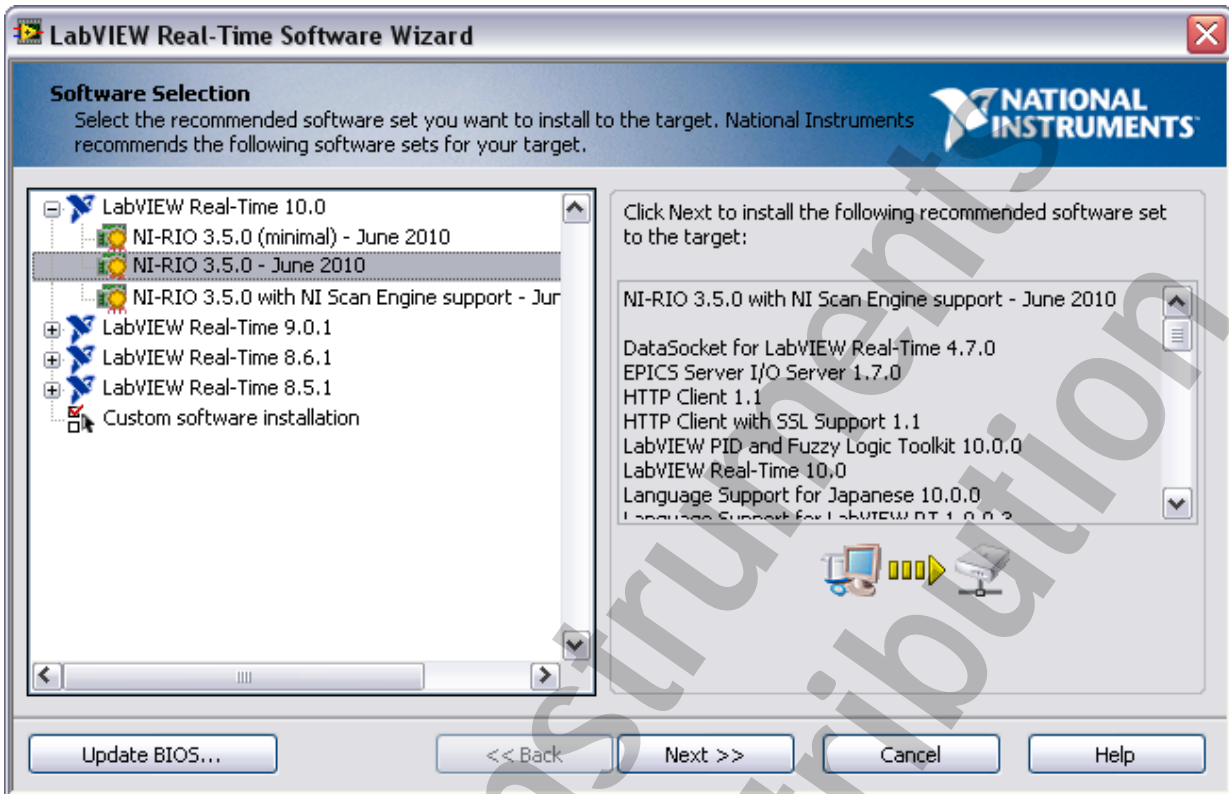


Figure 2-2. LabVIEW Real-Time Software Wizard

- ☐ Click **Next**.
 - ☐ Do not enable any add-on selections. 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 PCI-7831R. Before you begin development, you must create a LabVIEW project that contains the proper configuration for the FPGA target on the NI PCI-7831R.

Implementation

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

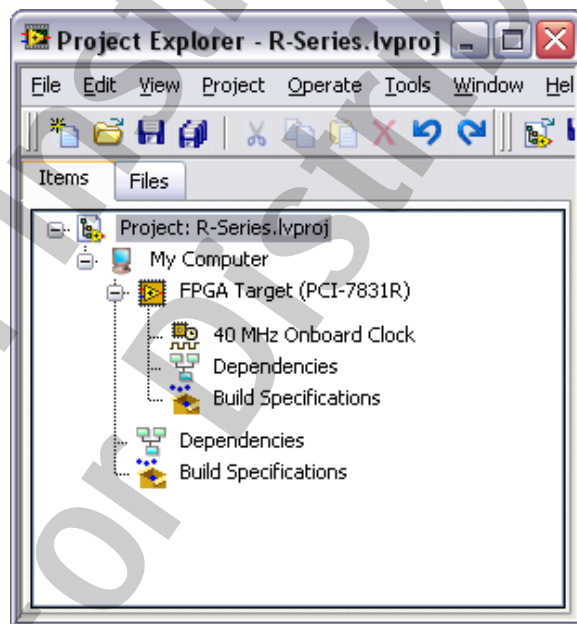


Figure 2-3. Basic R Series Under a Windows Computer

1. Launch LabVIEW and verify that the FPGA Module is installed.
 - ☐ Open LabVIEW from the desktop or select **Start»All Programs»National Instruments»LabVIEW 2010**.
 - ☐ 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.
 - ☐ Select **Empty Project** from the Getting Started window.
 - ☐ 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 PCI-7831R.
 - ☐ 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 PCI-7831R is a PCI 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 **PCI-7831R** target and click **OK**.
- ☐ In the Project Explorer window, expand the **FPGA Target (PCI-7831R)** 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.

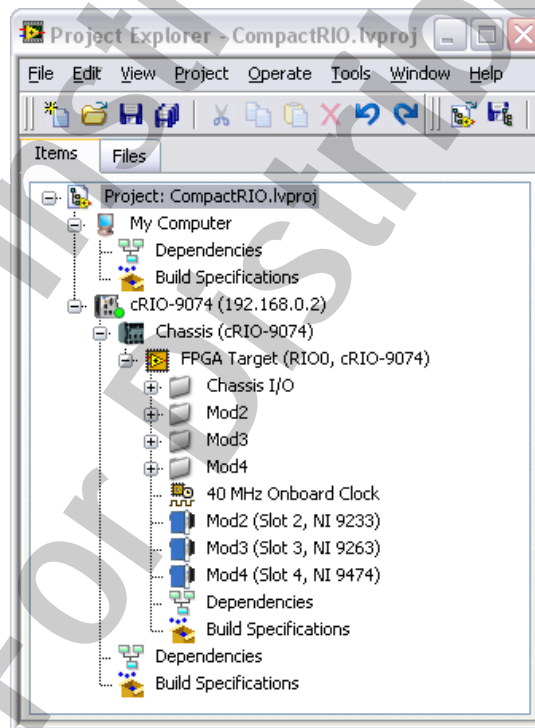


Figure 2-5. CompactRIO Project

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 both the LabVIEW Real-Time Module and the LabVIEW FPGA Module to be installed.



Figure 2-6. Real-Time Icon

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

2. Create a project.

- ☐ Select **File»New 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 Real-Time devices.

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

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 **MyComputer»FPGA Target** while the hierarchy for the CompactRIO is **cRIO-7094»Chassis»FPGA Target**.

End of Exercise 2-4

Notes

National Instruments
Not For Distribution

Notes

National Instruments
Not For Distribution

FPGA Programming Basics

Exercise 3-1 VI Execution 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 Function palettes that are available when the target of the VI is an FPGA target. 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.

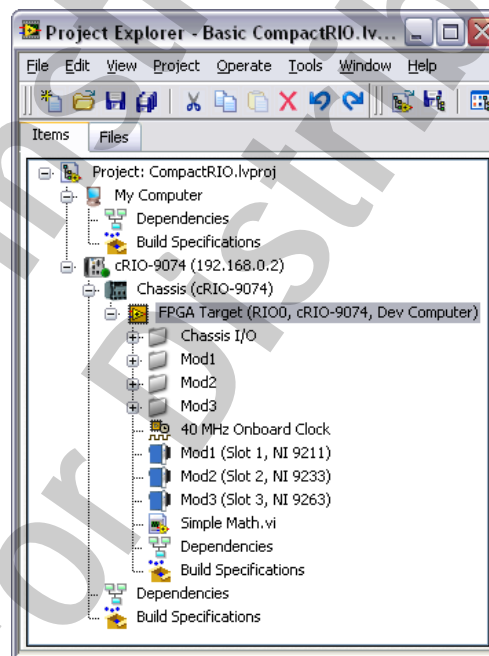


Figure 3-1. CompactRIO.lvproj with VI Executing on Development Computer

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

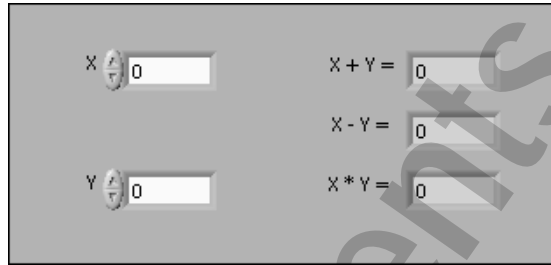


Figure 3-2. Simple Math.vi Front Panel

- ☐ Create two numeric controls, name them X and Y.
- ☐ Create three numeric indicators, name them X+Y=, X-Y=, and X*Y=.

5. Build the block diagram shown in Figure 3-3.

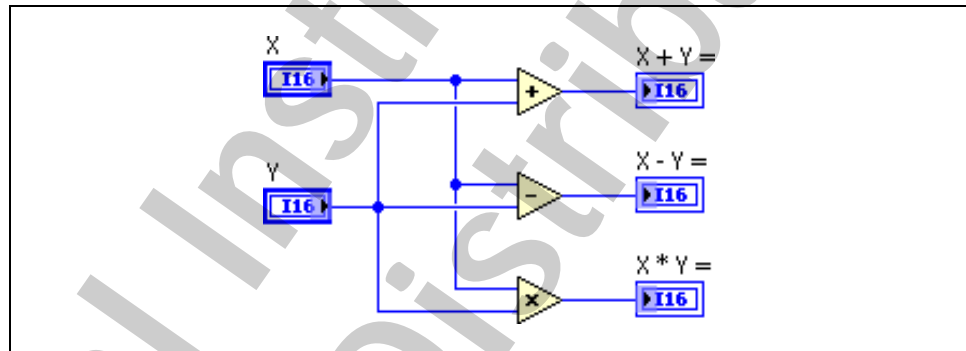


Figure 3-3. Simple Math.vi Block Diagram

- ☐ Add, Subtract, and Multiply are all located on the Numeric palette.
 - ☐ Wire the block diagram as shown in Figure 3-3.
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 VI Execution 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.

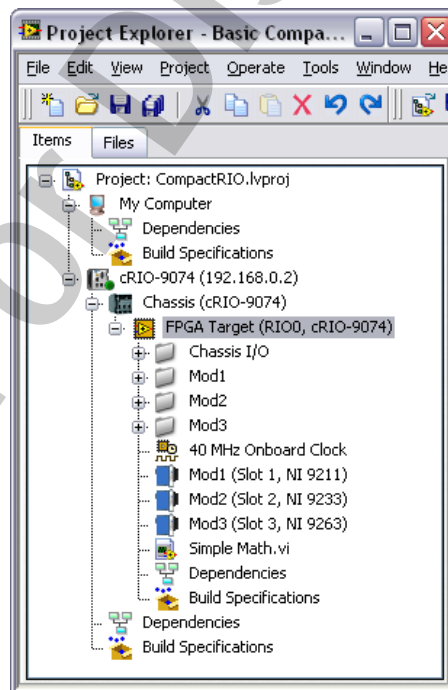


Figure 3-4. Basic CompactRIO.lvproj with VI Executing on FPGA Target

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**.

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.

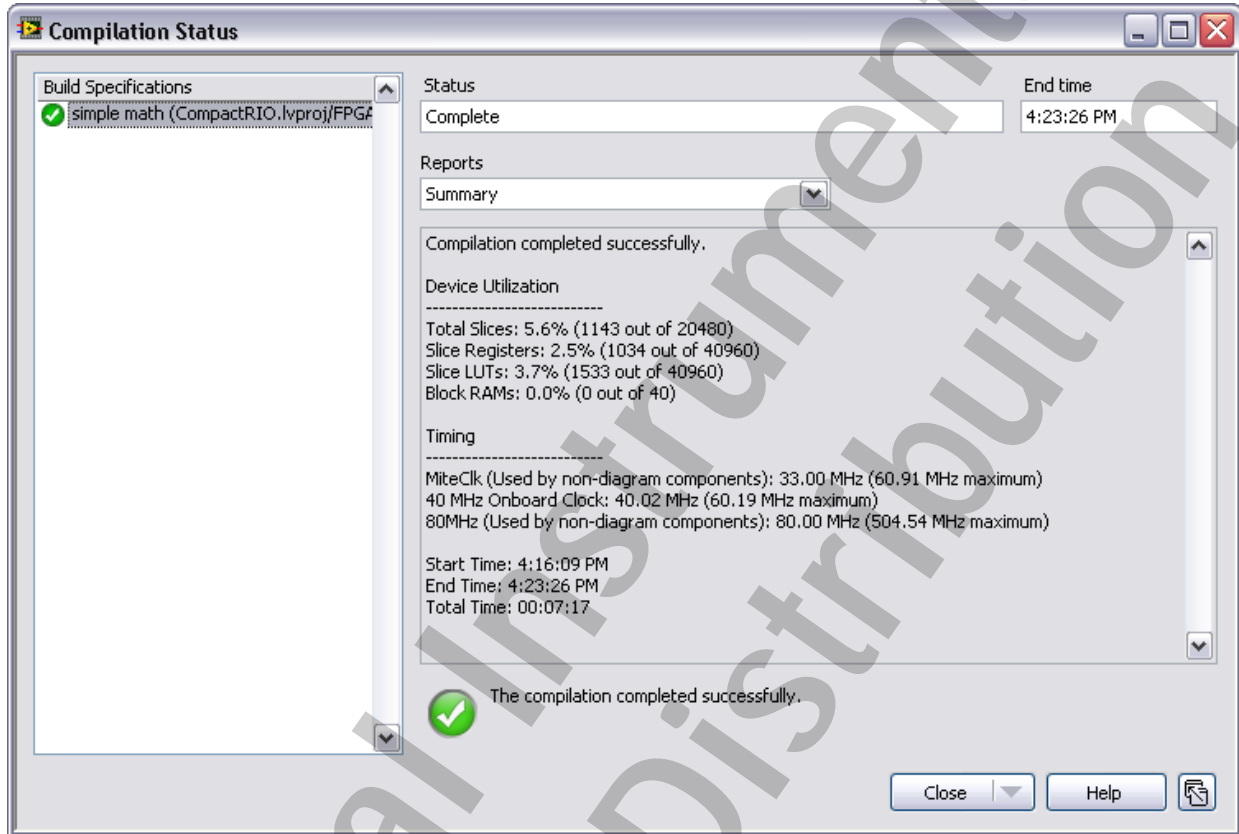


Figure 3-5. Compilation Status Window – Compilation Completed

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 the same information as the previous report, as generated during the mapping step of the compilation.
 - ☐ 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

Notes

National Instruments
Not For Distribution

Notes

National Instruments
Not For Distribution

FPGA I/O

Exercise 4-1 R Series I/O

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 PCI-7831R 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 PCI-7831R board as the FPGA Target. You created a similar project in Exercise 2-3.
 - ☐ Select **Empty 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 PCI-7831R 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.

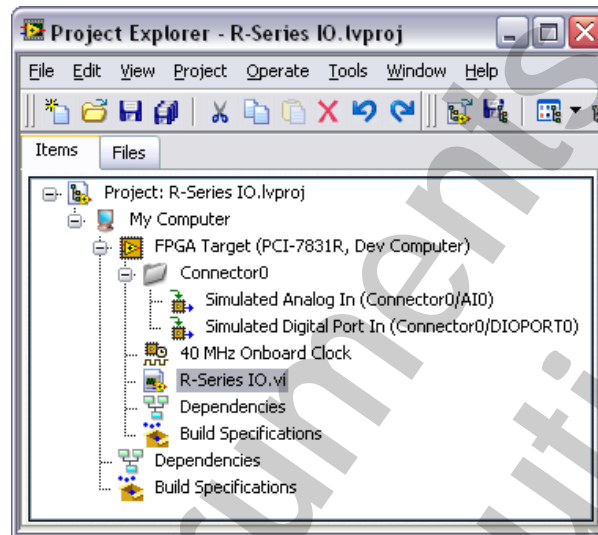


Figure 4-1. R Series Project with FPGA I/O

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

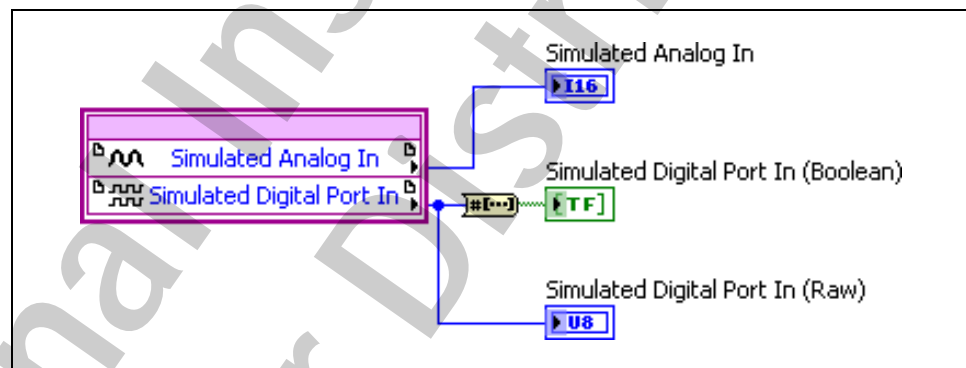


Figure 4-2. R-Series IO.vi Block Diagram

- ☐ Add the FPGA I/O node to the block diagram.
 - Drag **Simulated Analog In** from the Project Explorer window to the block diagram.
 - Expand the node to display **Simulated Digital Port In**.
- ☐ Right-click the I/O Item outputs and create indicators for each.

- ☐ Add a **Number to Boolean Array** function and create an indicator for its output. This VI converts the raw numeric that is returned by Simulated Digital Port In into a Boolean array, which better represents the state of each digital line of the port.
 - ☐ Wire the diagram and rename the indicators as shown.
8. Arrange the front panel shown in Figure 4-3.

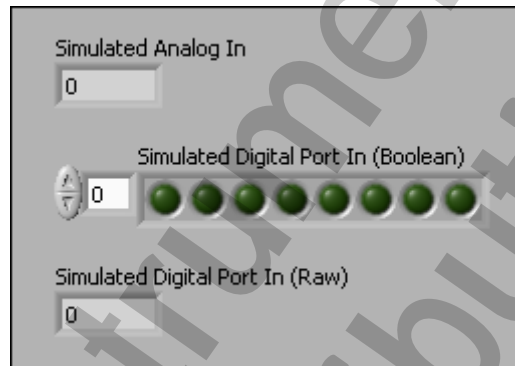


Figure 4-3. R-Series I/O.vi Front Panel

- ☐ 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 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 CompactRIO I/O

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 **Empty 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 Window should now resemble Figure 4-4.

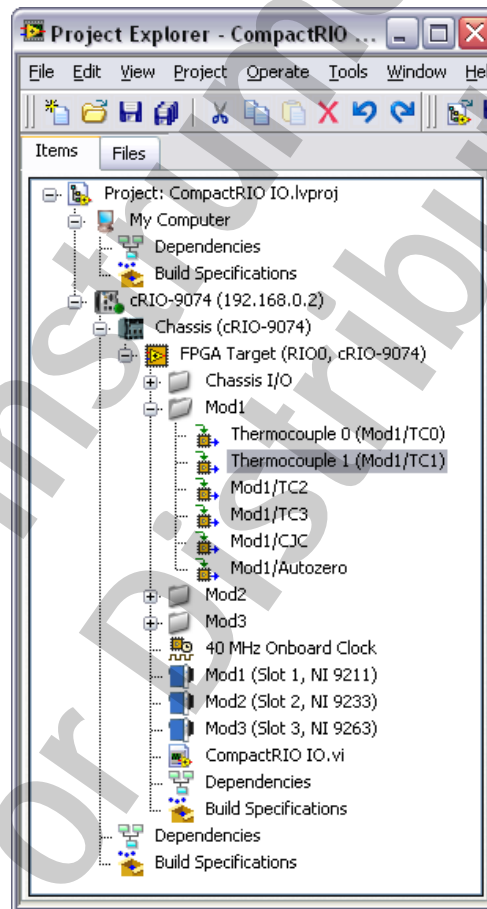


Figure 4-4. CompactRIO Project with FPGA I/O

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

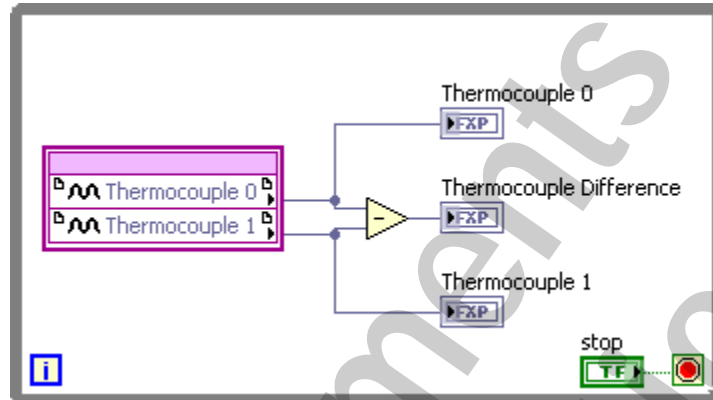


Figure 4-5. CompactRIO I/O Block Diagram

- ☐ Create a While Loop.
 - ☐ Add the FPGA I/O node to the block diagram.
 - Drag **Thermocouple 0** from the Project Explorer window to the block diagram.
 - Expand the node to display **Thermocouple 1**.
 - ☐ Right-click the I/O Item outputs and create indicators for each.
 - ☐ Add a **Subtract** function. Wire the I/O Item outputs from the FPGA I/O node to the X and Y inputs of the Subtract function.
 - ☐ Right-click and create an indicator for the Subtract function. Name this indicator **Thermocouple Difference**.
 - ☐ Create a control to stop execution of the While Loop.
 - Right-click the conditional terminal of the While Loop and select **Create Control**.
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.

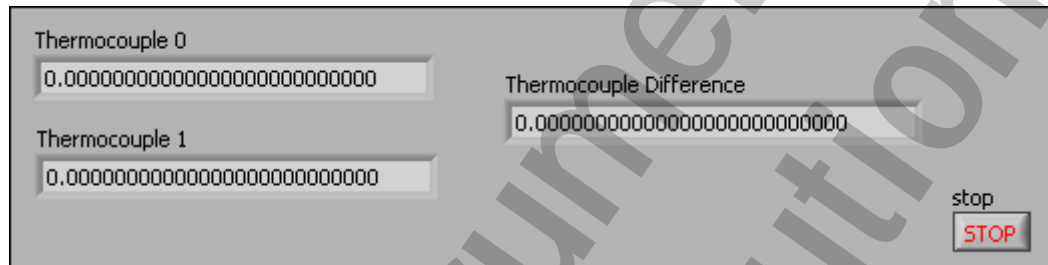


Figure 4-6. CompactRIO IO Front Panel

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 8-2.



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

Notes

National Instruments
Not For Distribution

Notes

National Instruments
Not For Distribution

Timing an FPGA VI

Exercise 5-1 While Loop Timing

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 PCI-7831R 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 PCI-7831R board as the FPGA target.
 - ☐ Select **Empty 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 PCI-7831R 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\DI00 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.

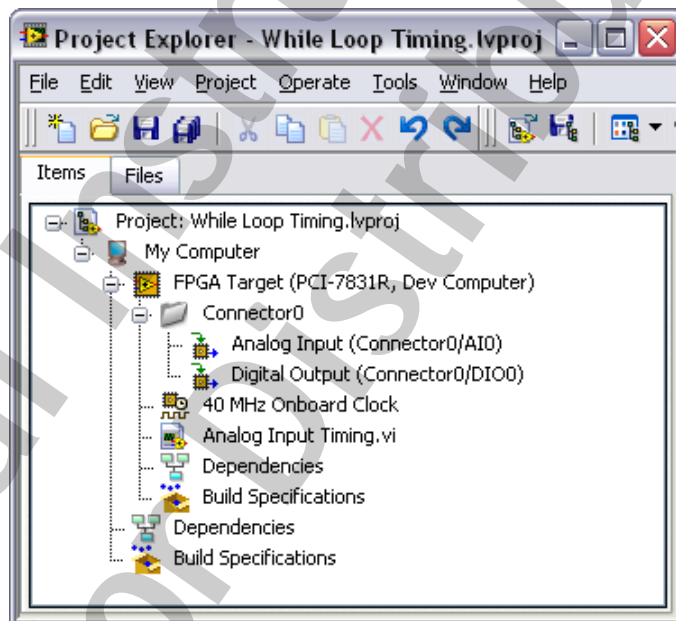


Figure 5-1. While Loop Timing Project Explorer

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

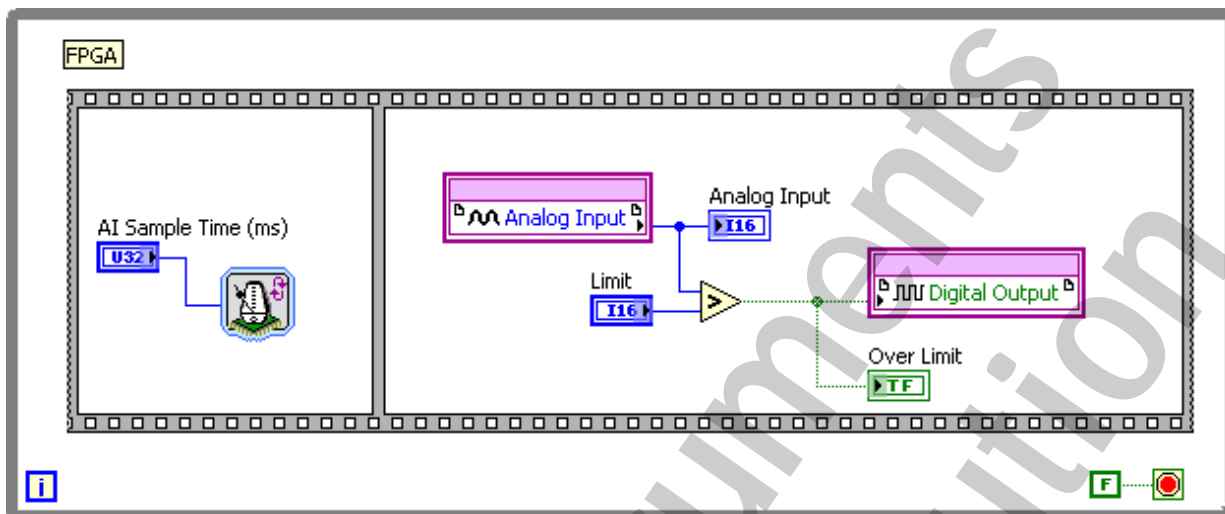


Figure 5-2. Analog Input Timing Block Diagram

- ☐ Flat Sequence structure
 - Place a Flat Sequence structure on the block diagram.
 - Right-click the Flat 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 **mSec**, as shown in Figure 5-3, and click **OK**.

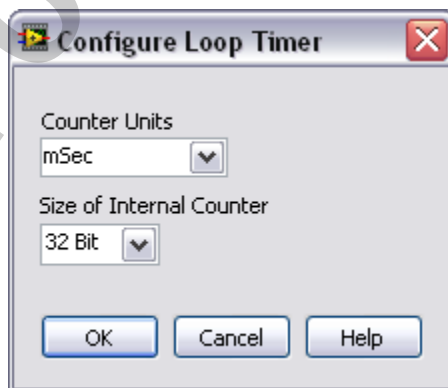


Figure 5-3. Configure Loop Timer

- ☐ AI Sample Time (ms) control
 - Right-click the **Count (mSec)** input of the Loop Timer Express VI and select **Create»Control**.
 - Rename the control AI Sample Time (ms).
- ☐ Analog Input FPGA I/O Node
 - In the Project Explorer, click the **Connector0»Analog Input** item and drag it to the block diagram.
- ☐ Analog Input indicator
 - Right-click the Analog Input output terminal of the Analog Input FPGA I/O Node and select **Create»Indicator**.
- ☐ Greater? function
- ☐ Limit control
 - Right-click the y input of the Greater? function and set the representation of this numeric control to I16.
 - Rename this control as Limit.
- ☐ Over Limit indicator
 - Right-click the output of the Greater? function and select **Create»Indicator**.
 - Rename the indicator as Over Limit.
- ☐ Digital Output FPGA I/O Node
 - In the Project Explorer, click the **Connector0»Digital Output** item and drag it to the block diagram.
 - Right-click the node and select **Change to Write**.
- ☐ While Loop
 - Add a While Loop around the Flat Sequence structure.
 - Wire a False constant to the loop conditional terminal.

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 While Loop Benchmarking

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 5-4 to benchmark the loop iteration period using the following items:

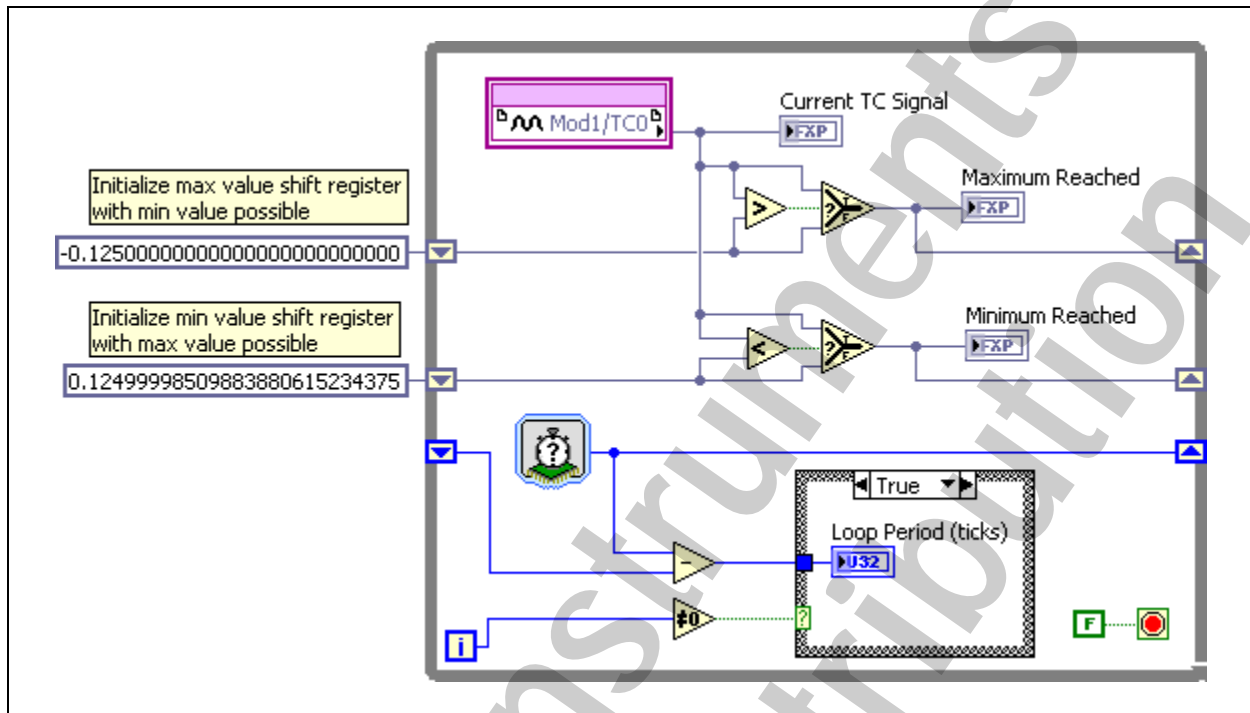


Figure 5-4. TC Max and Min Benchmark VI Block Diagram

- ☐ Shift register
 - Right-click the left border of the While Loop and select **Add Shift Register**.
- ☐ Tick Count
 - Place a Tick Count Express VI inside the While Loop.
 - In the Configure Tick Count dialog box, set Counter Units to **Ticks**, as shown in Figure 5-5, and click **OK**.

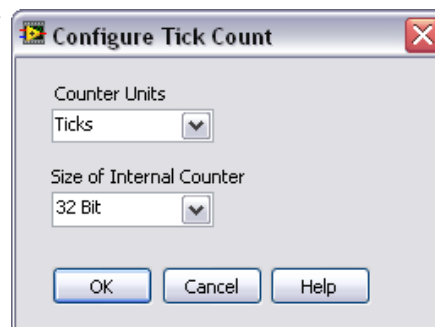


Figure 5-5. Configure Tick Count

- ☐ Case structure
 - Place a Case structure inside the While Loop.
 - Wire the output of the Not Equal To 0? function to the case selector input of the Case structure.
- ☐ Subtract function
 - Wire the output of the Subtract function to the left border of the Case structure.
- ☐ Not Equal To 0? function
- ☐ Wire the block diagram as shown in Figure 5-4.



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.

- ☐ Loop Period (ticks) indicator
 - Select the True case of the Case structure.
 - Right-click the tunnel created on the Case structure and select **Create»Indicator**.
 - Rename the indicator Loop Period (ticks).

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 **FPGA Module»CompactRIO Reference and Procedures»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

National Instruments
Not For Distribution

Notes

National Instruments
Not For Distribution

Notes

National Instruments
Not For Distribution

Data Sharing on FPGA

Exercise 6-1 Accelerometer Threshold

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 9233 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. Rename Mod2/AI0 as Y Accel.
4. 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 6-1.

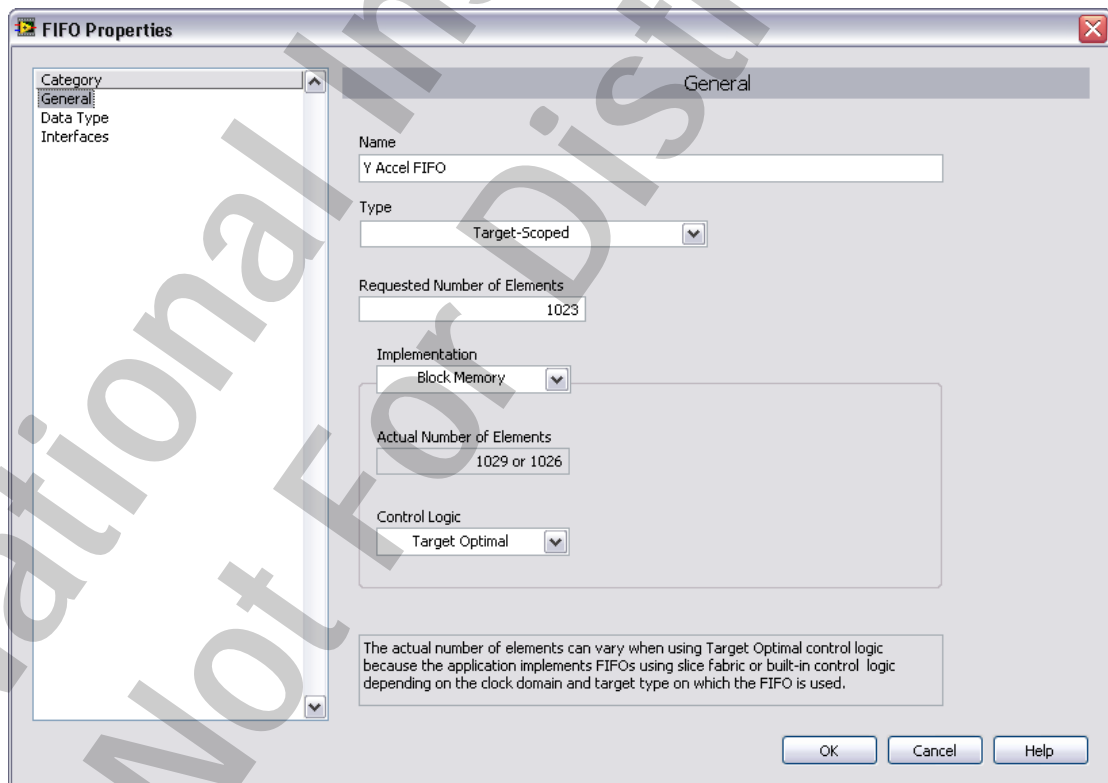


Figure 6-1. Y Accel FIFO Properties – General Category

- ❑ In the FPGA FIFO Properties dialog box, select **Data Types** from the Category list and configure the FIFO as shown in Figure 6-2.

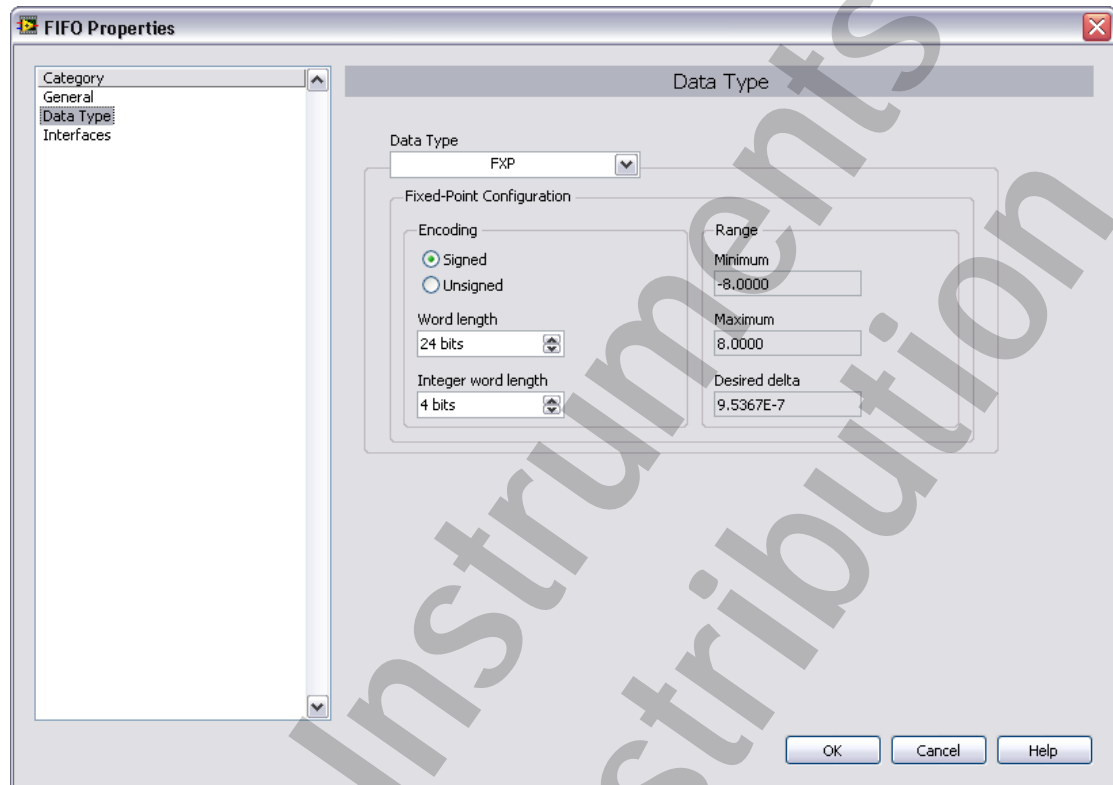


Figure 6-2. Y Accel FIFO Properties – Data Type Category



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 9233. This will help to avoid coercion of data.

- ❑ Click **OK**. The project should now resemble Figure 6-3.

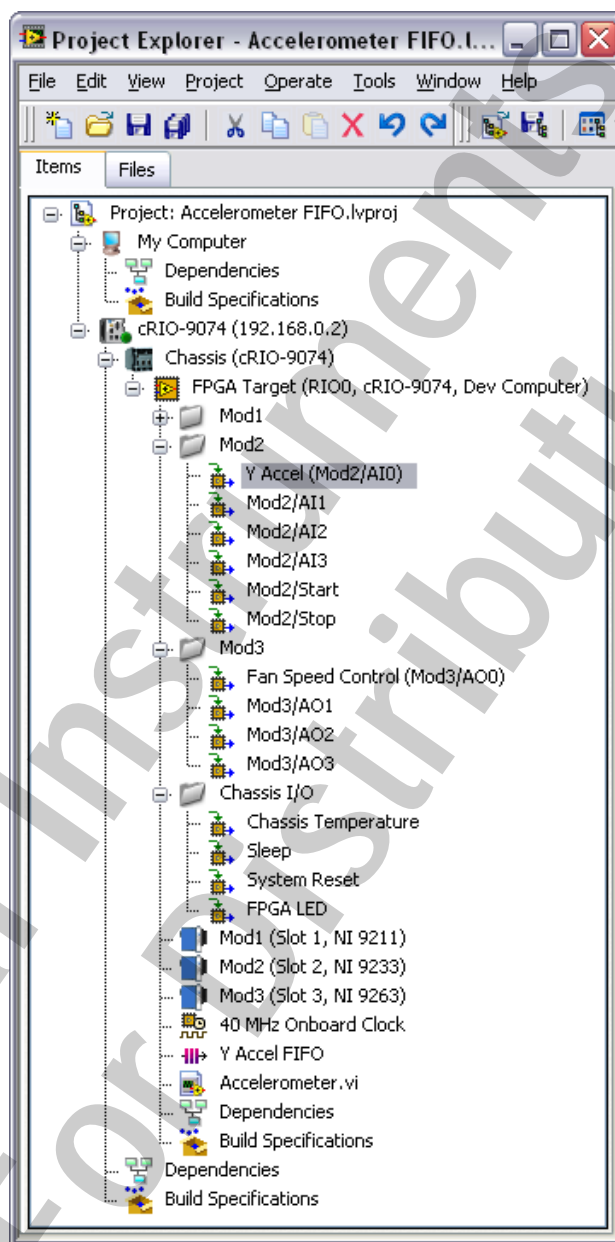


Figure 6-3. Accelerometer FIFO Project

5. Open `Accelerometer.vi` from the Project Explorer. 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 9233 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.
6. Modify the Acquisition Loop, as shown in Figure 6-4, using the following items:

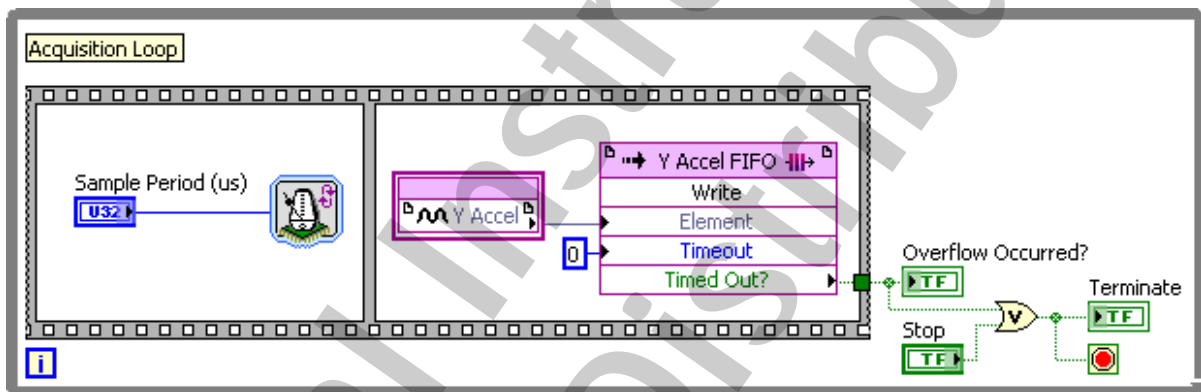


Figure 6-4. Accelerometer VI – Acquisition Loop

- ☐ FPGA I/O node—In the Project Explorer, click the **Mod2»Y Accel** item and drag it into the block diagram.
- ☐ FIFO Write node—This node allows you to write data to the FIFO that you configured in the Project Explorer.
 - In the Project Explorer, click the **Y Accel FIFO** item and drag it to the block diagram.
 - Right-click the **Timeout** input and select **Create»Constant**. Set the value of the constant to 0 so that the FIFO write times out as soon as the FIFO is full.
 - Right-click the **Timed Out?** output and select **Create»Indicator**. Rename the indicator **Overflow Occurred?**.

- ❑ OR function—This function stops the execution of the loop if the Stop button is clicked or the Y Accel FIFO times out.
 - Right-click the **OR** function output and select **Create»Indicator**. Rename the indicator **Terminate**.
- 7. 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.
- 8. Modify the Processing Loop, as shown in Figure 6-5, using the following items:

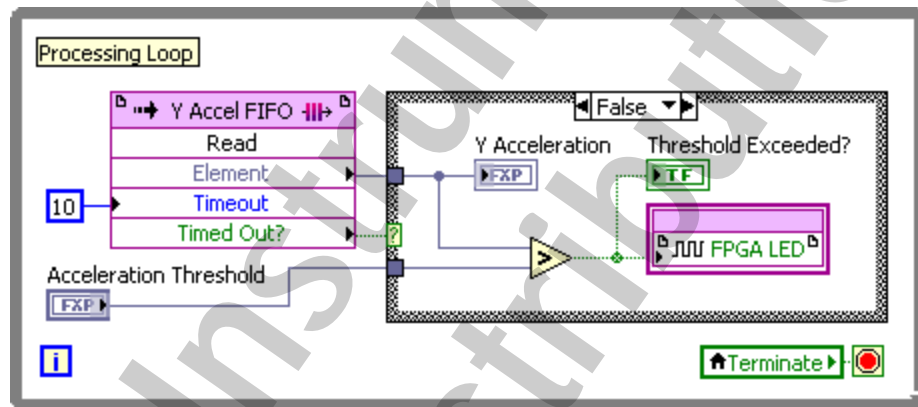


Figure 6-5. Accelerometer VI – Processing Loop

- ❑ FIFO Read node—This node reads data written by the FIFO Write node.
 - In the Project Explorer, click the **Y Accel FIFO** item and drag it to the block diagram.
 - Right-click the **Y Accel FIFO** node and select **Select Method»Read**.
 - Right-click the **Timeout** input and select **Create»Constant**. Set the value of the constant to 10 so that the FIFO read times out after ten clock ticks.
 - Right-click the **Element** output and select **Create»Indicator**. Rename the indicator **Y Acceleration**.
- ❑ 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.

- ❑ **Greater? function**—This function compares the element output of the FIFO Read node to the value of the Acceleration Threshold control.
 - Create an indicator for the output of the Greater? function and name it **Threshold Exceeded?**.
- ❑ **FPGA I/O 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, 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**.
 - Right-click the **FPGA LED** node and select **Change to Write**.
- ❑ **Local Variable for Terminate indicator**—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**.
 - Right-click the Local Variable and select **Change to Read**.
- ❑ **Wire the Processing Loop** as shown in Figure 6-5.

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

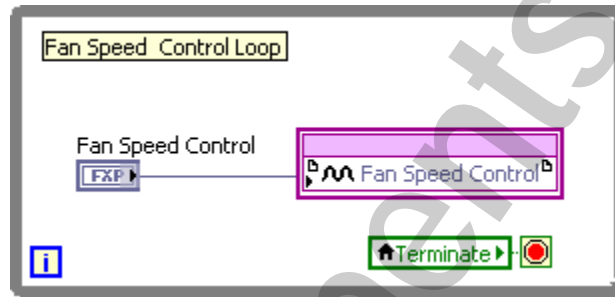


Figure 6-6. Accelerometer.vi – Fan Speed Control Loop

- ❑ 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.
 - Add a local variable to the block diagram.
 - Right-click the local variable and select **Select Item» Terminate**.
 - Right-click the Local Variable and select **Change to Read**.

10. Modify the block diagram of the Accelerometer VI, as shown in Figure 6-7, using the following items:

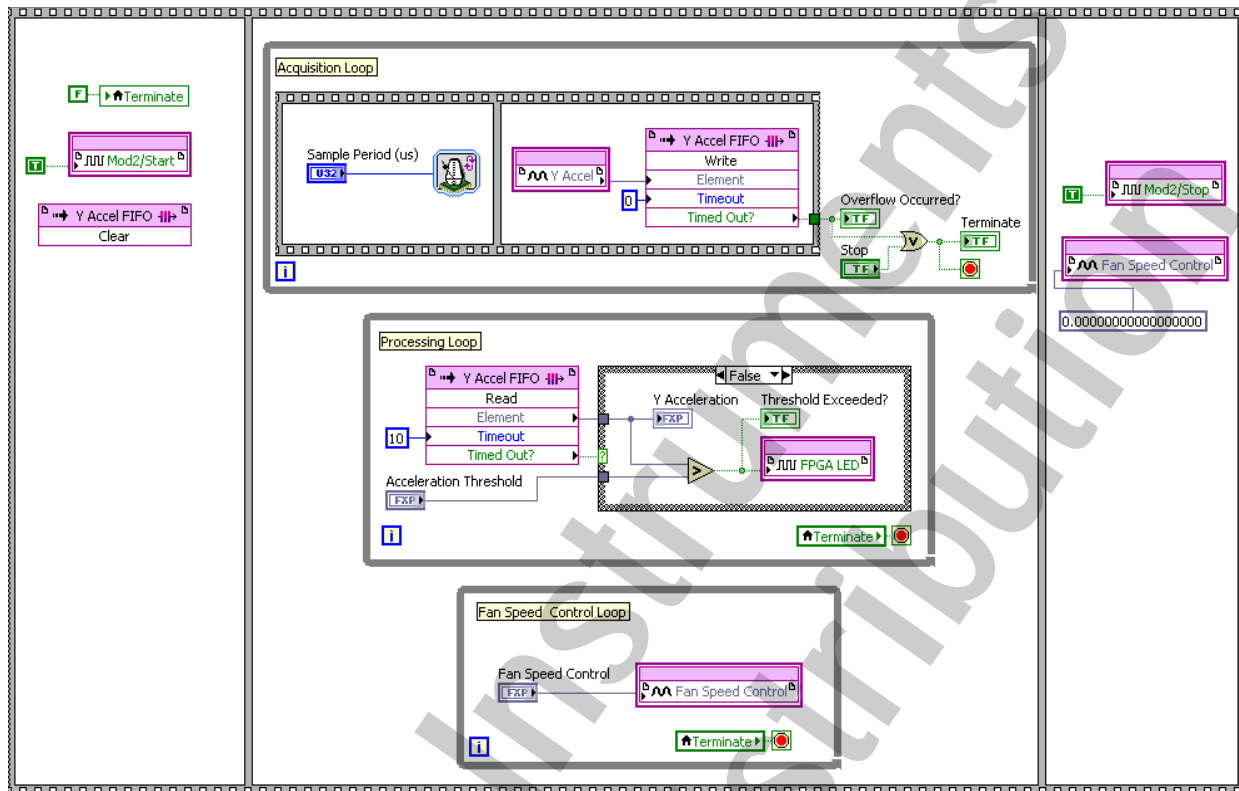


Figure 6-7. Accelerometer.vi – Block Diagram

- ❑ Local Variable for the Terminate indicator—This variable resets the Fan Control for subsequent execution of the VI.
 - Add a local variable to the block diagram.
 - Right-click the local variable and select **Select Item» Terminate**.
 - Right-click the local variable input and select **Create»Constant**. Set the constant to FALSE.
- ❑ FIFO Clear—This node clears the FIFO before executing second frame of the Sequence structure.
 - In the Project Explorer, click the **Y Accel FIFO** item and drag it to the block diagram.
 - Right-click the **Y Accel FIFO** node and select **Select Method» Control»Clear**.

- ☐ FPGA I/O node—This node turns sets the Fan Speed Control back to 0.
 - In the Project Explorer, click **Mod3»Fan Speed Control** and drag it to the block diagram.
 - Right-click the input of the **FPGA I/O** node and select **Create»Constant**. Set the constant to 0.

11. Arrange the front panel as shown in Figure 6-8.

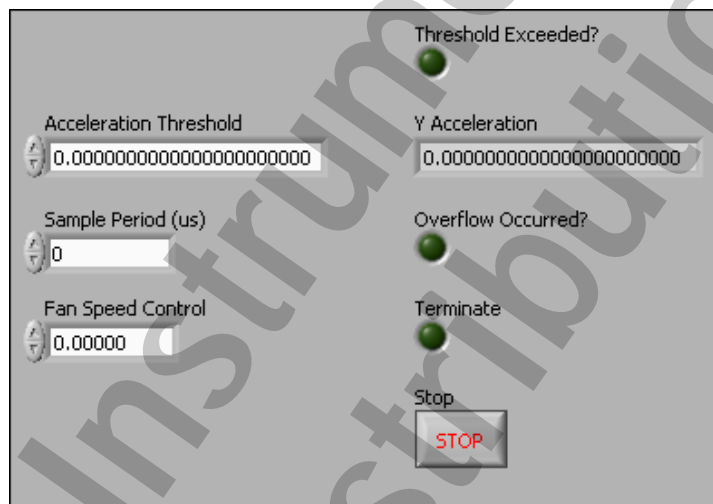


Figure 6-8. Accelerometer.vi – Front Panel

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

12. 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.

- ☐ Enter the following values for your front panel controls:
 - Acceleration Threshold: 0
 - Sample Period: 100
 - 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.
 - ☐ Change execution target to **FPGA Target**.
 - ☐ Run the VI.
 - ☐ Once compilation finishes, close the Compilation Status window.
 - ☐ Use the following values for your front panel controls:
 - Sample Period: 100
 - Fan Speed Control: 1.00000
 - Acceleration Threshold: 0
3. 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.
4. 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.
5. Close the VI and the Project.

End of Exercise 6-1

Notes

National Instruments
Not For Distribution

Single-Cycle Timed Loops

Exercise 7-1 While Loop Versus 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 amount 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 7-1, using the following item:

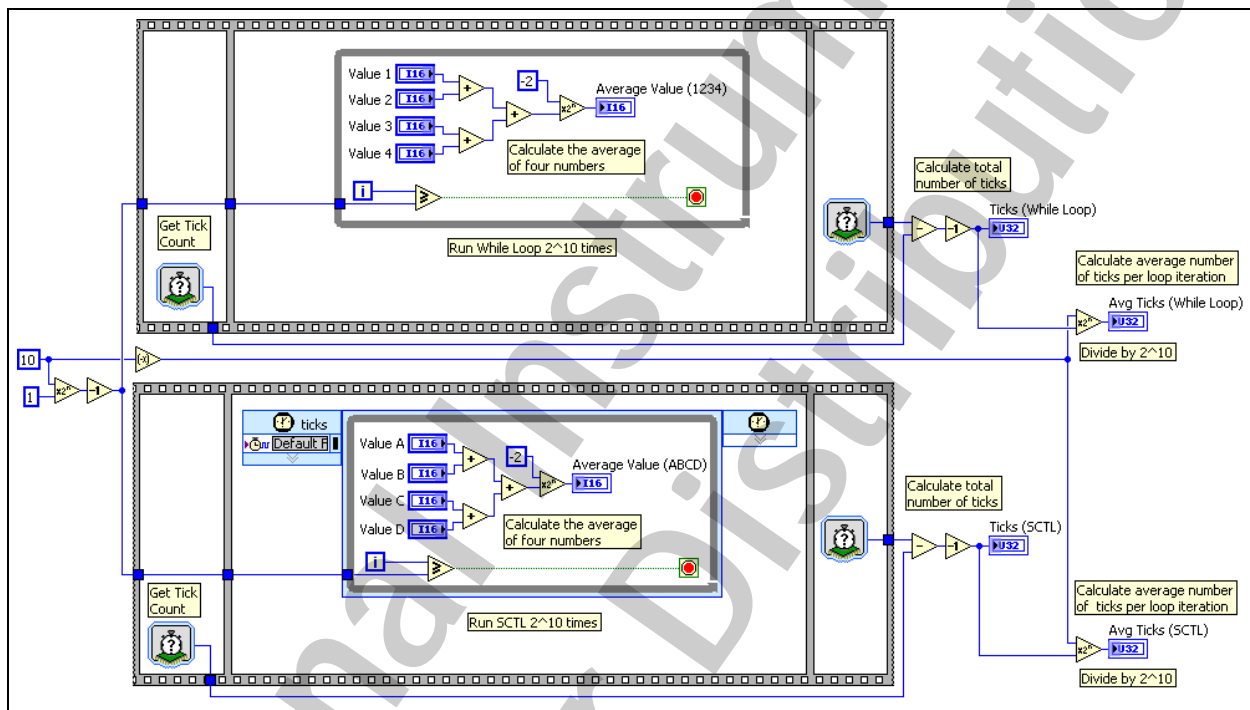


Figure 7-1. While Loop SCTL Benchmarking VI Block Diagram

- ☐ Timed Loop
 - Place a Timed Loop around the code in the second frame of the bottom sequence structure.
 - 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.

- Click **OK**.
 - Wire the loop iteration terminal and loop condition terminal as shown in Figure 7-1.
6. Save the VI.

Testing

1. Set the controls on the front panel 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 7-1

Exercise 7-2 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. From the Project Explorer, open `Investigate SCTL Errors.vi`.
4. Examine the block diagram.
 - ☐ Notice that the first frame of the Sequence structure starts the analog input acquisition on the NI 9233.
 - ☐ 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 9233 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 9233.

- 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.
- 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.
7. Modify the block diagram as shown in Figure 7-2 to remove the unsupported objects from the single-cycle Timed Loop by modifying the following items:

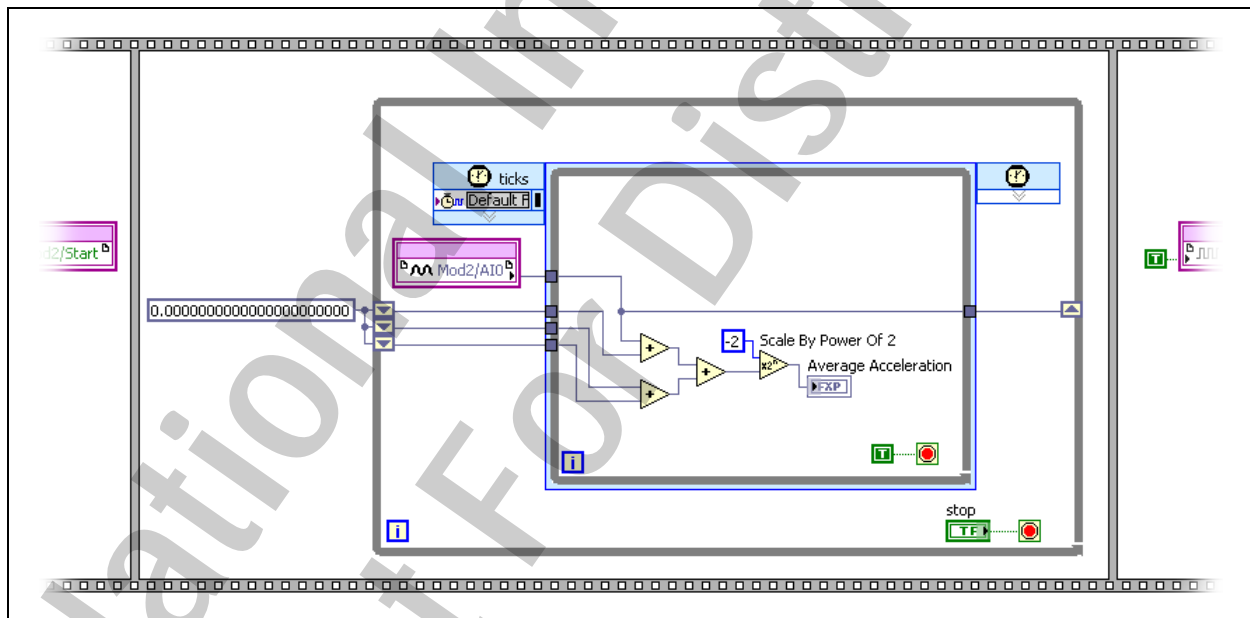


Figure 7-2. Investigate Timing Errors VI Block Diagram

- ❑ 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 7-2. Expand the Shift Register to display three elements.

8. Save the VI.

9. 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.

10. 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**.

11. Save the VI.

12. Run the VI to start the compilation process.

13. 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 4.95 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.

The difference between 5 ns and 4.95 ns is due to imperfections in the clock—finite accuracy and the presence of period jitter.

- You can see the total delay required for each path and item.

What is the total time of Path 1? _____ ns

Notice that the total delay of Path 1 exceeds the time requirement of 4.95 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 Path** to display the path on the block diagram.
- Close the Timing Violation Analysis window when finished.

14. 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**.

15. Save the VI and the project.

Testing

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 X 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 Mo2/AI0 signal.
 - ☐ When finished, click **Stop** to stop the application.
3. Close the VI and the project.

End of Exercise 7-2

Notes

National Instruments
Not For Distribution

Notes

National Instruments
Not For Distribution

Basic Host Integration – PC/Real-Time

Exercise 8-1 Windows Host Integration

Goal

Create a Windows host VI on a Windows computer that interacts with an FPGA VI on a simulated PCI-7831R board.

Scenario

You are assigned the task of creating an FPGA application that continuously acquires data from an analog input channel of the PCI-7831R 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 8-1 lists the front panel objects on the Windows host VI.

Table 8-1. Threshold Host PC 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 8-1.

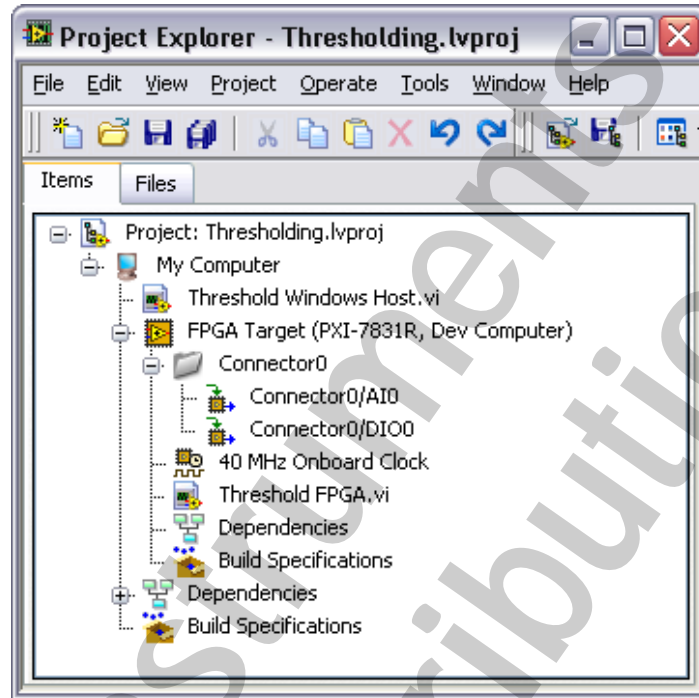


Figure 8-1. Thresholding Project Explorer

4. Create the front panel controls and indicators of the Threshold Windows Host VI as described in Table 8-1 as shown in Figure 8-2.

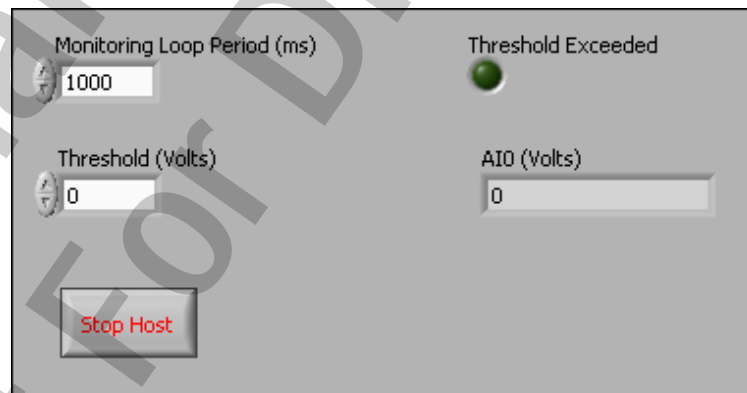


Figure 8-2. Threshold Windows Host VI Front Panel

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 8-3, using the following items:

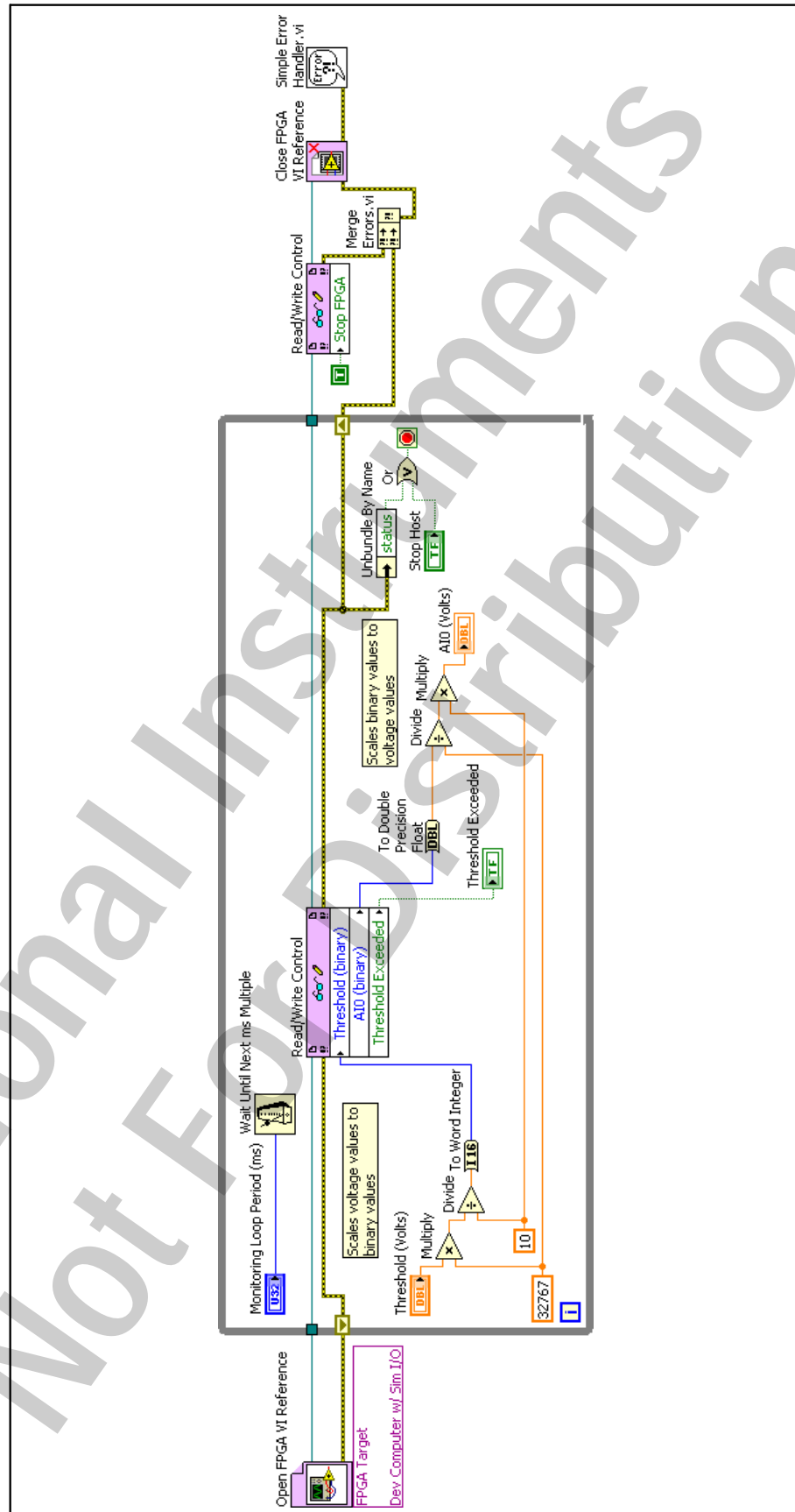


Figure 8-3. Threshold Windows Host VI Block Diagram

☐ 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 PCI-7831R have a 16-bit resolution and a voltage range from –10V to +10V. 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 8-1

Exercise 8-2 RT Host and Windows Integration

Goal

Develop an RT host VI that communicates with the FPGA and host VI running on Windows 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 hard drive. The CompactRIO starts and runs independently of the host VI running on Windows.

Design

User interface objects are listed in Table 8-2. The Temperature Monitor RT Host VI will be modified by moving the Waveform Chart to the user interface on the PC.

Table 8-2. Temperature Monitor Windows PC Host Inputs and Outputs

Type	Name	Properties
Waveform Chart	Temperature History (C)	Default = 16-bit floating-point, X-axis title = Sample, Y-axis title = Temperature (C)
Square LED	Temperature Alarm	Boolean, default = False, Off = Green, On = Red
File Path Control	Save File Path	Show Browse Button, default = C:\Exercises\LabVIEW FPGA\RT Host and Windows Integration\Temp Data.lvm
Stop button	Stop RT	Boolean, Latch when released, default = False
Stop button	Stop Windows GUI	Boolean, Latch when released, default = False

Table 8-2. Temperature Monitor Windows PC Host Inputs and Outputs (Continued)

Type	Name	Properties
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 8-1. 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, 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 8-4.

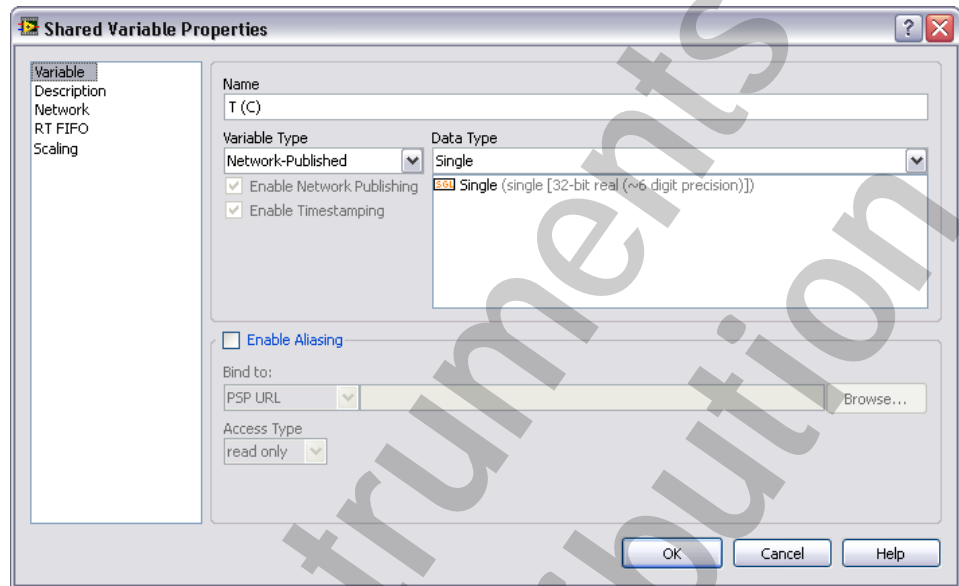


Figure 8-4. T (c) Shared Variable Properties

- ❑ Select the RT FIFO category.
- ❑ Examine the RT FIFO properties of the variable as shown in Figure 8-5.

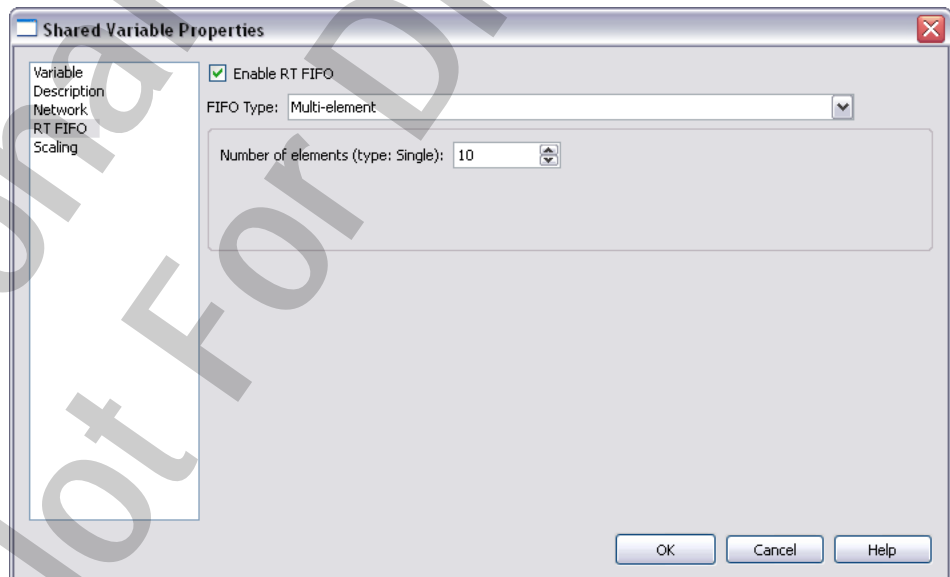


Figure 8-5. Temperature Shared Variable RT FIFO Properties

- ☐ Click **OK**.
- ☐ Refer to Table 8-3 to see the configuration of the other shared variables in the Variables library.

Table 8-3. 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 8-6:

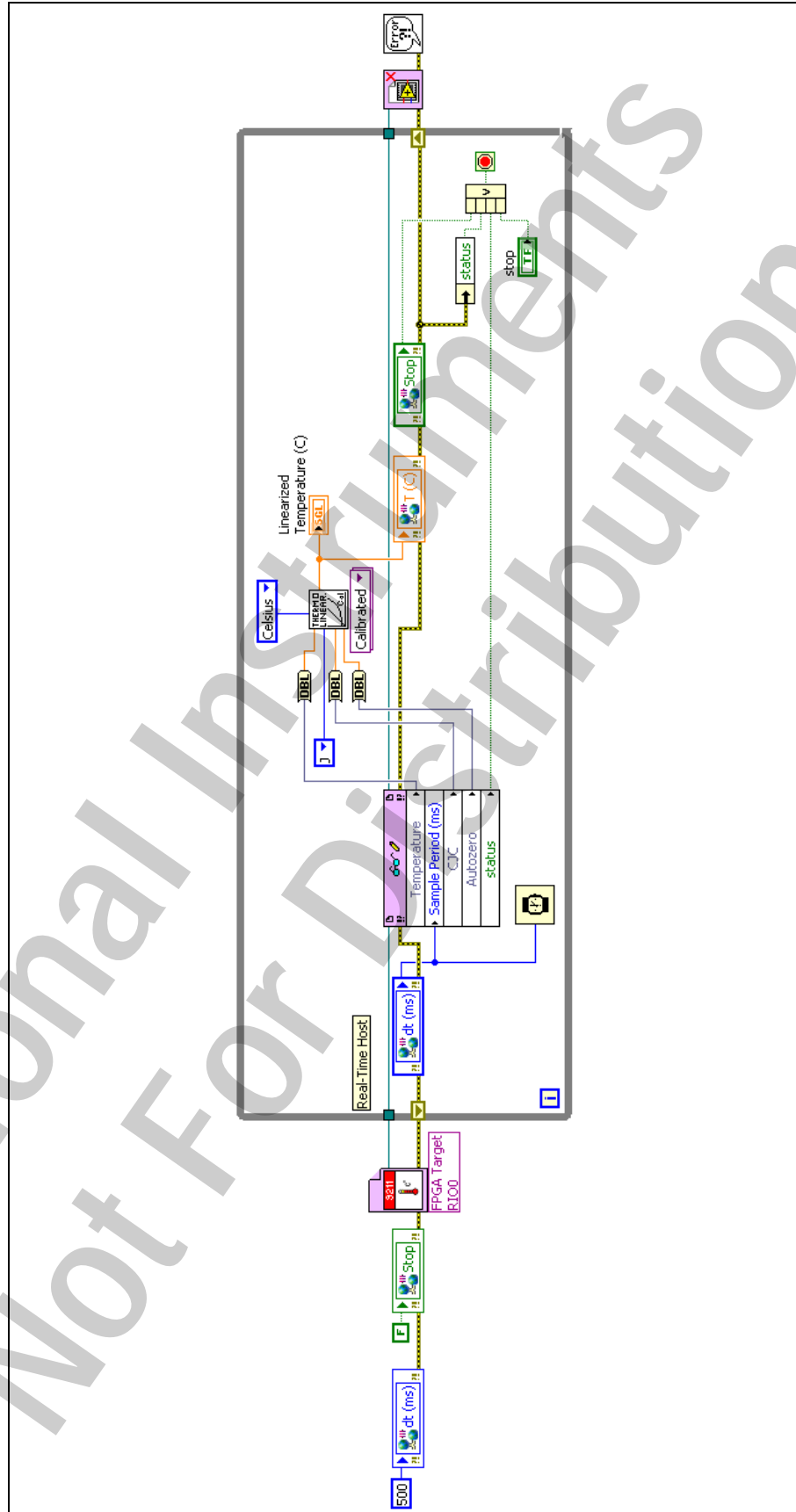


Figure 8-6. Networked Temperature Monitor RT Host VI Block Diagram

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» Change to 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» Change to 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 8-6.
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» Change to Write**.

- ☐ Place a copy of the Stop shared variable after the T (C) shared variable.
- ☐ Wire the shared variables as shown in Figure 8-6.
- 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 8-6.
- 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 Host 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 8-2.
 - ☐ Arrange the controls and indicators on the front panel as shown in Figure 8-7.

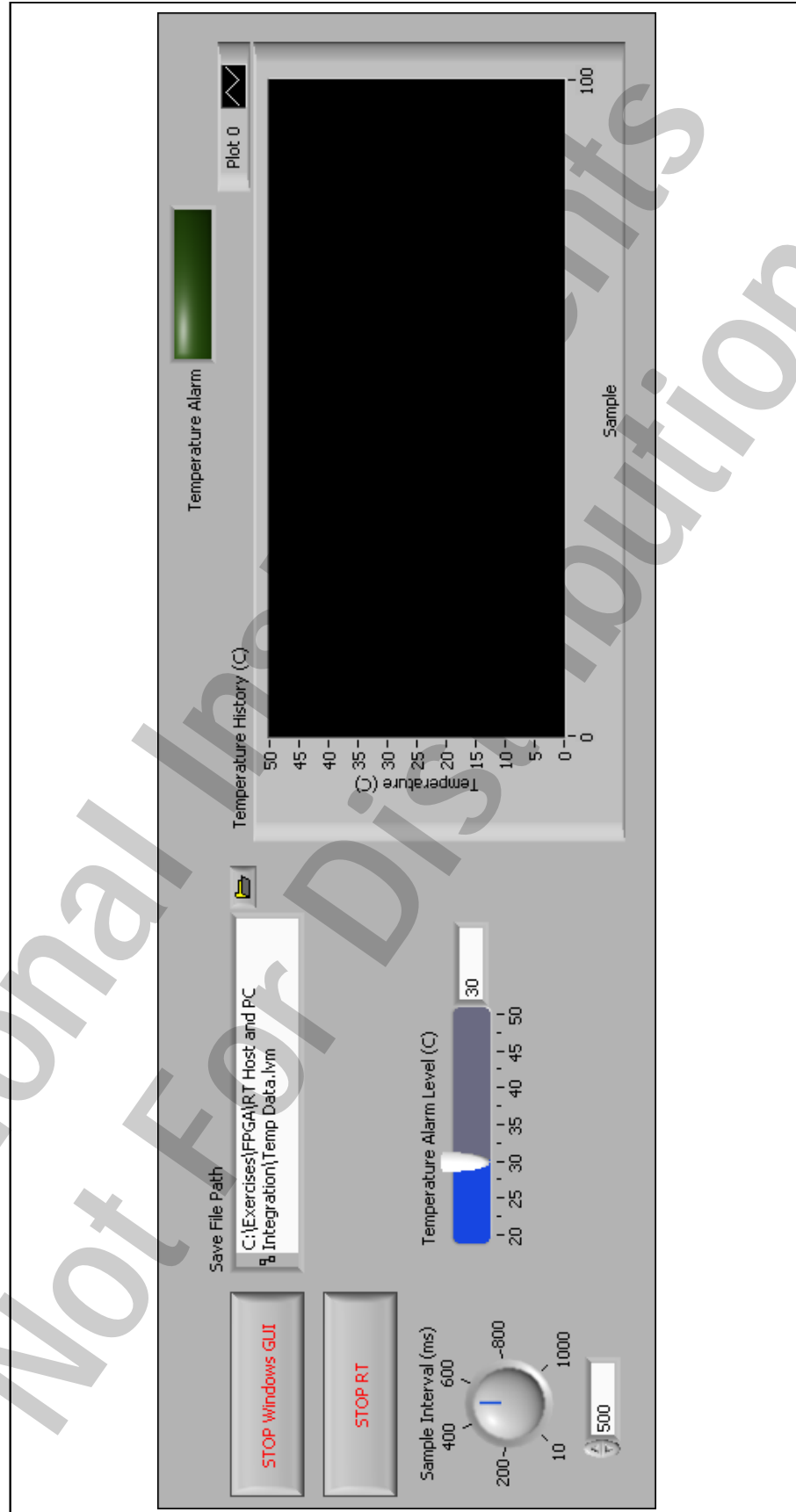


Figure 8-7. Temperature Monitor PC VI Front Panel

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

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 8-8.

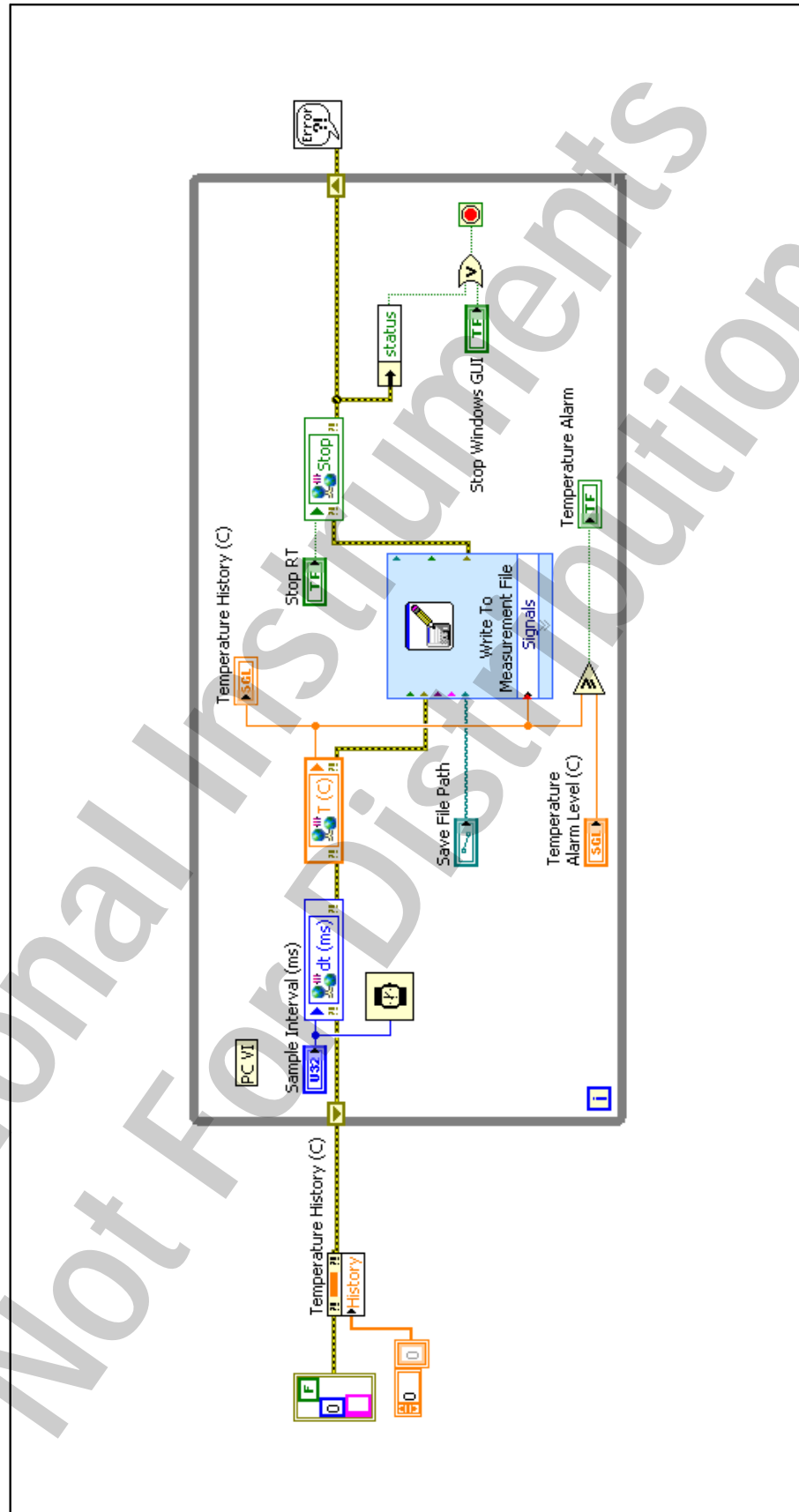


Figure 8-8. Temperature Monitor PC VI Block Diagram

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 8-9 and click **OK**.

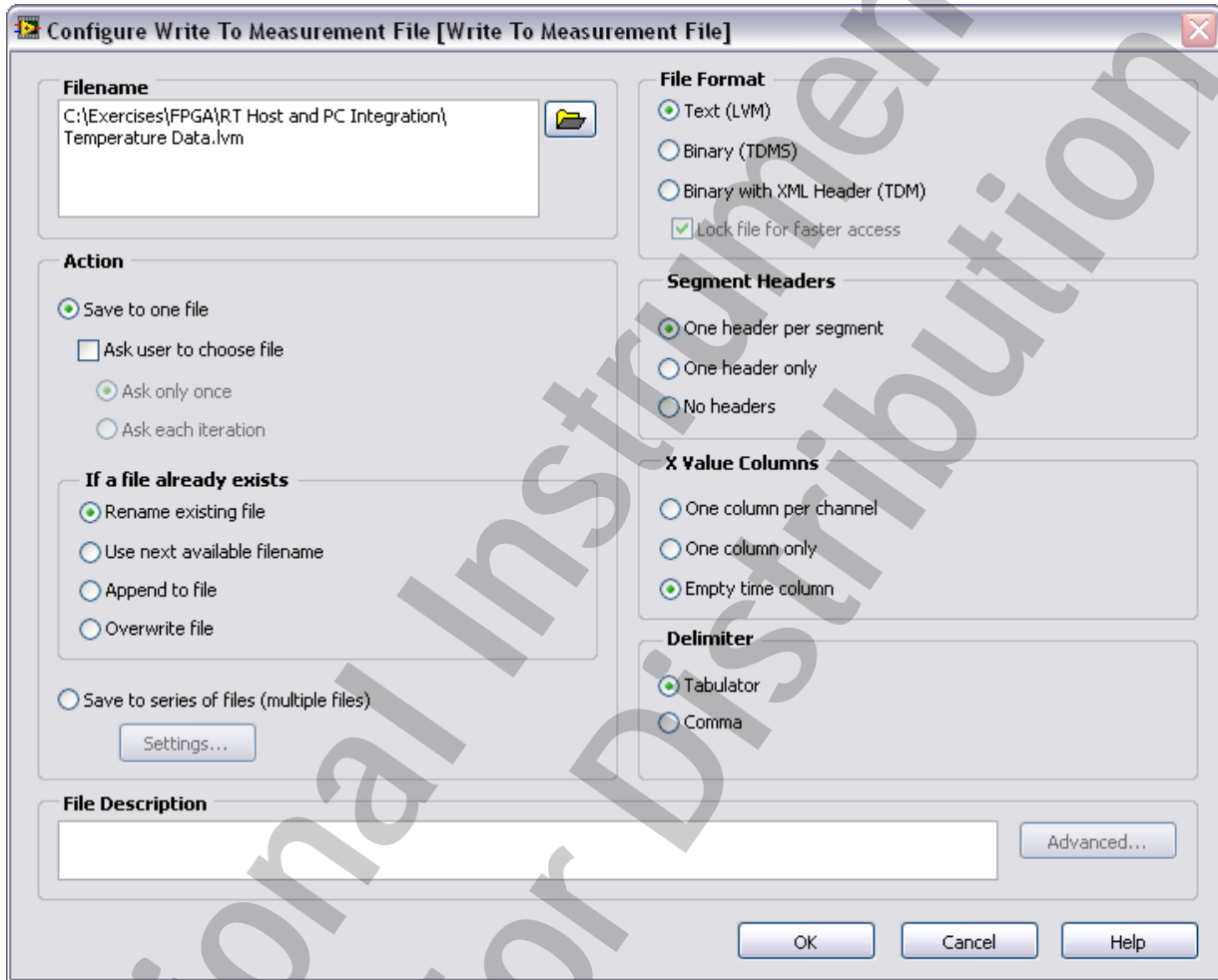


Figure 8-9. Write to Measurement File Configuration

5. Add a loop and timing.
 - ☐ Add a While Loop to the block diagram as shown in Figure 8-8.
 - ☐ 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 8-8.
- ☐ 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 8-8.
 - ☐ Right-click the History Data Property Node and select **Create»Constant**.
 - ☐ Right-click the Error In 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 8-8.
11. Save the VI and project.

Test the Windows Host

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.
 - ☐ 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 8-2

Notes

National Instruments
Not For Distribution

DMA Data Transfers

Exercise 9-1 Custom Triggering

Goal

Develop an FPGA VI that waits for multiple trigger conditions to be met before acquiring analog data and writing that data to a DMA FIFO for transfer to the RT Host.

Scenario

You are assigned the task of developing an application that acquires analog data from a PCI 7831R. The data acquisition should not begin until two criteria are met:

- The analog data is outside a specified range
- The host VI has enabled the trigger

The user defines the number of samples to acquire once both of these criteria are met. The data should be transferred to the RT host using a DMA FIFO. You must develop this application using simulated hardware since you do not have a PCI 7831R for development/testing.

Design

VI Design for Custom Trigger AI

Name the FIFO Analog Input DMA. It is a stack of 1023 I16 values and should be configured as a target-to-host DMA. The program indicates if the FIFO is full. The host VI will write a TRUE value to Trigger Enable when the user wants to enable the acquisition to begin once the analog data value is within the threshold range specified by AI Threshold Low and AI Threshold High. Once the user-defined number of samples are acquired, the FPGA VI should stop executing.

Custom Trigger Host VI Design

This VI acts as the user interface for the Custom Trigger AI VI. It passes the user-defined values to the FPGA VI and presents data that is acquired in a waveform graph.

Implementation

1. Create Custom Trigger Analog Input.lvproj in the <Exercises>\LabVIEW FPGA\Custom Trigger Analog Input directory.
 - ☐ Select the PCI-7831R as your FPGA Target.
 - In the Project Explorer, right-click the FPGA target and select **Execute VI on»Development Computer with Simulated I/O**.
 - Add Connector0\AI0 to the project as your FPGA I/O.
 - Rename Connector0\AI0 as Analog Input.
2. Create Analog Input DMA to act as your DMA channel for communicating 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 Figures 9-1.

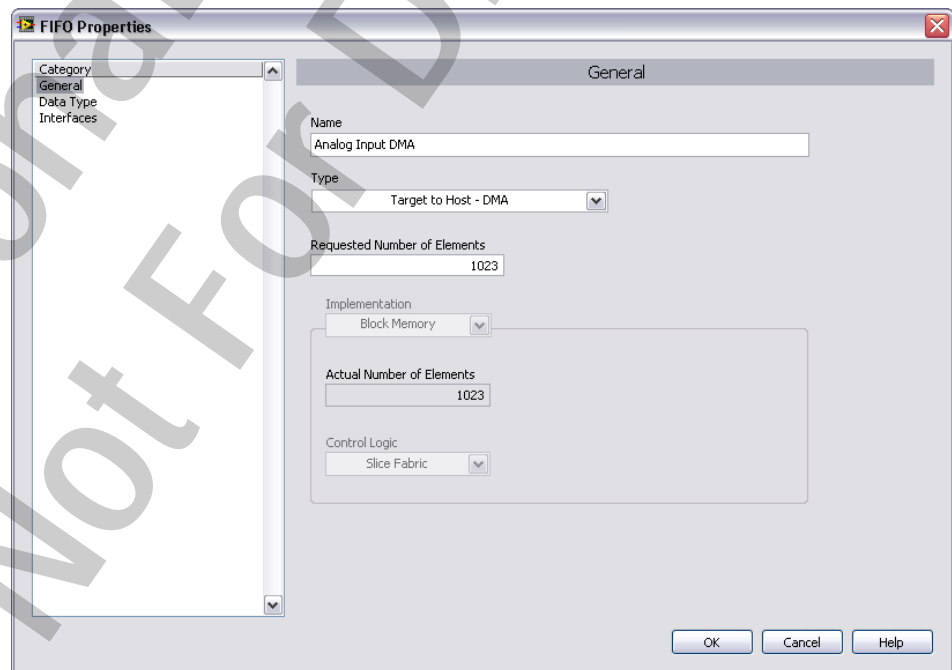


Figure 9-1. Analog Input DMA Properties – General

- ☐ Select the **Data Type** category and configure settings as shown in Figure 9-2.

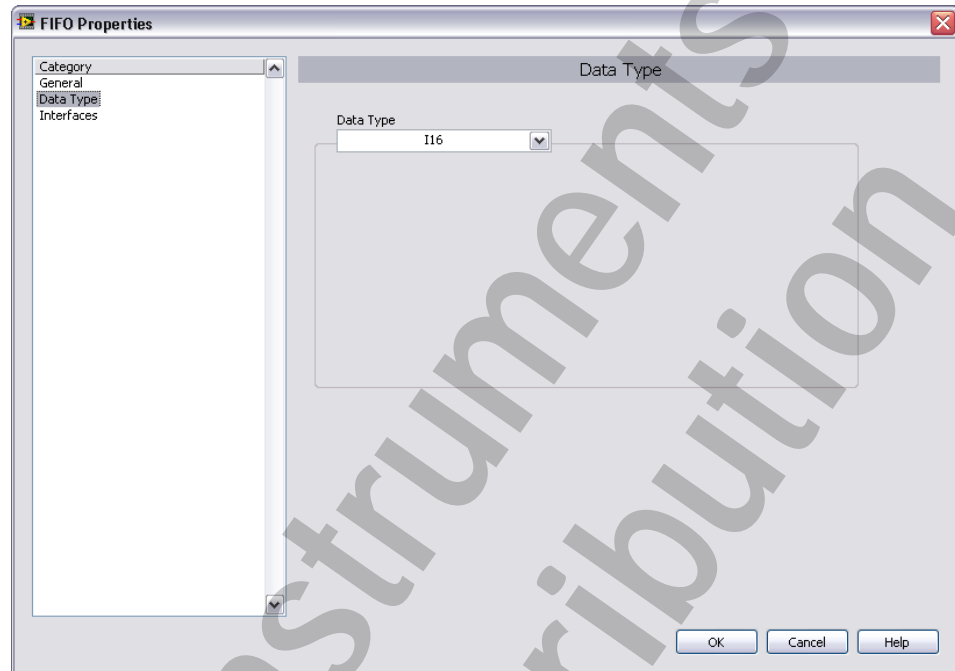


Figure 9-2. Analog Input DMA Properties – Data Type

3. Create Custom Trigger AI.vi on the FPGA target.
 - ☐ Create a new VI on the FPGA target.
 - ☐ Save the VI as Custom Trigger AI.vi.
 - ☐ Close the VI.
4. Create the Custom Trigger Windows Host VI on the PC host.
 - ☐ Create a new VI under My Computer.
 - ☐ Save the VI as Custom Trigger Windows Host.vi.
 - ☐ Close the VI.

- Verify that your Project Explorer window now resembles Figure 9-3.

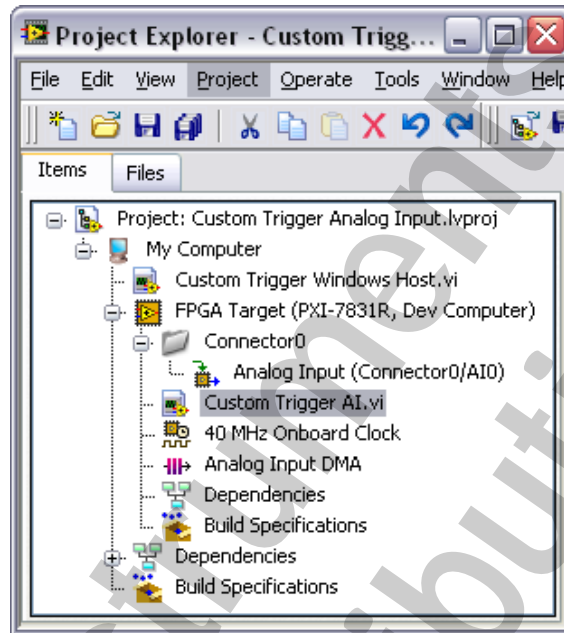


Figure 9-3. Custom Trigger Analog Input Project

5. Build the Custom Trigger AI VI, as shown in Figure 9-4. This VI waits for the trigger to be enabled and for the value of Analog Input to exceed the high or low user-defined threshold values. When these conditions are met, a user-defined number of points is acquired and written to Analog Input DMA. Use the following items to build the block diagram:

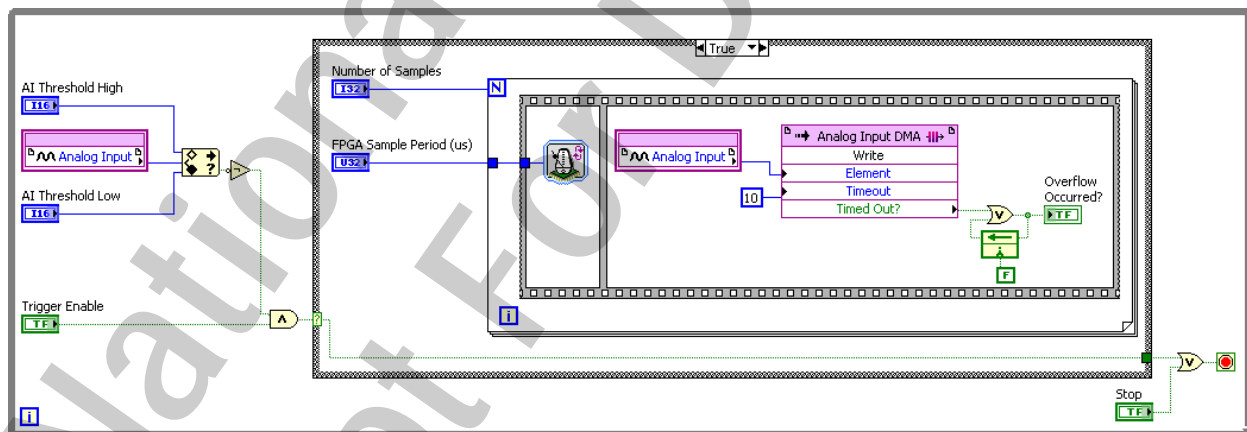


Figure 9-4. Custom Trigger AI.vi Block Diagram (True Case)

- While Loop

- ❑ **FPGA I/O Node**—This node acts as a trigger for the analog acquisition. The second node acquires data that will be written to the DMA channel when the trigger conditions are met.
 - Configure this node to acquire data from Analog Input on Connector0.
- ❑ **In Range and Coerce**—This VI is located on the Comparison palette. It determines whether or not the input value is within a specified range of values. When input x is between the two limit values, then In Range? returns TRUE.
 - Right-click the upper limit input to this function and select **Create»Control**. Name this control AI Threshold High.
 - Right-click on the lower limit input of this function and select **Create»Control**. Name this control AI Threshold Low.
- ❑ **Not function**—This function inverts the In Range? Output of the In Range and Coerce function. The output of this function will be TRUE when the x input of In Range and Coerce is outside of the limits specified for that function.
- ❑ **And function**—The output of this function is TRUE only when both of its inputs are TRUE.
 - Right-click the y input of this function and select **Create»Control**. Name this control Trigger Enable.

When Trigger Enable is TRUE and the value of Analog Input is outside of the range given by AI Threshold High and AI Threshold Low, the result of this function will be TRUE. Otherwise it will be FALSE.

- ❑ **Case Structure**—When the trigger condition has been met and the trigger has been enabled, the True case of this structure will execute.
 - Right-click the count terminal of the For Loop and select **Create»Control**. Name this control Number of Samples.



Note The data point that acts as the trigger for the acquisition is not written to the FIFO or transferred to the host VI.

- ❑ Flat Sequence Structure—Place this structure inside of the For Loop.
 - Right-click on the left edge of the Flat Sequence Structure and select **Add Frame Before**.
- ❑ Loop Timer—Place this VI in the first frame of the Flat Sequence Structure. This VI determines the sample period of the acquisition.
 - Set the Counter Units to **uSec** and the Size of Internal Counter to **32 Bit**. Click **OK**.
 - Right-click the Count(uSec) input of the Loop Timer and select **Create»Control**. Name this control `FPGA Sample Period (us)`. Move this control outside the For Loop and reconnect it to the Loop Timer.
- ❑ FPGA I/O Node—Place this node in the second frame of the Flat Sequence Structure. This node acquires analog data that is written to the DMA FIFO.
 - Configure this node to acquire data from Analog Input on Connector0.
- ❑ FIFO Write—Navigate to the Memory and FIFO Palette and select FIFO Method Node. Place this node into the second frame of the sequence structure
 - Right-click on the FIFO Write and select **Select FIFO»Analog Input DMA**.
 - Right-click the Timeout input of the FIFO Write and select **Create»Constant**. Set the value of this constant to 10.
- ❑ Or function—Use this function to latch the result of the Timed Out? output of the FIFO Write node. If it has ever timed out, then an overflow has occurred.
 - Wire the Timed Out? output of the FIFO Write node to the top input of this function.
 - Wire the result output to the second input to create a feedback node. Initialize the feedback node as FALSE.
 - Right-click the result output of this function and select **Create»Indicator**. Name the indicator `Overflow Occurred?`.

- ❑ Or function—This function stops execution of the While Loop if the user stops it manually or if the trigger is enabled, the trigger conditions are met, and the acquisition has completed.
 - Right-click the y input of this function and select **Create» Control**. Name the control `Stop`.
 - ❑ Wire the diagram as shown in Figure 9-4.
6. Select the False case of the Case Structure. Wire the diagram as shown in Figure 9-5.

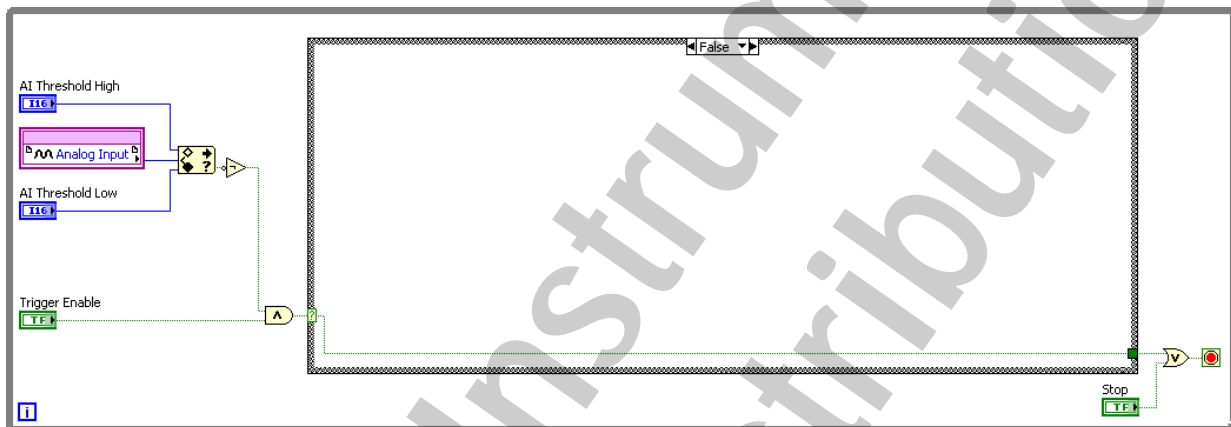


Figure 9-5. Custom Trigger AI.vi Block Diagram (False Case)

7. **Modify Custom Trigger AI Front Panel**, as shown in Figure 9-6.

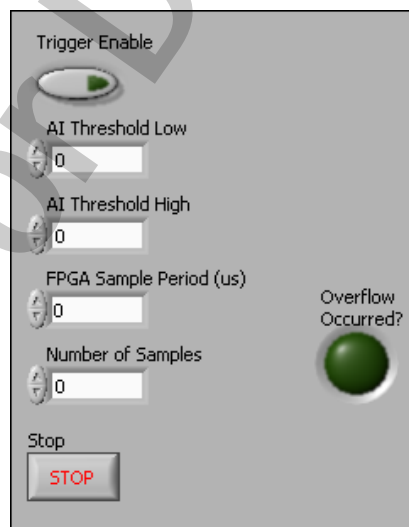


Figure 9-6. Custom Trigger AI VI Front Panel

- ☐ Change the Stop push button control into a stop button control.
 - Right-click the Stop control and select **Replace»Modern» Boolean»Stop Button**.
- 8. Save and close the VI. If you do not close this VI, the host VI you create in the following steps will not execute properly.
- 9. Build the block diagram for Custom Trigger Windows Host.vi, as shown in Figure 9-7. This VI runs on the PC host target 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 Custom Trigger AI.vi and reads the data that was written to Analog Input DMA. Build the block diagram using the following items:

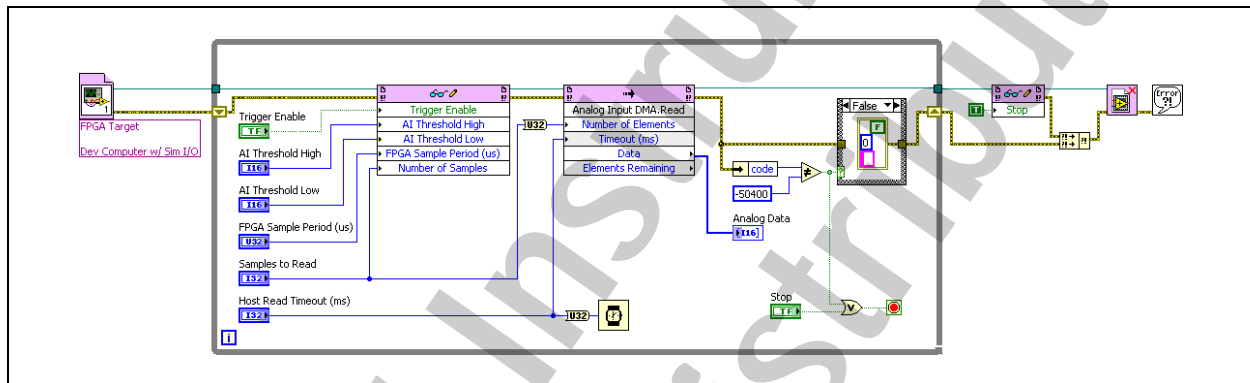


Figure 9-7. Custom Trigger Windows Host VI Block Diagram

- ☐ Open FPGA VI Reference—This function is located on the FPGA Interface Palette. This function opens a reference to Custom Trigger AI.vi.
 - Place this function on the block diagram. Right-click the function and select **Configure Open FPGA VI 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
 - Add a shift register to the left side of the While Loop.

- ❑ Read/Write Control—This function is located on the FPGA Interface Palette. It writes data to the controls on the front panel of your FPGA VI.
 - Expand the Read/Write Control to display five elements.
 - Click **Unselected** and select **Trigger Enable**.
 - Set the remaining elements as shown in Figure 9-7.
 - Create a control for each control input of this function.
- ❑ To Unsigned Long Integer—Place two instances of this function on the block diagram. This function converts the number input into an unsigned 32-bit integer.
- ❑ Wait (ms)—This function ensures a consistent loop rate, even when DMA FIFO data is available, before reaching the specified timeout.
- ❑ Invoke Method—This function is located on the FPGA Interface Palette. It reads data from the Analog Input DMA FIFO.
 - Right-click the function and select **Method»Analog Input DMA»Read**.
 - Right-click the Timeout (ms) input of this function and select **Create»Control**.
 - Create an indicator for the Data output of this function. Name the indicator Analog Data.
- ❑ Unbundle by Name—This function extracts the code element of the error cluster.
 - Click the Unbundle by Name function and select **Code**.
- ❑ Not Equal?—This function is used to handle error -50400. This error is generated if the attempt to read from Analog Input DMA times out. This will occur if the trigger conditions are not met in the time specified by the Timeout (ms) input to the Invoke Method function.
 - Right-click the y input of this function and select **Create»Control**. Set the value of this constant to -50400.

- ☐ Case Structure—This structure determines which action to take based on the output of the Not Equal? function. If error –50400 occurs, then the False case will overwrite the error cluster to remove the error. If no error occurs, or a different error occurs, then the True case executes, passing the error cluster through.
 - Right-click the error out output of the Invoke Method function and select **Create»Constant**. Place this constant in the False case of this structure.
- ☐ Or function—This function is used to stop execution of the While loop if the DMA Read method does not time out or if the user clicks on the Stop button.
 - Right-click the second input of this function and select **Create»Control**. Name this control Stop.
- ☐ Read/Write Control—Place this function outside of the While Loop to stop execution of the FPGA VI when the While Loop completes execution.
 - Configure this function to write to the Stop control of the FPGA VI.
- ☐ Merge Errors—Combine errors from the code inside of the While Loop and the Read/Write Control.
- ☐ Close FPGA VI Reference—This function closes the reference that you opened with Open FPGA VI Reference.
- ☐ Simple Error Handler
- ☐ Wire the diagram as shown.

10. Modify Custom Trigger Windows Host Front Panel, as shown in Figure 9-8.

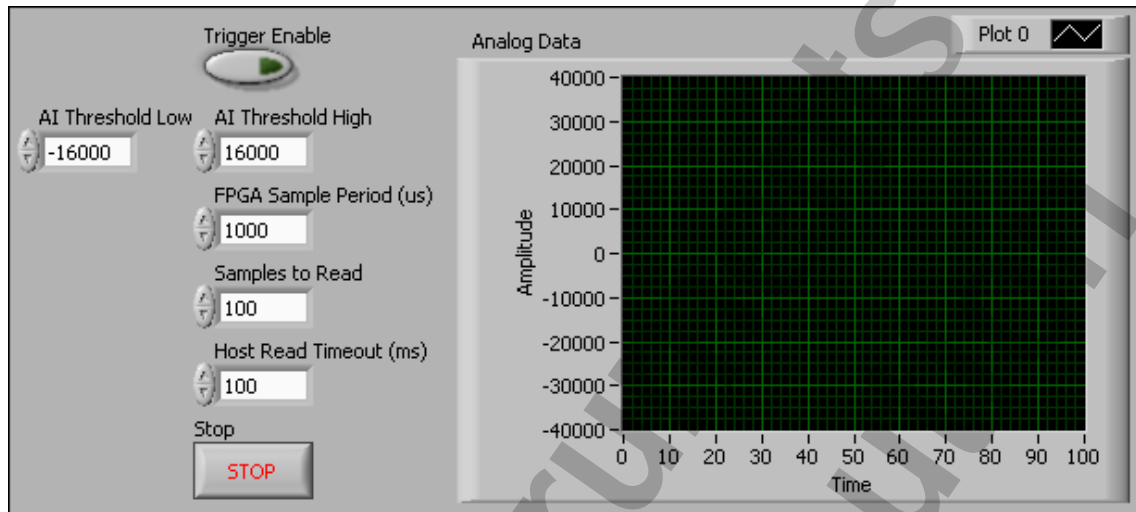


Figure 9-8. Custom Trigger Windows Host.vi Front Panel

- ☐ Replace the two-dimensional array named Analog Data with a Waveform Graph.
- ☐ Change the mechanical action Trigger Enable to Latch When Pressed.
 - Right-click on **Trigger Enable** and select **Mechanical Action» Latch When Pressed**.

11. Open the Project Explorer window and select **Save All**.

Test

1. Run the Custom Trigger Windows Host.vi on development machine. When you run the VI on the development machine, random data is generated for the Analog Input channel of Connector0.
 - ☐ Open the front panel of Custom Trigger Windows Host.
 - ☐ Use the following values for the controls:
 - AI Threshold High: 16000
 - AI Threshold Low: -16000
 - FPGA Sample Period (us): 1000

- Samples to Read: 100
- Host Read Timeout: 100



Note Execution on the development computer produces correct results in regard to data flow, but there is no timing accuracy.



Tip When testing on the development machine, it is important to pay attention to the data types that are being used for each control and indicator. Since AI Threshold High, AI Threshold Low, and Analog Data are all I16 numbers, their values can range from -32768 to 32767. The random data that is generated for Analog Data will use that full range. The threshold values above were selected so that the trigger conditions would be met after a few iterations of the FPGA VI.

☐ Run the VI.

- Notice that the Analog Data graph does not update.
- Click Trigger Enable. This enables the trigger that we developed in Custom Trigger AI.vi. Once the trigger conditions are met, Analog Data is updated with the requested number of samples and the VI stops execution.

2. Save and close the project.

End of Exercise 9-1

Exercise 9-2 AI Interleaved DMA

Goal

Create an FPGA VI that uses DMA FIFOs to transfer analog data from multiple devices and channels from the FPGA target to the RT host using the blocking method of data transfer.

Scenario

You are assigned the task of developing an application to acquire a fixed number of samples of accelerometer data from an NI 9233 and thermocouple data from an NI 9211. This data should then be transferred from the FPGA target to the RT host. Since there are only three DMA channels available, the data acquired from each module should be interleaved prior to transfer, and de-interleaved on the RT host VI.

Design

AI Low-Pass Filter VI Design

Create two target-to-host DMA FIFOs. Name the first FIFO *Accelerometer Data*. This FIFO is a stack of 1023 fixed-point values, configured as $\langle \pm, 24, 4 \rangle$. Name the second FIFO *Thermocouple Data*. This FIFO is a stack of 1023 fixed point values, configured as $\langle \pm, 24, -2 \rangle$.

This program acquires thermocouple data from two channels on the NI 9211 and accelerometer data from two channels on the NI 9233. The accelerometer data is interleaved and written to the *Accelerometer Data* DMA FIFO, and then thermocouple data is interleaved and written to the *Thermocouple Data* DMA FIFO.

Noise Processing VI Design

This VI passes data to the controls on the front panel of the FPGA VI and runs the VI. The DMA FIFOs are read and their data is presented on their respective waveform graphs.

Implementation

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 and select **Properties**.
 - ☐ In the General category, set the IP Address to the IP Address of your CompactRIO target.
3. Create a target-to-host DMA FIFO to transfer accelerometer data acquired from the NI 9233.
 - ☐ Right-click the FPGA Target and select **New»FIFO**.
 - ☐ Configure the FPGA FIFO Properties, as shown in Figure 9-9.

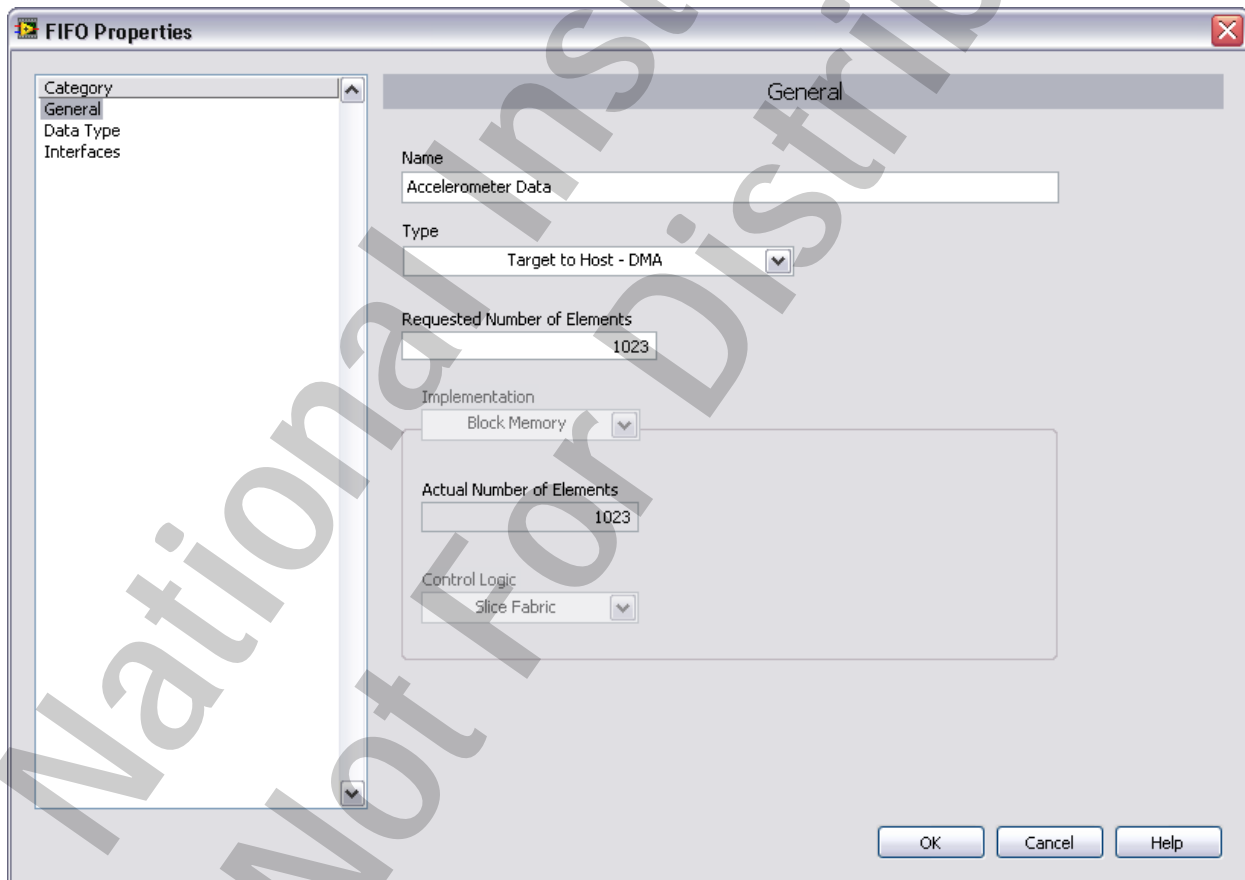


Figure 9-9. Accelerometer Data Properties – General

- ❑ Select the **Data Type** category. Configure the settings as shown in Figure 9-10.

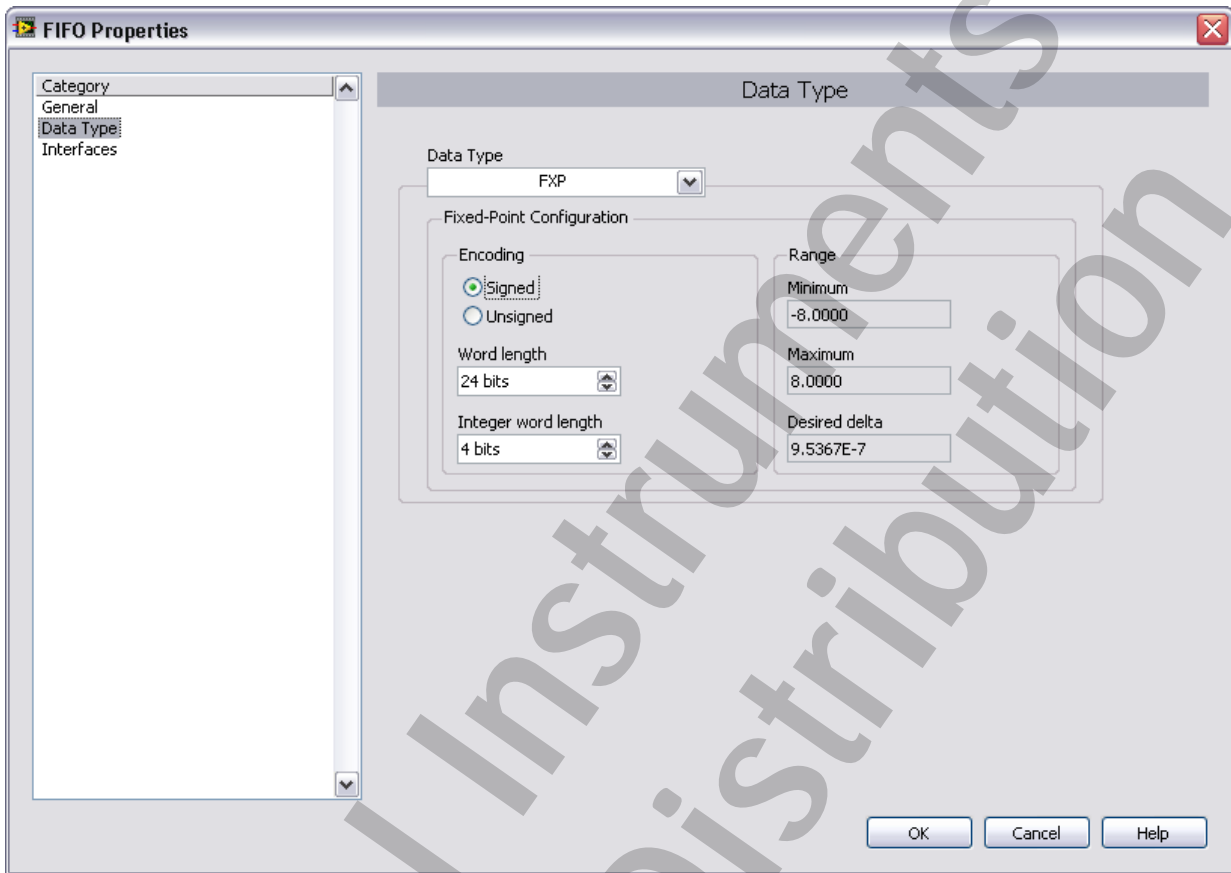


Figure 9-10. Accelerometer Data Properties – Data Type

4. Create a target-to-host DMA FIFO to transfer thermocouple data acquired from the NI 9211.
 - ☐ Right-click the FPGA Target and select **New»FIFO**.
 - ☐ Configure the FPGA FIFO Properties, as shown in Figure 9-11.

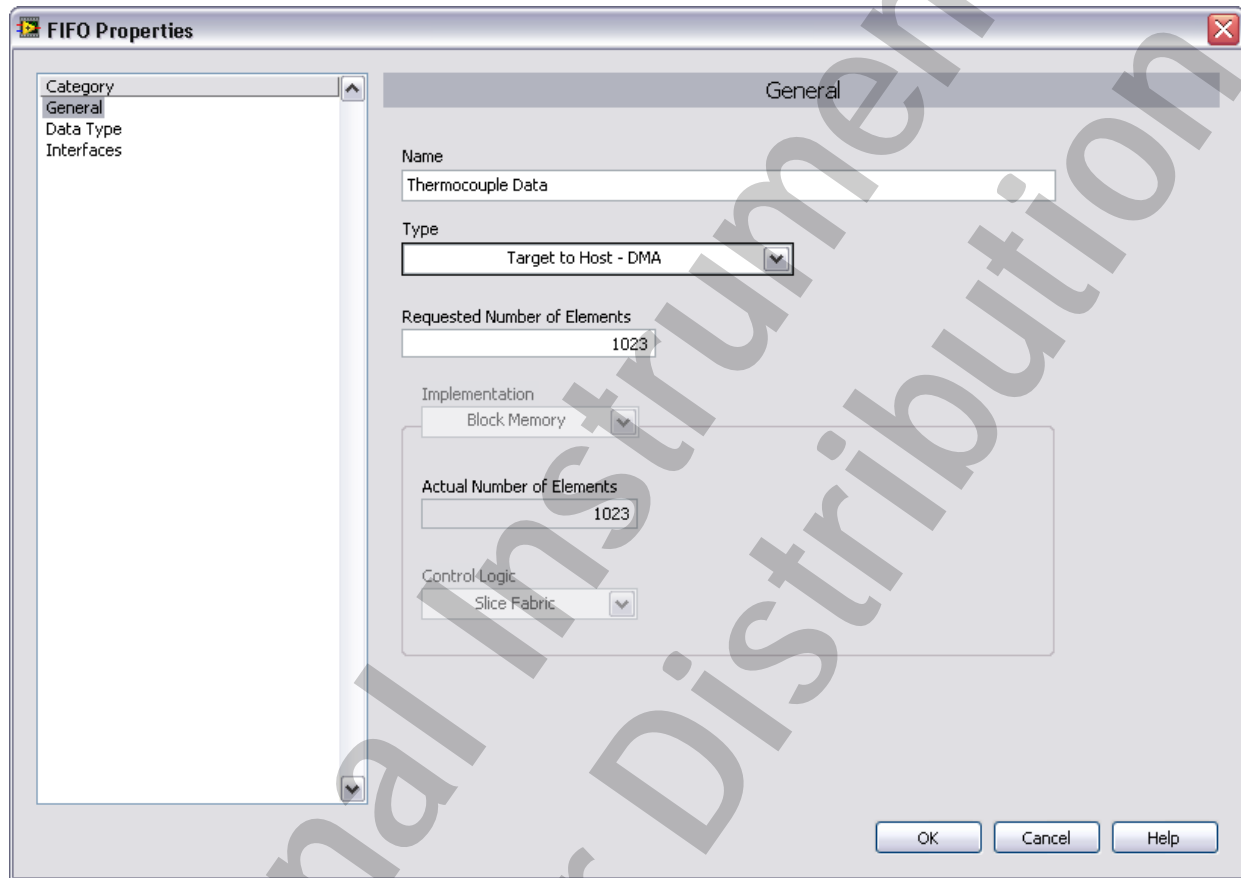


Figure 9-11. Thermocouple Data Properties – General

- ❑ Select the **Data Types** category. Configure the settings as shown in Figure 9-10.

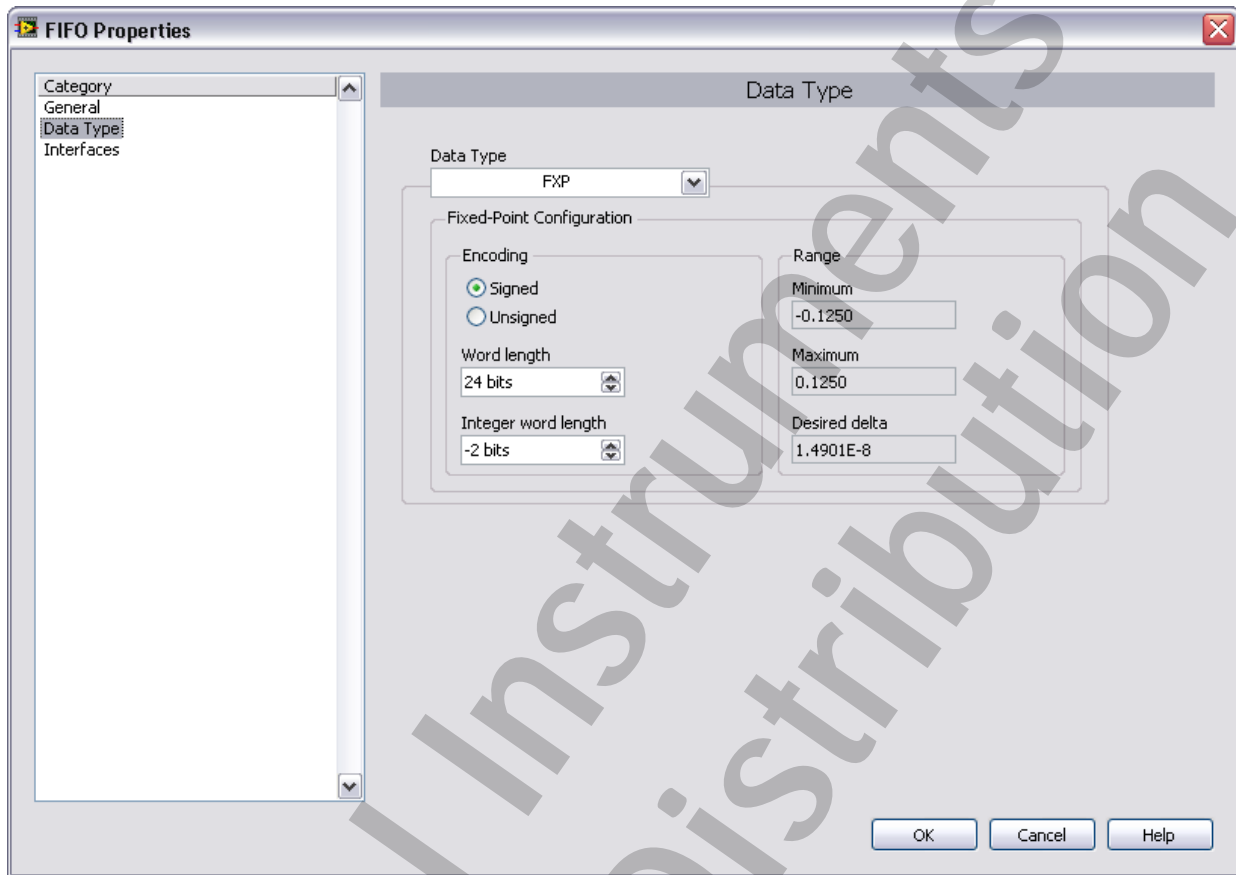


Figure 9-12. Thermocouple Data Properties – Data Type



Note While both FIFOs use fixed-point data, the configuration of the data generated by the NI 9211 is different from the data generated by the NI 9233. Coercing the data to the same type would allow for one FIFO to be used, but the precision of the data would be altered. To avoid this issue, this application makes use of two DMA channels.

5. Create a new VI on the FPGA Target. Name the VI `FPGA Interleaving.vi` and save it in the `<Exercises>\LabVIEW FPGA\Interleaved DMA` directory.

6. Create a new RT host VI. Name the VI RT Host Interleaving.vi and save it in the <Exercises>\LabVIEW FPGA\Interleaved DMA directory.
- ☐ Your project should resemble Figure 9-13.

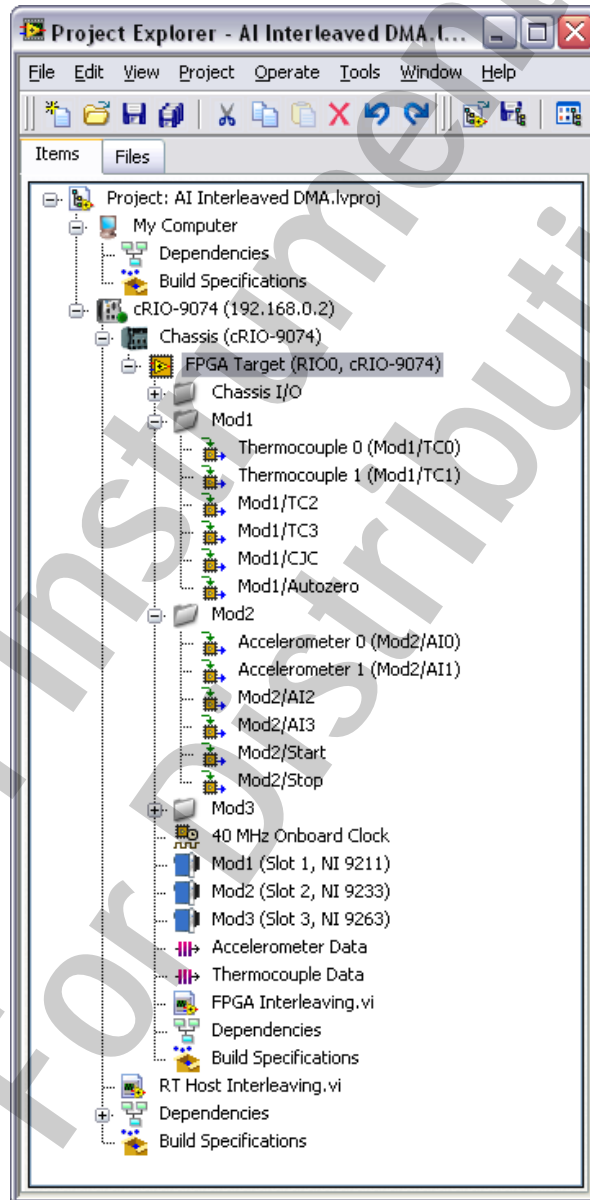


Figure 9-13. Interleaved DMA.lvproj

7. Open FPGA Interleaving.vi.
8. Using the following items, build the block diagram of the FPGA Interleaving VI to match Figure 9-14:

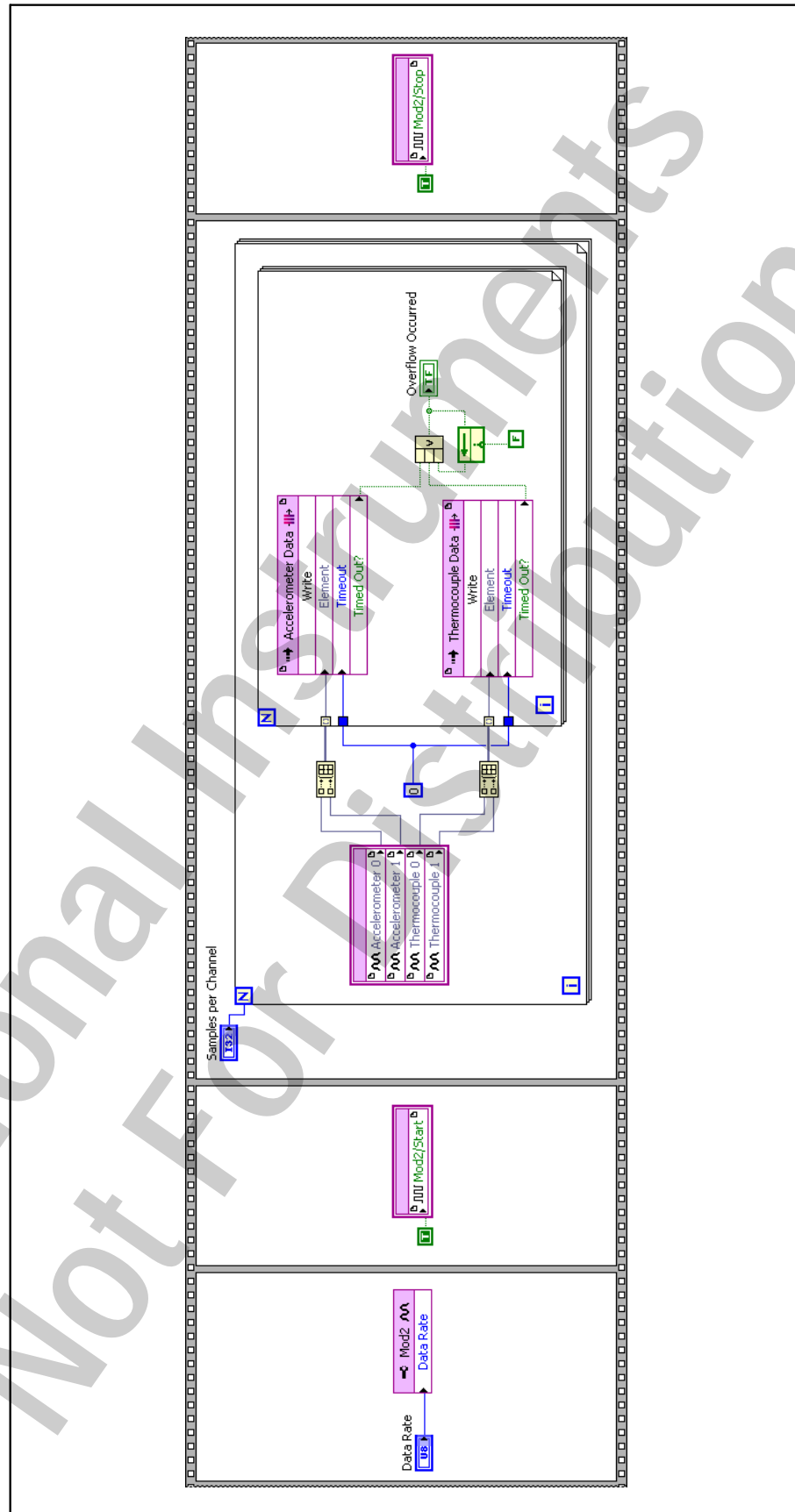


Figure 9-14. FPGA Interleaving Block Diagram

- ☐ Flat Sequence Structure with four frames.
- ☐ FPGA I/O Property Node—This node is located on the FPGA I/O palette. It enables you to configure certain properties for the I/O modules that you use. Different CompactRIO modules have different properties available for modification in this way. Place this node in the first frame of the sequence structure.
 - Configure the node to write to the Data Rate property of Mod2.
 - Right-click the node and select **Select Item»FPGA Target»Mod2**.
 - Click **Property** and select **Data Rate**.
 - Right-click the Data Rate input and select **Create»Control**.
- ☐ FPGA I/O Node—This node starts data acquisition of the NI 9233. Place this node in the second frame of the sequence structure.
 - Configure the node to write to Mod2/Start.
 - Create a true constant for the Mod2/Start input.
- ☐ For Loop—Place the For Loop into the third frame of the sequence structure.
 - Right-click the count terminal and select **Create»Control**. Name the control `Samples per Channel`. This loop executes once for each sample requested.
- ☐ FPGA I/O Node—This node acquires data from four analog input channels.
 - In the Project Explorer, click **Mod2»Accelerometer 0** and drag it to the block diagram.
 - Expand the node to display four I/O channels.
 - In addition to Accelerometer 0, configure the I/O node to read from **Accelerometer 1**, **Thermocouple 0**, and **Thermocouple 1**.

- ❑ Build Array—Place two instances of this function on the block diagram.
 - Build an array consisting of the acquired accelerometer data.
 - Build an array consisting of the acquired thermocouple data.
- ❑ For Loop—This loop iterates through the two arrays that you created.
- ❑ FIFO Write—Place two FIFO Method nodes on the block diagram, one for each DMA FIFO. Each FIFO Write method alternates between the two analog input channels that correspond to that FIFO.
 - In the Project Explorer window, click and drag Accelerometer Data and Thermocouple Data to the block diagram.
 - Wire the appended array output of the accelerometer Build Array function to the Element input of the Accelerometer Data FIFO Write method node. Verify that indexing is enabled on the data tunnel.
 - Repeat the above step for the thermocouple array and the Thermocouple Data FIFO Write method node.
 - Right-click the Timeout input of either FIFO Write method node and select **Create»Constant**. Place this constant outside the For Loop and wire it to the Timeout input of both FIFO Write method nodes.
- ❑ Compound Arithmetic—Use this function to latch the value of Overflow Occurred? to determine whether or not either FIFO Write method ever times out.
 - Configure this function to perform the Or operation.
 - Expand this function to display three inputs.
 - Wire the Timed Out? output of the two FIFO Write nodes to the first two inputs of this function.
 - Right-click the result output of this function and select **Create»Indicator**. Name the indicator *Overflow Occurred*.
 - Wire the result output to the third input to create a feedback node. Initialize the feedback node as FALSE.

- ☐ FPGA I/O Node—This node stops the data acquisition of the NI 9233.
 - Configure the node to write to Mod2/Stop.
 - Create a true constant for the I/O Item input.

9. Arrange the front panel so that it resembles Figure 9-15.



Figure 9-15. FPGA Interleaving Front Panel

10. Test the application on the development computer.



Note Execution on the development computer is not supported by RT targets. As a result, we will test the FPGA VI by adding indicators to show the data that is going to be written to the DMA FIFOs.

- ☐ Set the FPGA target to execute on the development computer with simulated I/O.
- ☐ Create indicators for the output of each Build Array function. This is the data that will be written to each of the two DMA FIFOs.
- ☐ Save the VI.
- ☐ Set the Samples per Channel control to 2 and run the VI. For each iteration of the outer For Loop, two Accelerometer samples and two Thermocouple samples are written to the respective FIFOs for a total of four elements each.
- ☐ Check the values stored in the array indicators that you created. There are two elements in each array, since two channels of data are being acquired and written to the FIFO at a time.
- ☐ Remove the array indicators that you created and save the VI.

11. Start the compilation of this VI.

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

12. While the FPGA VI compiles, open `RT Host Interleaving.vi`.

13. Using the following items, modify the block diagram to match Figure 9-16:

National Instruments
Not For Distribution

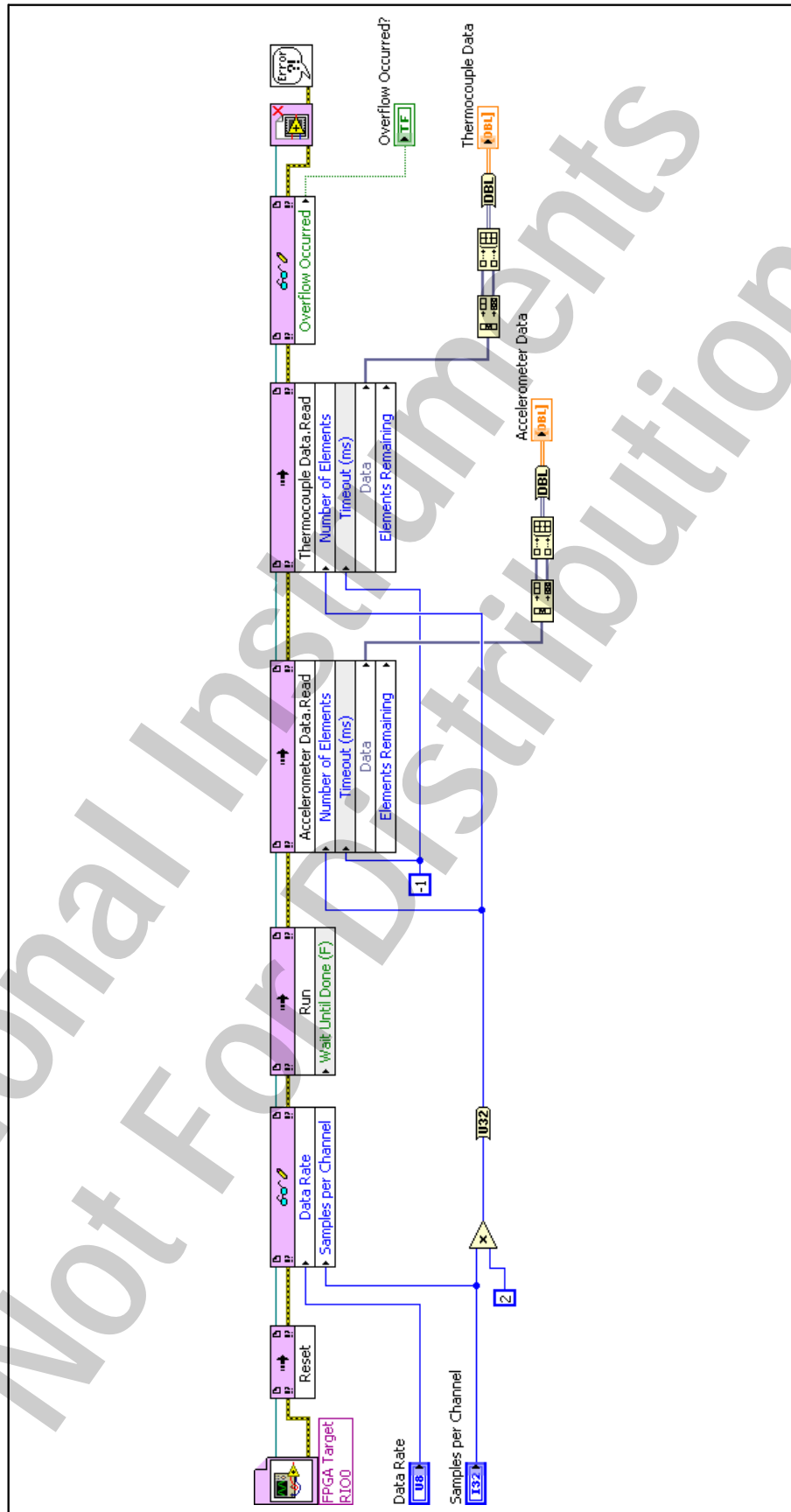


Figure 9-16. RT Host Interleaving Block Diagram

☐ Open FPGA VI Reference

- Configure this VI to open `FPGA Interleaving.vi`.
- In the Configure Open FPGA VI Reference window, disable the **Run the FPGA VI** checkbox. You run the VI after you write values to the FPGA controls.
- Disable the **Dynamic Mode** checkbox.



Note For each Read/Write Control and Invoke Method that you use, you must wire the FPGA VI Reference created by Open FPGA VI Reference to the FPGA VI Reference In terminal of the node. This populates the available controls, indicators, and methods.

☐ Invoke Method

- Configure this Invoke Method function to call the Reset method to clear the DMA FIFOs.

☐ Read/Write Control

- Configure this node to write to the Data Rate and Samples per Channel controls of `FPGA Interleaving.vi`.
- Right-click the Data Rate input and select **Create»Control**.
- Right-click the Samples per Channel input and select **Create»Control**.

☐ Multiply—Calculate the number of elements that are read from each DMA FIFO.

- Multiply the value of the Samples per Channel control by 2 since each FIFO is transferring data for two channels.

☐ To Unsigned Long Integer—This function converts the output of the Multiply function to a U32 for use with the DMA Read Invoke Nodes.

☐ Invoke Method

- Configure this Invoke Method function to call the Run method to run the FPGA VI.

- ☐ **Invoke Method**—Place two Invoke Methods onto the block diagram. These nodes read data from the two DMA FIFOs.
 - Right-click on the first Invoke Method and select **Method»Accelerometer Data»Read**.
 - Right-click the Timeout (ms) input and select **Create»Constant**. Set the value of that constant to -1.
 - Right-click on the second Invoke Method and select **Method»Thermocouple Data»Read**.
- ☐ **Decimate 1D Array**—Place two instances of this function on the block diagram. This function separates the interleaved Data output of each DMA FIFO Read method into the data acquired from each channel.
- ☐ **Build Array**—Place two instances of this function on the block diagram. This function creates a two-dimensional array used to plot the data acquired from both channels of the DMA FIFO.
- ☐ **To Double Precision Float**—Place two instances of this function on the block diagram. This function converts the fixed-point data acquired from the DMA FIFOs into data that can be easily plotted on a waveform graph.
 - Create an indicator for the double precision float output of each instance of this function. Name the indicator for the NI 9233 data Accelerometer Data. Name the indicator for the NI 9211 data Thermocouple Data.
- ☐ **Read/Write Control**
 - Configure this node to read the value of Overflow Occurred from FPGA Interleaving.vi.
 - Right-click the Overflow Occurred output and select **Create»Indicator**.
- ☐ **Close FPGA VI Reference**
- ☐ **Simple Error Handler**
- ☐ Wire the block diagram as shown in Figure 9-16.

14. Modify the front panel so that it matches Figure 9-17. Develop the front panel using the following items:

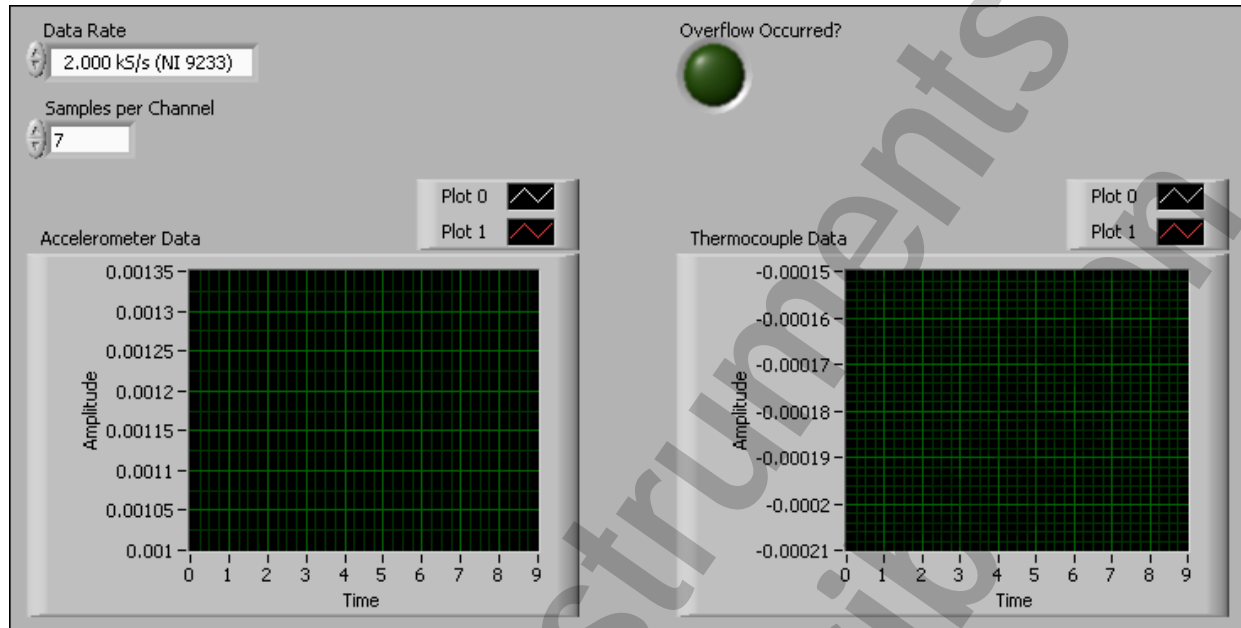


Figure 9-17. RT Host Interleaving Front Panel

- ☐ Waveform Graph—Replace Accelerometer Data and Thermocouple Data with waveform graphs to better display the data transferred.
 - Right-click on Accelerometer Data and select **Replace»Graph»Waveform Graph**.
- ☐ Repeat the previous step for Thermocouple Data.

15. Save the VI.

Testing

1. Open and examine the data sheets for the NI 9211 and NI 9233, located in the <Exercises>\Hardware Manuals directory. Take note of the max sample rate of each module. Always keep in mind the max sample rates of your modules as you develop your FPGA and Host VIs.
2. Test the application on the FPGA target.
 - ☐ In the RT Host Interleaving VI, enter the following values in the front panel controls:
 - Data Rate: 2.000 kS/s (NI 9233)
 - Samples per Channel: 7
 - ☐ Based on the values you input, how long should it take to acquire 7 samples? How long to acquire 14 samples?
 - ☐ Run the VI. Note how long it takes to acquire samples.
 - How long does it take to acquire 7 samples? _____
 - How long does it take to acquire 14 samples? _____

The NI 9233 can acquire data at rates ranging from 2 to 50 kS/s for each channel. However, the NI 9211 is only able to acquire data at 14 samples/second across all channels and that rate is divided by two when two channels are used. The slower rate of the NI 9211 limits the overall speed of this application to a maximum of 7 samples per second.

- To test this further, set your samples per channel to 35 and run the VI. The execution should finish in approximately 5 seconds.

This illustrates the importance of understanding the properties of your CompactRIO modules when acquiring data and transferring from the target to your host. If the timeout value for the DMA Read nodes were set to less than five seconds, those operations would time out.

3. Close the VIs and project. Save any changes.

End of Exercise 9-2

Notes

National Instruments
Not For Distribution

Notes

National Instruments
Not For Distribution

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 **Reentrant execution** checkbox is enabled 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:

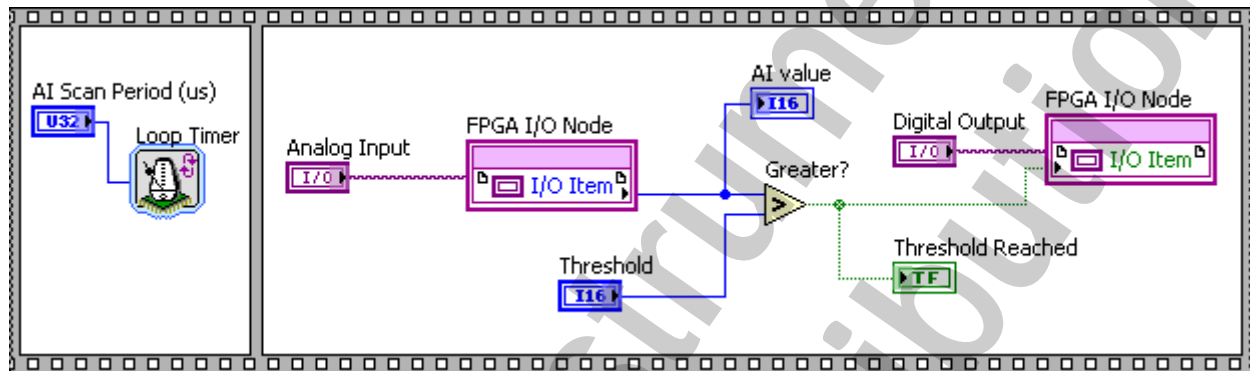


Figure 10-1. Max AI (SubVI) VI Block Diagram

- ☐ 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
 - 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**.
 - 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.

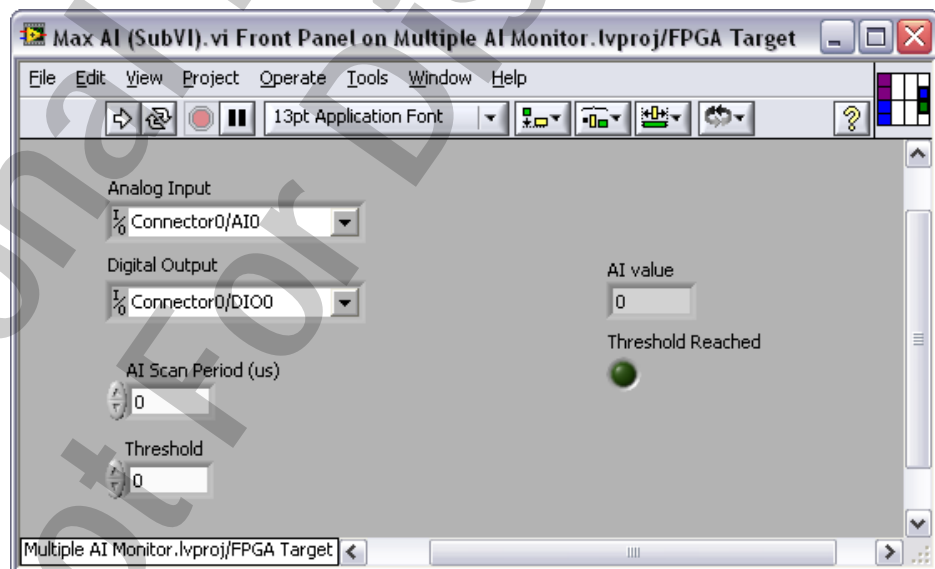


Figure 10-2. Max AI (SubVI) Front Panel with Connector Pane

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:

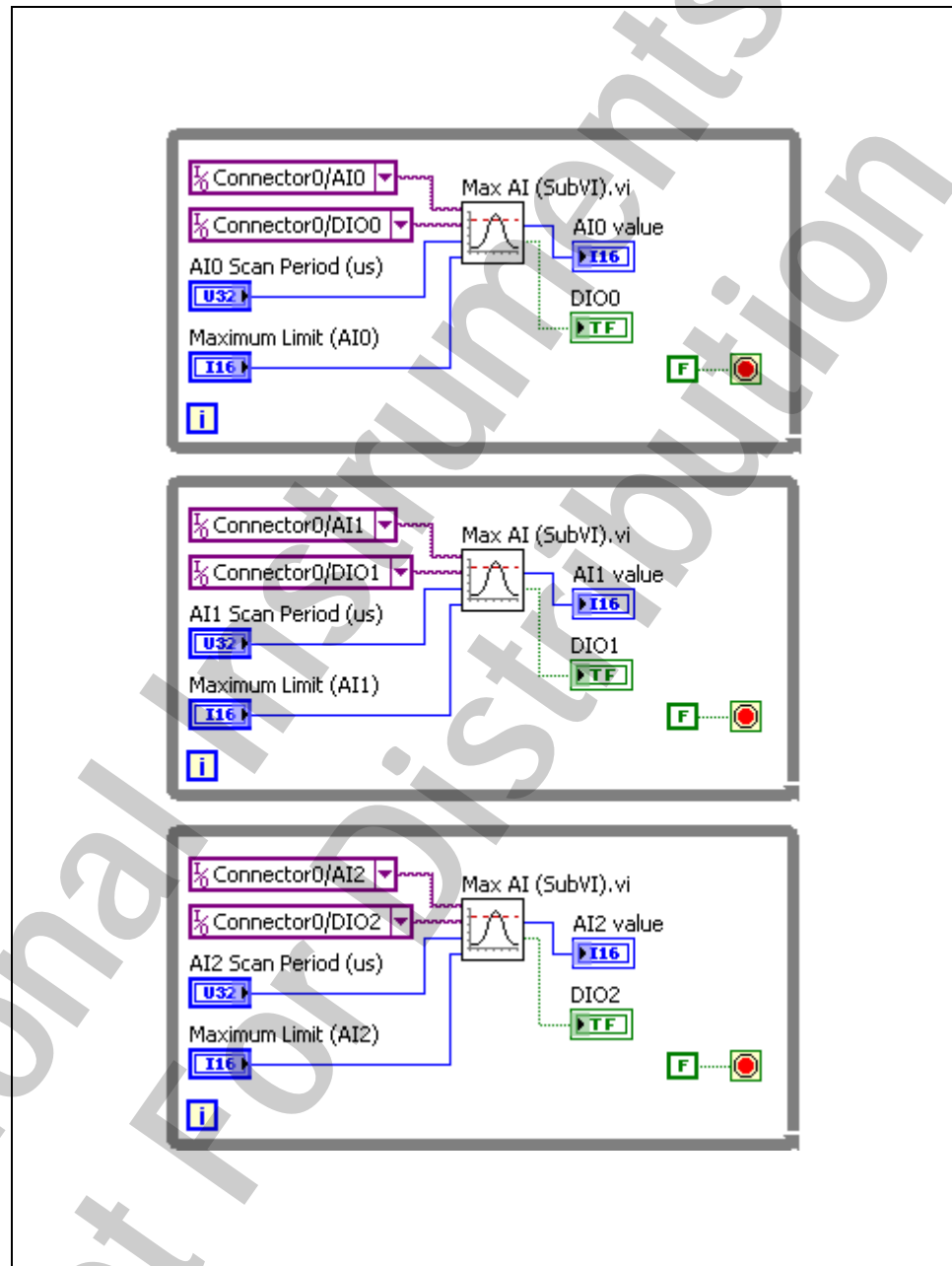


Figure 10-3. Multiple AI Monitor FPGA (Main) VI Block Diagram

- ☐ 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.

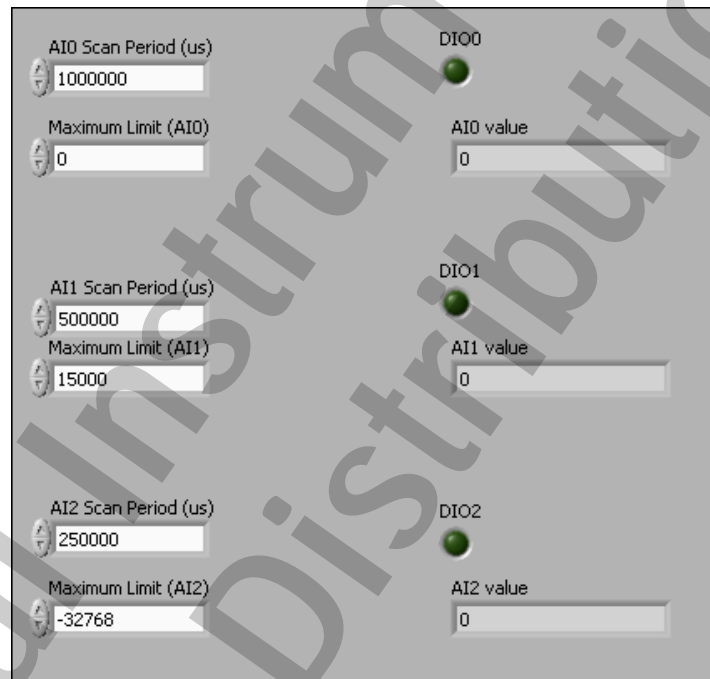


Figure 10-4. Multiple AI Monitor FPGA (Main) VI Front Panel

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

Notes

National Instruments
Not For Distribution

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.

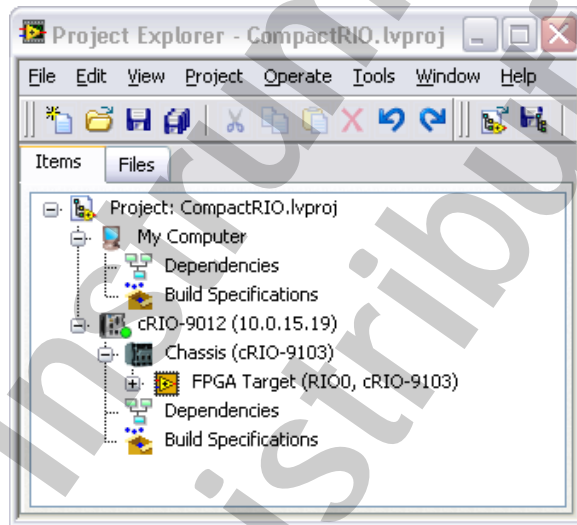


Figure A-1. LabVIEW FPGA Project for cRIO-9012 and cRIO-9103

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


- A. Introduction to LabVIEW FPGA
- B. LabVIEW FPGA Basics
- C. FPGA Programming Basics
- D. FPGA I/O
- E. Timing an FPGA VI
- F. Data Sharing on FPGA
- G. Single-Cycle Timed Loops
- H. Basic Host Integration – PC/Real-Time
- I. DMA Data Transfers
- J. Modular Programming
- K. Appendix A – Pipelining

Lesson 1

Introduction to LabVIEW FPGA

TOPICS

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

How does an FPGA work?

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


When is an FPGA used?

- Custom hardware, where it doesn't make sense to pay the high price of developing an ASIC
- Reconfiguration required after deployment

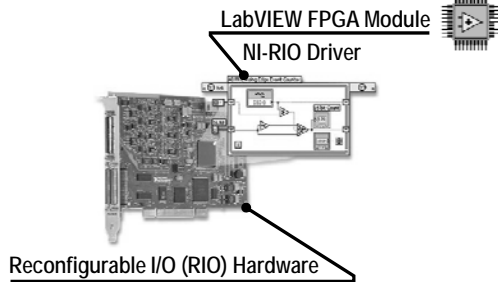

ni.com/training

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. LabVIEW FPGA System



ni.com/training

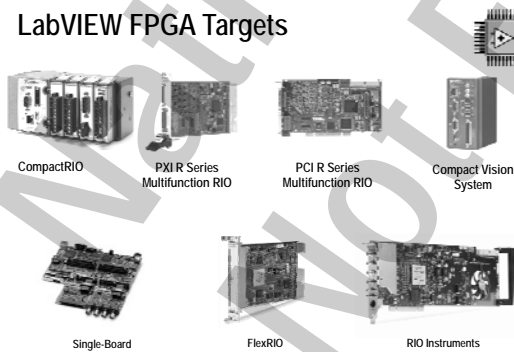
LabVIEW FPGA Module

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

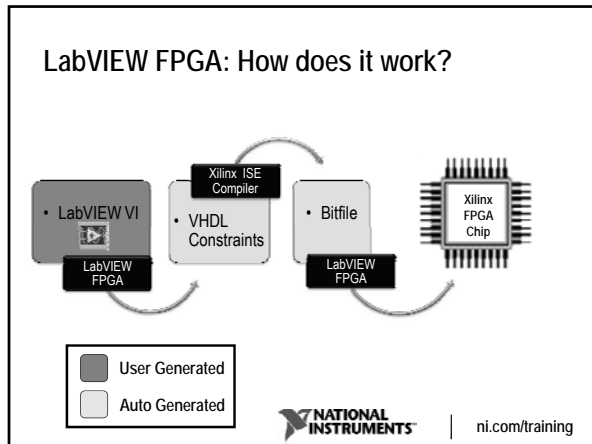


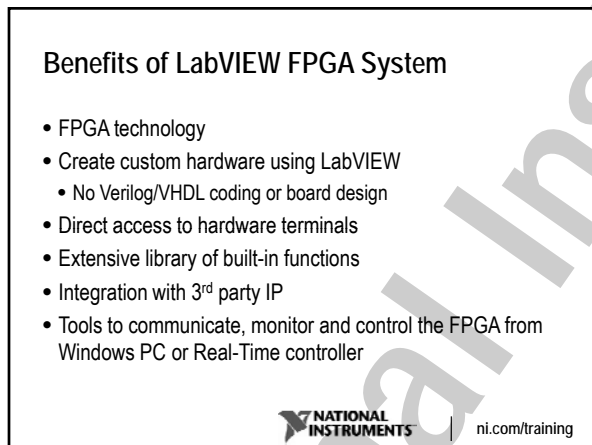
ni.com/training

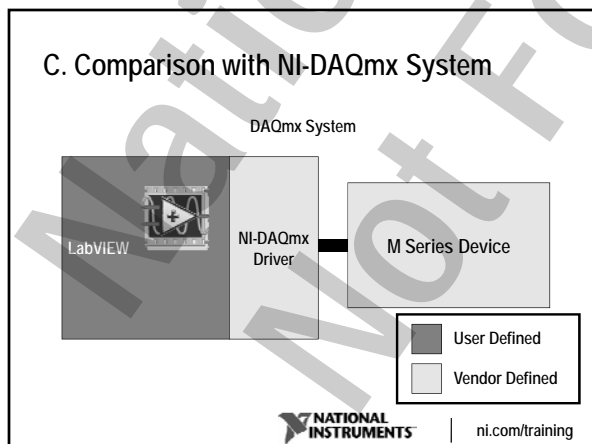
LabVIEW FPGA Targets

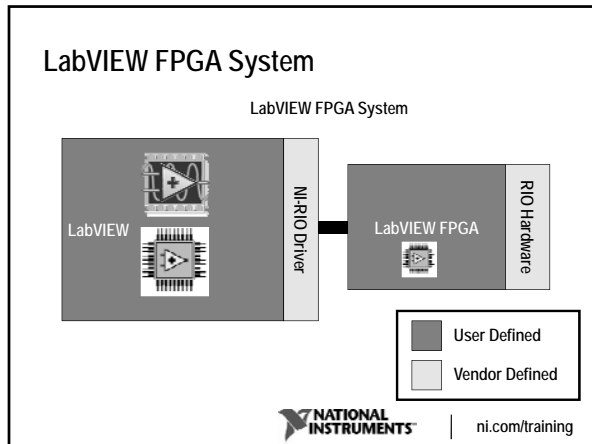


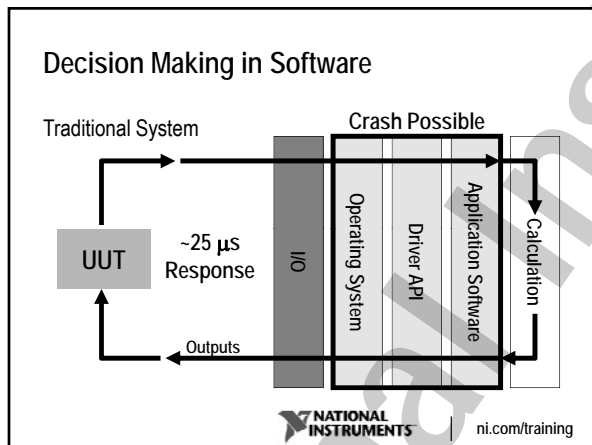
ni.com/training

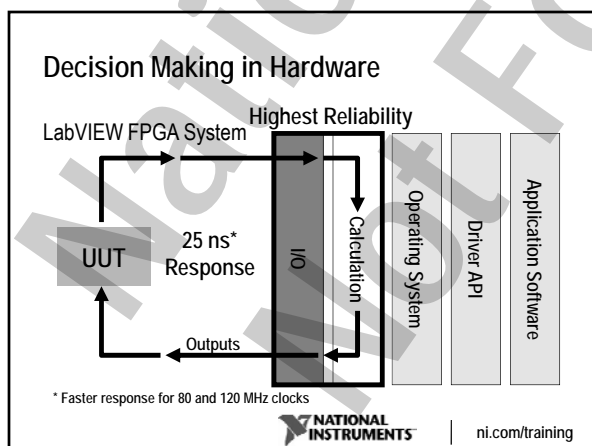












D. LabVIEW FPGA Applications

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. PharLap
 - 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. PharLap
 - 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

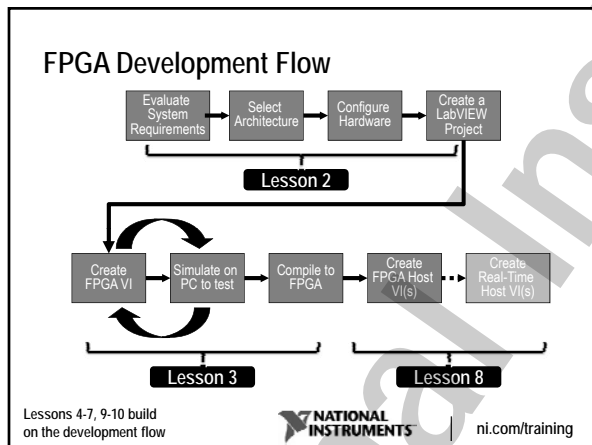
LabVIEW FPGA Basics

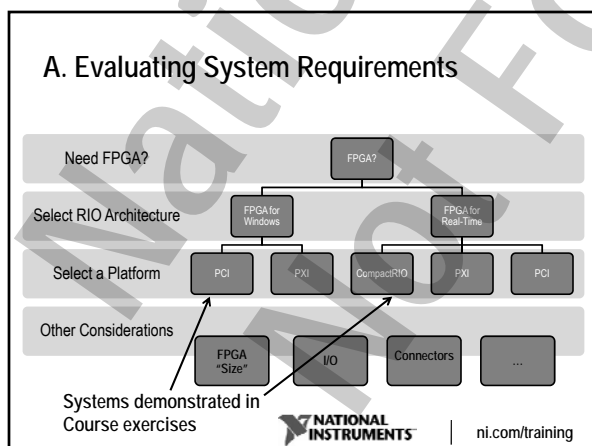
TOPICS

- A. Evaluating System Requirements
- B. FPGA System Architectures
- C. RIO Platforms
- D. Software Installation

- E. Windows Hardware Configuration
- F. Real-Time Hardware Configuration
- G. Creating a LabVIEW FPGA Project

ni.com/training





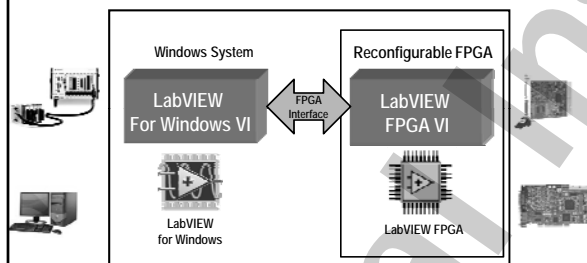
B. FPGA System Architectures

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



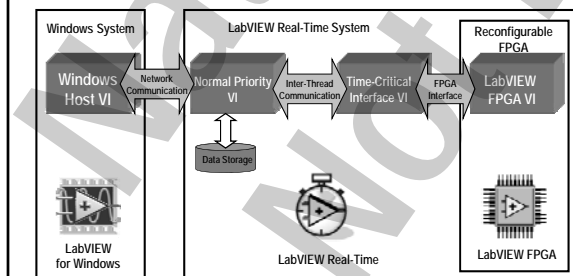
ni.com/training

FPGA – Windows



ni.com/training

FPGA – Real-Time



ni.com/training

LabVIEW Programming & CompactRIO

User Interface
LabVIEW

Real-Time Processor
LabVIEW Real-Time

Reconfigurable FPGA
LabVIEW FPGA

NATIONAL INSTRUMENTS | ni.com/training

C. Common Reconfigurable I/O (RIO) Platforms

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

NATIONAL INSTRUMENTS | ni.com/training

R Series Multifunction RIO

- PXI, PXI Express, or PCI form factor
- Define custom data acquisition
- Integrate FPGA with Analog and Digital I/O
- High channel count

NATIONAL INSTRUMENTS | ni.com/training

Building a CompactRIO System

Select the Real-Time Controller


Processor speed
Flash memory size

Select the Reconfigurable Chassis

Number of slots
FPGA size
Integrated or not


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.

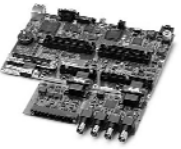


NATIONAL INSTRUMENTS | ni.com/training

Other CompactRIO Form Factors



NI CompactRIO Integrated Controller



NI Single-Board RIO

NATIONAL INSTRUMENTS | 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, Look-up-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

FPGA Resource Table

	Virtex II 1000	Spartan 3 1000	Virtex 5 LX30	Virtex II 3000	Virtex 5 LX50	Spartan 3 2000	Virtex 5 LX85	Virtex 5 LX110
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
Multipliers	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	cRIO-9116 PXI-7853R	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

D. Software Installation

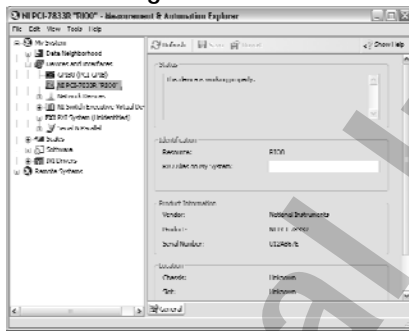
Before installing any device:

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



ni.com/training

E. Windows Configuration



ni.com/training

F. Real-Time Configuration

Select Host-Target Network Setup

Detect the Remote Target

Configure Target Network Settings

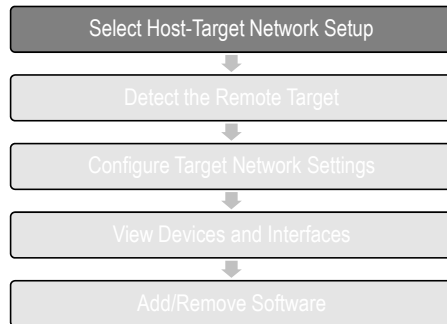
View Devices and Interfaces

Add/Remove Software



ni.com/training

Real-Time Configuration



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 assigned	<ul style="list-style-type: none"> •IP address automatically assigned by DHCP server. •DHCP is a common network protocol for administering IP addresses.
Link-local	<ul style="list-style-type: none"> •IP address allocated using automatic private IP addressing when DHCP server not available. •AutoIP attempts to assign an address using DHCP first then link local if DHCP fails. •Address range: 169 . 254 . x . x
Static IP	•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.



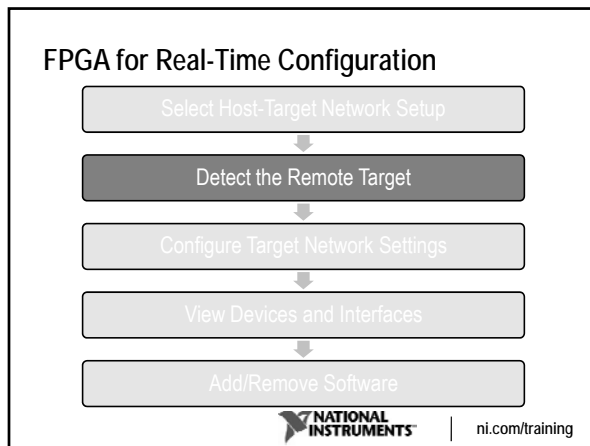
ni.com/training

Ethernet Cabling Options

Option	Use when...
Standard	<ul style="list-style-type: none"> •Host and target are connected with hub, router, or network switch
Crossover	<ul style="list-style-type: none"> •Directly connecting host and target



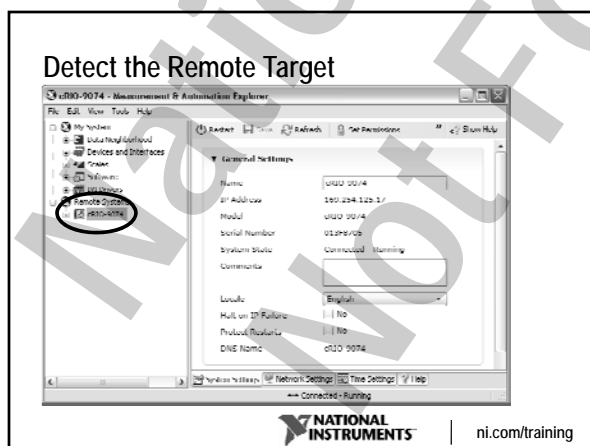
ni.com/training

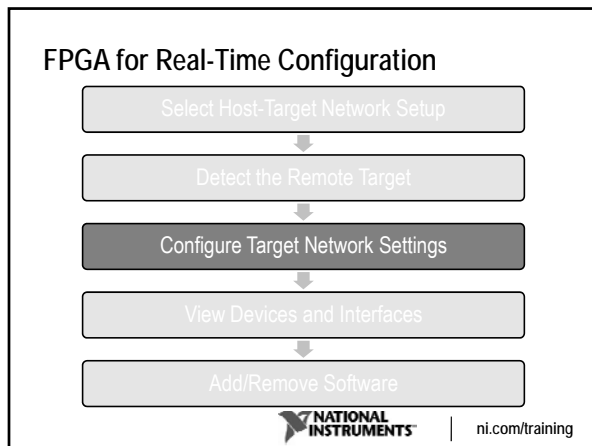


Detect Remote Target - CompactRIO

1. Connect CompactRIO to 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

NATIONAL INSTRUMENTS | ni.com/training





Configure Network Settings – IP Address

Recommended approach:

- Attempt to obtain IP address automatically
 - Will work if connected to DHCP or controller supports AutoIP
- If fail, then specify IP address using suggested value

NATIONAL INSTRUMENTS | ni.com/training

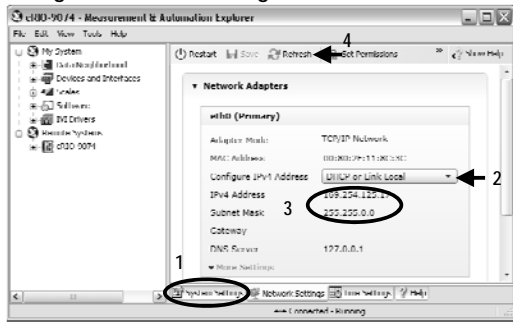
Configure Network Settings – DHCP & AutoIP

Automatic IP Address from DHCP Server or APIPA

- Select Obtain an IP address automatically
- Click Apply
- Notes:
 - Address allocated on each reboot. Not guaranteed to be the same address.
 - Might need to enable IP Reset switch

NATIONAL INSTRUMENTS | ni.com/training

Configure Network Settings – DHCP & AutoIP



ni.com/training

Configure Network Settings – Link Local or Static IP

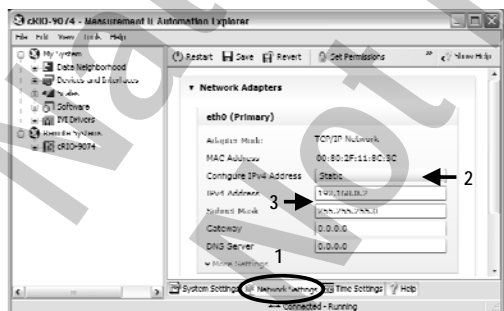
Manually assign a static IP

- Select Use the following IP address
- Click on Suggest Values... to view suggested value
- Click OK
- Click Apply
- Notes:
 - Address must be on same subnet as Windows host
 - Might need to enable IP Reset switch or set in Safe Mode

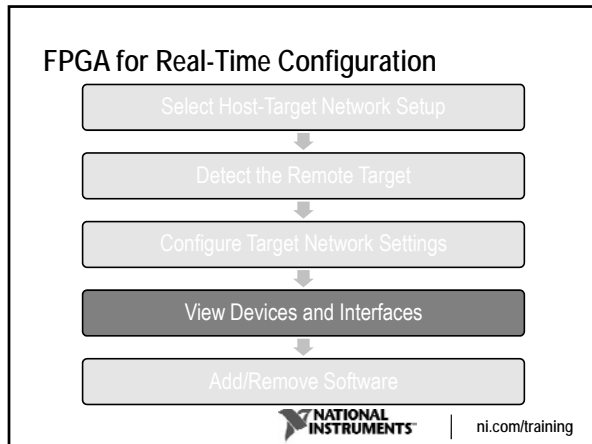


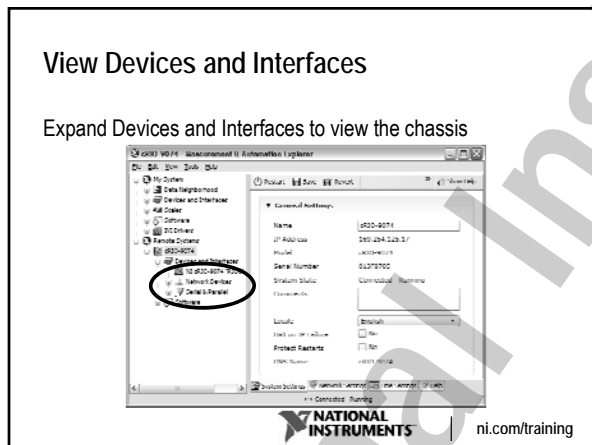
ni.com/training

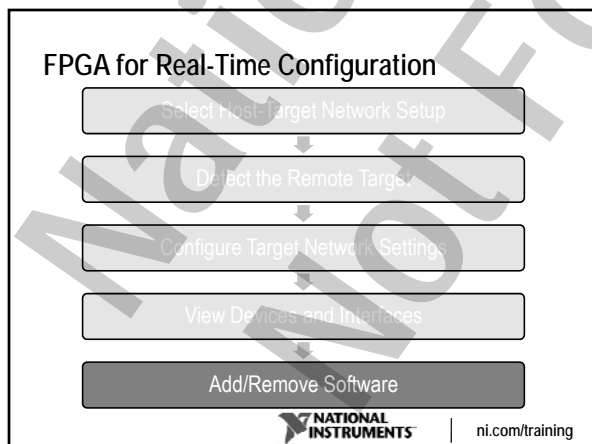
Configure Network Settings – Link Local or Static IP



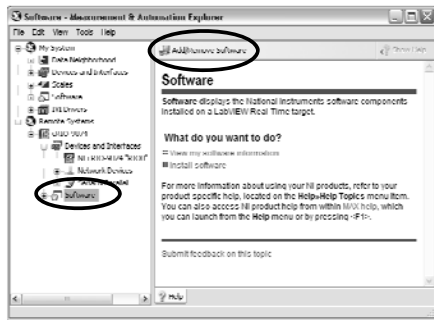
ni.com/training





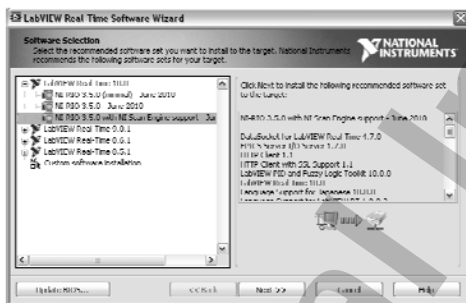


Add/Remove Software



ni.com/training

Add/Remove Software



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?
- How would configuration change if you used DHCP?

DISCUSSION

G. 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 a local
 - Add under "Project" if device is a 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

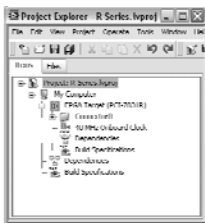


- Existing target or device
 - Use when your device can be accessed from your development computer
- New target or device
 - Use when you are not connected with a physical target or device is not present.



ni.com/training

Windows Project



- FPGA Target is under My Computer



ni.com/training

New Target – Real-Time Target



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



ni.com/training

Add Targets and Devices – Real-Time 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.



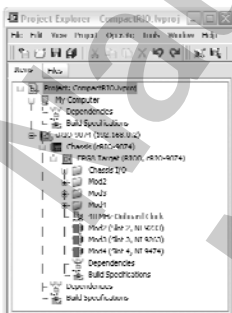
ni.com/training

Select Programming Mode - CompactRIO



ni.com/training

Project with CompactRIO Controller



- My Computer and cRIO target are siblings
 - Bright green dot → target connected
 - Dark green dot → target not connected
- Chassis is under cRIO target
- FPGA target is under the Chassis



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 PCI-7831R
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?
 - Manually add controller, chassis, and FPGA target
- When would you not want LabVIEW to auto-discover your C Series modules?

DISCUSSION

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

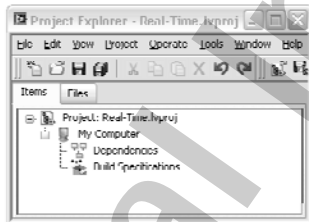
1. Which of the following operations are performed in MAX?
 - a. Install software on an R Series board
 - b. Install software on a CompactRIO target
 - c. Verify connection between the host and a real-time target
 - d. Assign the IP address of your host computer



ni.com/training

Summary—Quiz

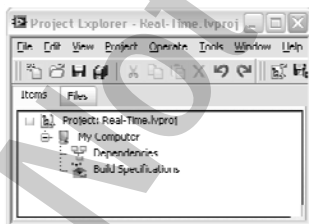
2. Under which node do you add a CompactRIO target?
 - a. Real-Time.lvproj
 - b. My Computer
 - c. Dependencies
 - d. Build Specifications



ni.com/training

Summary—Quiz Answer

2. Under which node do you add a CompactRIO target?
 - a. Real-Time.lvproj
 - b. My Computer
 - c. Dependencies
 - d. Build Specifications

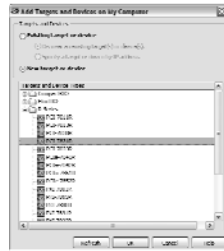


ni.com/training

Summary—Quiz

3. When would you select New target or device?

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

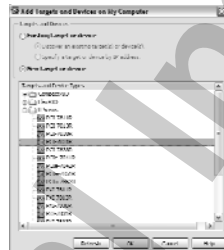


ni.com/training

Summary—Quiz Answer

3. When would you select New target or device?

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




ni.com/training

Lesson 3

FPGA Programming Basics

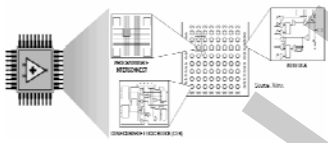
TOPICS


A. Introduction	E. Selecting an Execution Mode
B. Defining FPGA Logic with LabVIEW	F. Compiling the FPGA VI
C. Developing the FPGA VI	G. Basic Optimizations
D. Interactive Front Panel Communication	

 | ni.com/training

A. Introduction

FPGA Layout and Components




 | ni.com/training

How an FPGA Works

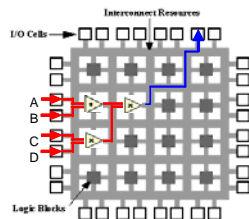
- Programmable interconnect switches and wires route signals between various hardware resources in an FPGA
- Hardware resources include logical gates, flip-flops, and block memories

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

 | ni.com/training

How FPGA Works - Example

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

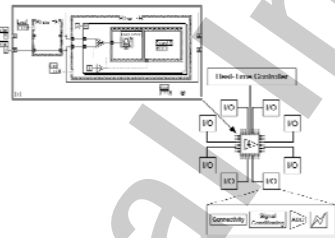


ni.com/training

B. Defining FPGA Logic with LabVIEW

LabVIEW FPGA

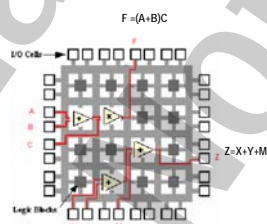
- Do not have to learn VHDL or Verilog
- True parallel execution
- Deterministic



ni.com/training

True Parallel Execution

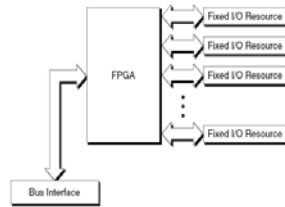
$F = (A+B)C$ and $Z = X+Y+M$ in separate gates on an FPGA



ni.com/training

I/O, FPGA, and Host Communication

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



ni.com/training

Timing Features 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

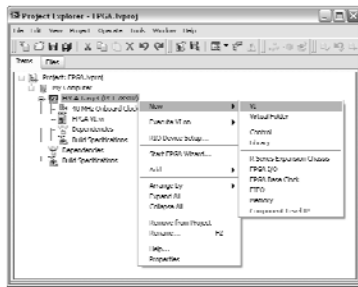
C. Developing the FPGA VI

- 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
- Use integer or fixed-point math to perform calculations



ni.com/training

Add a VI Under the FPGA Target

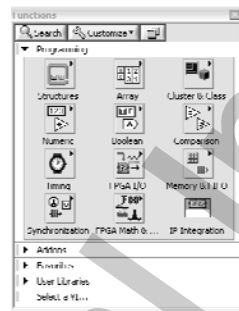


**NATIONAL
INSTRUMENTS**

[nvidia.com/training](https://www.nvidia.com/training)

FPGA Palettes

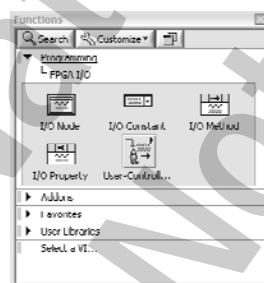
- VIs under the FPGA target inherit the FPGA Function Palette
- Many palettes are similar to LabVIEW for Windows
- FPGA-specific palettes
 - FPGA I/O
 - Memory & FIFO
 - Synchronization
 - FPGA Math & Analysis
 - IP Integration



**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 space on the FPGA
- Functions perform point-by-point calculations



**NATIONAL
INSTRUMENTS**

[nvidia.com/training](https://www.nvidia.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



**NATIONAL
INSTRUMENTS**

ni.com/training

D. Interactive Front Panel Communication

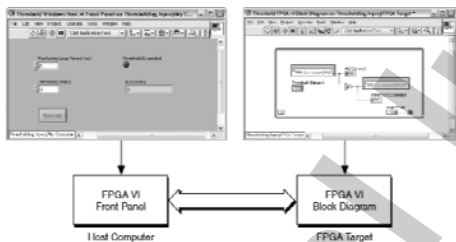
FPGA has no user interface

- 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

E. 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

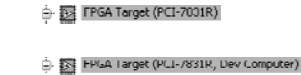
- Right-click the FPGA Target and select Execute VI on to select mode directly



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

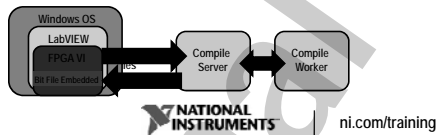
Note how quickly you were able to progress from development to testing.

Have you ever tried troubleshooting a VI without using the debugging tools?

DISCUSSION

F. Compiling the FPGA VI

- Turn the FPGA VI into executable code
 - FPGA Module compiles VI
 - Graphical code translated to VHDL
 - Xilinx ISE compiler creates circuit from VHDL
 - Compiler optimizes the implementation
- A bitstream file results
- Bitstream loads at run-time



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

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 generations 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

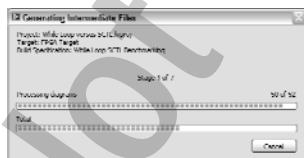
1. Generation of Intermediate Files (HDL code)
2. Estimation of resource usage
3. HDL Compilation, Analysis and Synthesis
4. Mapping
5. Placing and Routing
6. Generating Bitstream



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

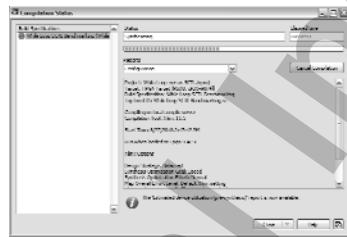
Begin Exercise 3-2 to start compilation of a working VI to run on an FPGA target.

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

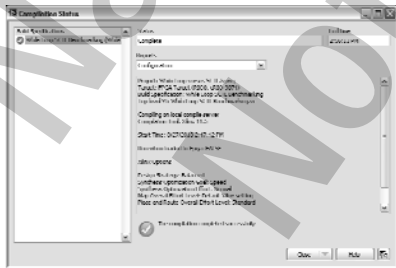


NATIONAL
INSTRUMENTS

ni.com/training

Configuration Report

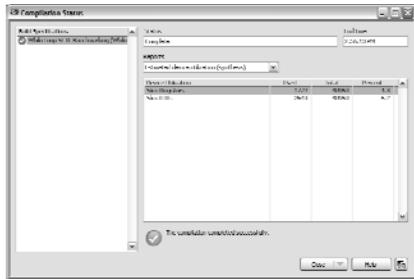
- Project information
- Xilinx compiler configuration for the FPGA VI



ni.com/training

Device Utilization Report

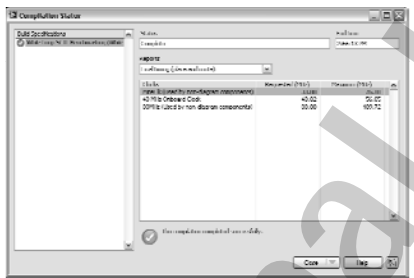
Separate reports generated at pre-synthesis and synthesis stages of compilation



ni.com/training

Timing Report

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



ni.com/training

Xilinx Log Report

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



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 hadn't tested on the development computer and the VI did not function as intended?
- What if your compilation fails?

DISCUSSION

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
- Device utilization
 - The design requires more resources than are available on the FPGA



ni.com/training

G. 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 primarily affect FPGA size



ni.com/training

Types of Basic Optimizations

- Limit front panel objects
- Use small data types
- Avoid large functions



ni.com/training

Limit Front Panel Objects

- 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

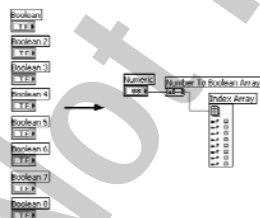
- Avoid using arrays on the Front Panel
 - All arrays must be of fixed size
 - Compile fails if more bytes in array than are available
 - Can quickly use large amounts of FPGA size
 - Each bit in the array uses its own flip-flop on the FPGA
- If only enough time to do 1 optimization, do this optimization



ni.com/training

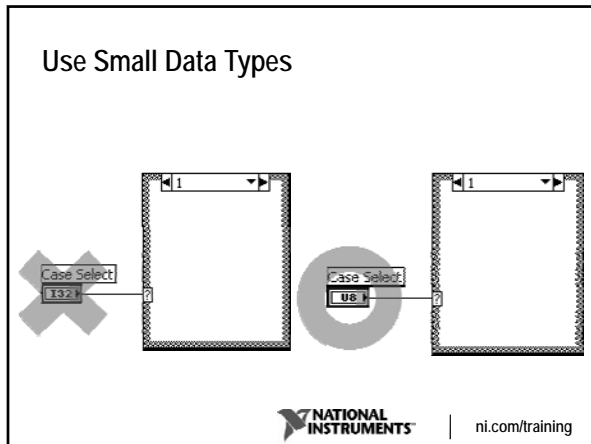
Bitpack Boolean Logic

- 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



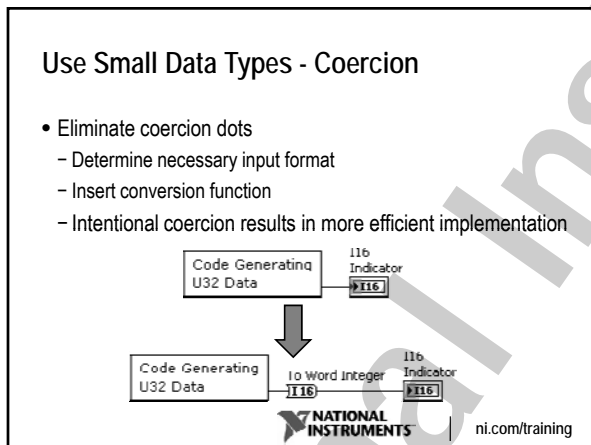
ni.com/training

Use Small Data Types



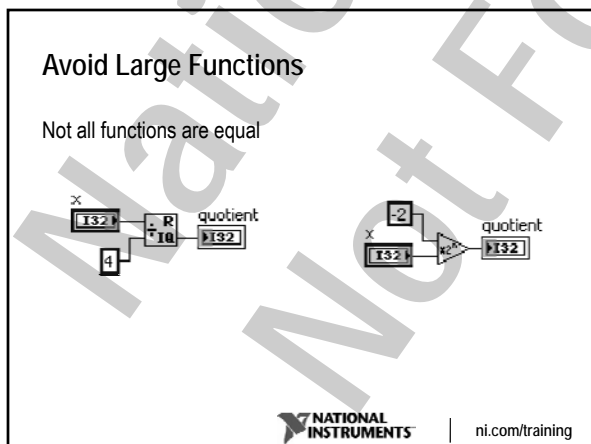
Use Small Data Types - Coercion

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



Avoid Large Functions

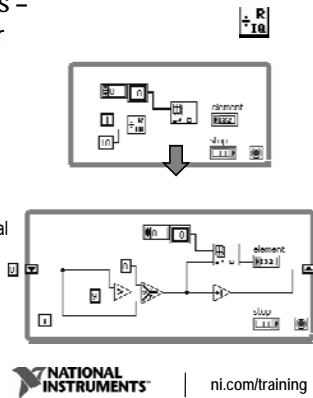
Not all functions are equal



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

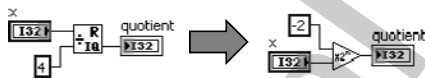


NATIONAL INSTRUMENTS

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



NATIONAL INSTRUMENTS

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

NATIONAL INSTRUMENTS

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 of your FPGA VI's functionality?
 - a. FPGA target
 - b. Development computer



ni.com/training

Summary—Quiz Answer

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



ni.com/training

Summary—Quiz

3. Which of the following is NOT the name of a report generated as part of the compilation process?
- Summary
 - Final Device Utilization (map)
 - Final Timing (place and route)
 - Optimization



ni.com/training

Summary—Quiz Answer

3. Which of the following is NOT the name of a report generated as part of the compilation process?
- Summary
 - Final Device Utilization (map)
 - Final Timing (place and route)
 - Optimization



ni.com/training

Summary—Quiz

4. Which of the following does NOT describe a VI that is developed under the FPGA target and set to execute on the FPGA?
- The Front Panel of the FPGA VI executes on the FPGA target
 - Non-sequential code in an FPGA VI executes in parallel
 - FPGA VI executes independently of the host
 - Only one top-level VI can run on the FPGA at a time



ni.com/training

Summary—Quiz Answer

4. Which of the following does NOT describe a VI that is developed under the FPGA target and set to execute on the FPGA?
- a. The Front Panel of the FGPA VI executes on the FPGA target
 - b. Non-sequential code in the FPGA VI executes in parallel
 - c. The FPGA VI executes independently of the host
 - d. Only one top-level VI can run on the FPGA at a time.



ni.com/training

National Instruments
Not For Distribution

Lesson 4 FPGA I/O

TOPICS

- A. Introduction
- B. Configuring FPGA I/O
- C. I/O Types
- D. Integer Math

- E. Fixed-Point Math
- F. CompactRIO
- G. Error Handling

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

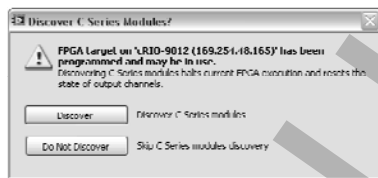
Adding FPGA I/O to the LabVIEW project

- 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



[nvidia.com/training](https://www.nvidia.com/training)

Adding FPGA I/O to a CompactRIO Project



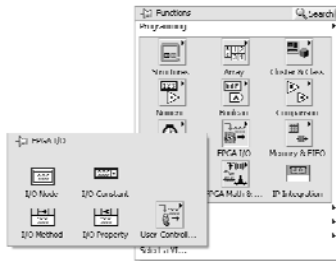
ni.com/training

Adding FPGA I/O to an R Series Project



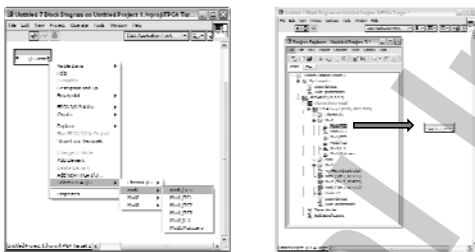
ni.com/training

FPGA I/O Palettes



ni.com/training

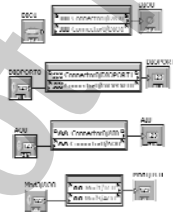
Placing I/O Nodes



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



ni.com/training

Digital I/O



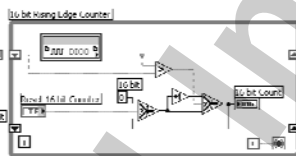
- Individual Lines – Boolean data type
- Ports (collection of lines) – Integer data type (1 bit per line)
- Depending on your hardware, digital I/O can be unidirectional or bidirectional
 - If the digital I/O line or port is bidirectional and you want to change direction of the line or the port from 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 are ordinarily not performed in the FPGA
 - Use the host for floating-point analysis instead



ni.com/training

D. Integer Math

- Analog Input
 - Converts binary representation of the voltage to a signed integer
 - 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

- Calculations can be performed with integers and fixed-point numbers


ni.com/training

Integer Division

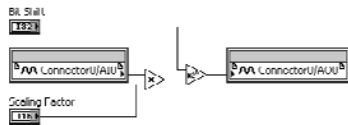
Divide function cannot return an integer value, so you must use other functions for integer math.

- 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 AIO by 0.7 we could use:

Scaling Factor: 11500

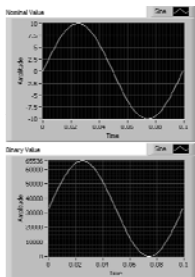
Bit Shift: -14 bits

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 occurs in the Host VI because FPGA does not support floating-point math



ni.com/training

Converting Binary Representations (cont.)

- Conversion is hardware dependent
- Hardware-specific factors used in conversion
 - Voltage Range
 - ADC Resolution
 - Offset
 - 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 value of renaming your I/O resources?
- What is the value of the I/O node returning data from the digital port as a U8?

DISCUSSION

E. Fixed-Point Math

- Greatly simplifies arithmetic on the FPGA
 - Simplifies computations
 - Size and speed advantages of integer math
- 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

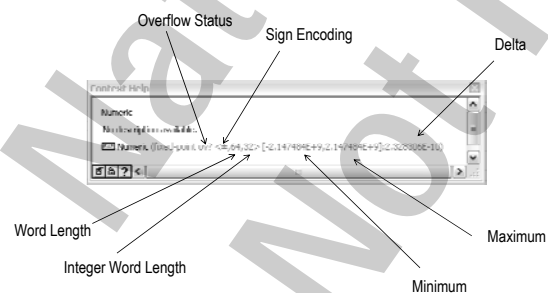
Sign Encoding – The option that determines whether the fixed-point data is signed (\pm) or unsigned (+)

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 Context Help


ni.com/training

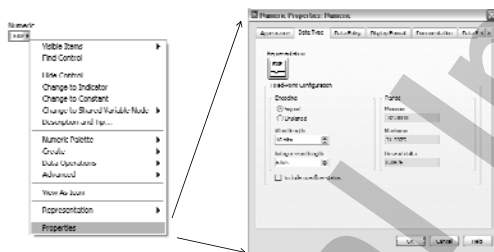
Numeric Representation Examples

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



ni.com/training

Fixed Point Configuration



ni.com/training

Fixed-Point Arithmetic

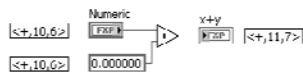
- Range of output values will be large enough to accommodate largest possible input values, up to 64 bits
- LabVIEW propagates the range of values along each wire. The representation is then calculated to be as small as possible without losing data.
- We will examine the following operations and provide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



ni.com/training

Fixed-Point Addition

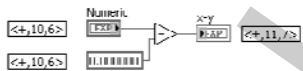
- Inputs should have the same range of values
 - If inputs have different ranges, then one or both will be coerced to a range that accommodates both
- For the output, the word length and integer word length will each increase by one and the sign encoding will match the input sign encoding
 - $\langle \pm, A, B \rangle + \langle \pm, A, B \rangle = \langle \pm, A+1, B+1 \rangle$



ni.com/training

Fixed Point Subtraction

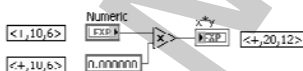
- The output representation follows the same guidelines as addition
- The output will be signed, regardless of whether or not the inputs were signed or unsigned.
 - $\langle +, A, B \rangle - \langle +, A, B \rangle = \langle \pm, A+1, B+1 \rangle$



ni.com/training

Fixed-Point Multiplication

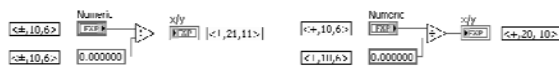
- Inputs can have different ranges of values
- For the output, the word lengths and integer word lengths are each added together
- If either input is signed, then the output will be signed. Otherwise, the output is unsigned.
 - $\langle +, A, B \rangle * \langle +, C, D \rangle = \langle +, A+C, B+D \rangle$



ni.com/training

Fixed-Point Division

- Inputs can have different ranges of values
 - Signed and unsigned division result in different ranges of values for the output.
 - Signed: $\langle \pm, A, B \rangle / \langle \pm, C, D \rangle = \langle \pm, A+C+1, B+C-D+1 \rangle$
 - Unsigned: $\langle +, A, B \rangle / \langle +, C, D \rangle = \langle +, A+C, B+C-D \rangle$


ni.com/training

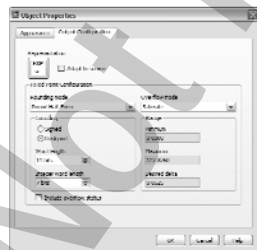
Demonstration – Simple Fixed-Point Math

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

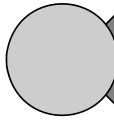
DEMONSTRATION

Configuring Fixed Point Functions

- Adapt to Source
- Rounding Mode
- Overflow Mode


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.

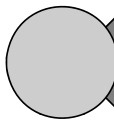
Rounding Mode

- 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

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 Centigrade, where should that conversion take place?
- What was the data type of Thermocouple Difference? Does the data type match what you expected it to be based on the inputs?

DEMONSTRATION

F. CompactRIO

Different modules can have different

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

Refer to the *LabVIEW Help* for module-specific information.



ni.com/training

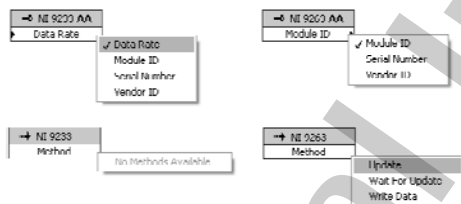
Module Data Types

- 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

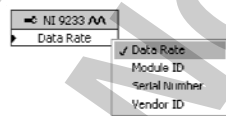
Different modules have different properties and methods. These are two examples:

ni.com/training

Module Timing

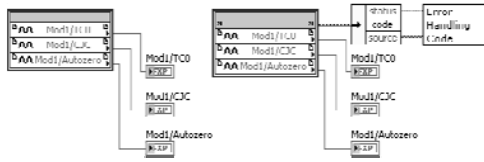
Timing is module-specific

- NI 9211 (Thermocouple) – single samples
- NI 9233 (Analog Input) – multiple samples at user specified rate using the data rate property

ni.com/training

G. Error Handling

- 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

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

- | | |
|-----------------|--|
| 1. Terminal | a. A name assigned by the developer to a particular I/O resource |
| 2. I/O Resource | b. A hardware connection, such as on a CompactRIO module |
| 3. I/O Name | c. A logical representation in LabVIEW FPGA of a terminal |



ni.com/training

Summary—Quiz Answers

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

- | | |
|-----------------|--|
| 1. Terminal | a. A name assigned by the developer to a particular I/O resource |
| 2. I/O Resource | b. A hardware connection, such as on a CompactRIO module |
| 3. I/O Name | c. A logical representation in LabVIEW FPGA of a terminal |



ni.com/training

Summary—Quiz

4. What is the default data type of the result of adding two signed fixed-point numbers with a word length of 20 and an integer word length of 10?

- <±,40,20>
- <+,20,10>
- <±,21,11>
- <+,30,30>



ni.com/training

Summary—Quiz Answer

4. What is the default data type of the result of adding two signed fixed-point numbers with a word length of 20 and an integer word length of 10?
- a. $\langle \pm, 40, 20 \rangle$
 - b. $\langle +, 20, 10 \rangle$
 - c. $\langle \pm, 21, 11 \rangle$
 - d. $\langle +, 30, 30 \rangle$



ni.com/training

Summary—Quiz

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



ni.com/training

Summary—Quiz Answer

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



ni.com/training

Summary—Quiz

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

- 6. NI PCI-7831R R Series board a. Integer
- 7. NI 9233 CompactRIO module b. Fixed-point



ni.com/training

Summary—Quiz Answer

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

- 6. NI PCI-7831R R Series board → a. Integer
- 7. NI 9233 CompactRIO module → b. Fixed-point




ni.com/training

Lesson 5 Timing an FPGA VI

TOPICS

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

 | ni.com/training

A. Timing Express VIs

Loop Timer Express VI

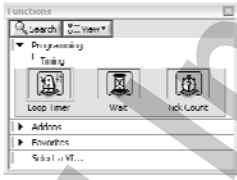
- Implementing Loop Rates


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

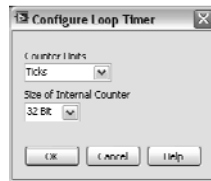
- Waits the value you specify in Count between loop iterations
- Controls the period of a loop
- Place the Loop Timer Express VI inside a loop
 - Perform I/O operations at a specific frequency
 - Control loop execution rate



 | ni.com/training

Configure Loop Timer

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



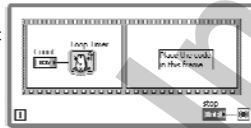
Size of Internal Counter determines the maximum time a timer can track. To save space on the FPGA, use the smallest Size of Internal Counter possible.



ni.com/training

Loop Timer Express VI Functionality

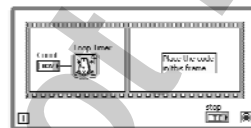
- First iteration
 - Records current time as initial time
 - Immediately executes the code in the next frame without any delay
- Second iteration
 - Adds Count to initial time and waits until Count has elapsed from the initial recorded time
 - Executes the code in the next frame after Count has elapsed
- 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 40MHz FPGA clock, the tick period is 25ns
- 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

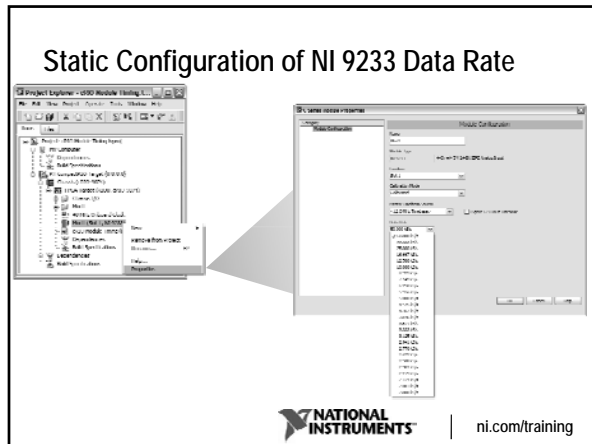
cRIO Module Loop Execution Rates

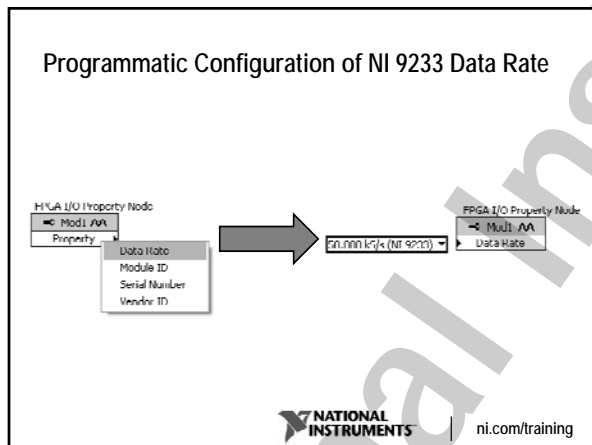
Some cRIO modules have configurable timing

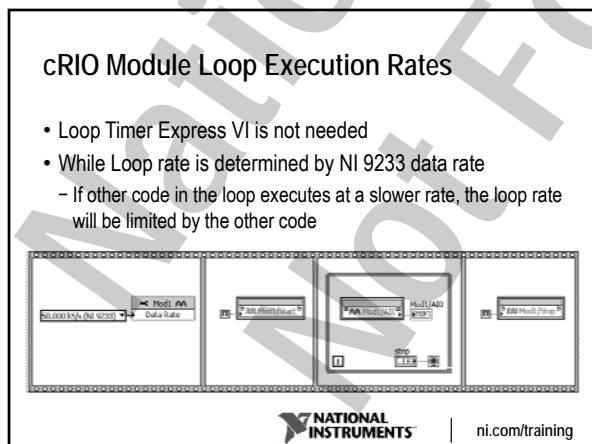
- Do not need to use Loop Timer Express VI to time loop rate
- Example
 - NI 9233
 - You can configure the data rate at which the NI 9233 module acquires and returns data
 - Static/Programmatic Configuration
 - C-Series Module Properties dialog box
 - FPGA I/O Property Node



ni.com/training

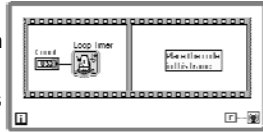






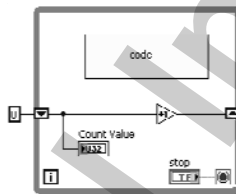
Loop Conditional Terminal

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

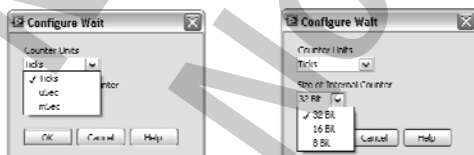


NATIONAL
INSTRUMENTS

ni.com/training

Wait Express VI Configuration

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

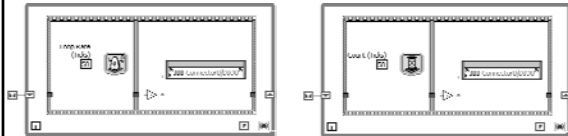


NATIONAL
INSTRUMENTS

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



ni.com/training

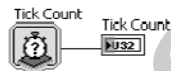
D. Measuring Time Between Events

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

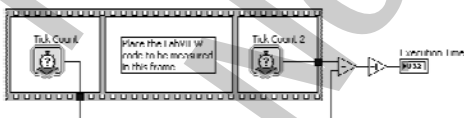
- Examples
 - Measure the time between edges on a digital signal
 - Determine the execution time of a section of code



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

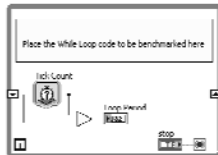


ni.com/training

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



ni.com/training

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	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

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 μ s 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

- | | |
|--------------------------|--|
| 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 |



ni.com/training

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 |



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

Data Sharing on FPGA

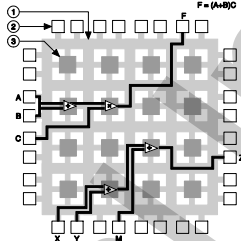
TOPICS


A. Parallel Loop Execution	E. Race Conditions
B. Shared Resources	F. FPGA FIFOs
C. Variables	G. Comparison of Data Sharing Methods
D. Memory Items	

 | ni.com/training

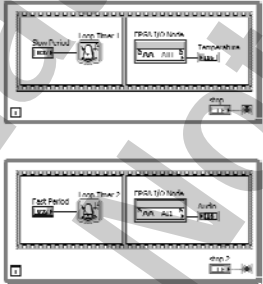
A. Parallel Loop Execution


- Graphical programming promotes parallel code architectures
- LabVIEW FPGA implements true parallel execution



 | ni.com/training

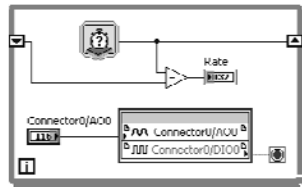
Parallel Loops with Different Sampling Rates



 | ni.com/training

Loop Rate Limitation – Longest Path

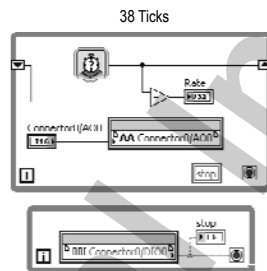
- AO takes ~35 ticks, DI takes 1 tick (HW Specific)
- DI limited by AO when in same loop



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



ni.com/training

Sharing Data Between Parallel Loops

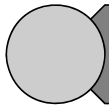
What if you want to pass data from one loop to another?

- 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.com/training

B. Shared Resources



Shared Resource – Any resource that is accessed by multiple objects in an FPGA VI

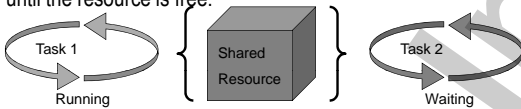
- Digital outputs on most targets
- Analog inputs on most targets
- Analog output on most targets
- Digital inputs on some targets
- Memories
- FIFOs
- Non-reentrant subVIs
- Local and global variables



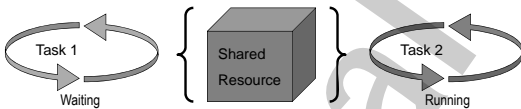
ni.com/training

B. Shared Resources

Before a task can begin using a shared resource, it must wait until the resource is free.

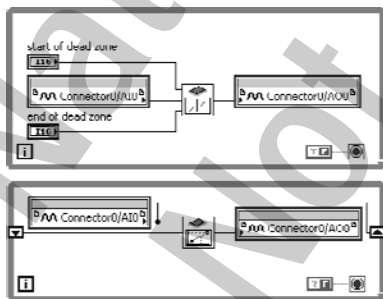


After Task 1 finishes, Task 2 can proceed.



ni.com/training

Shared Resources Example



ni.com/training

C. 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

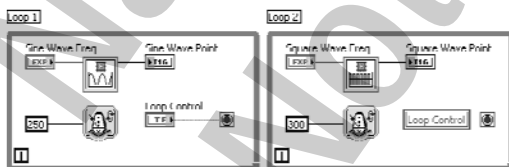
- Store data in front panel controls and indicators
- Accessible only by a single VI
- Use to maintain data separately for separate re-entrant subVIs
- Create by right-clicking on a control or indicator on the block diagram and selecting Create»Local Variable
- If you use this approach, you may find 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

D. 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

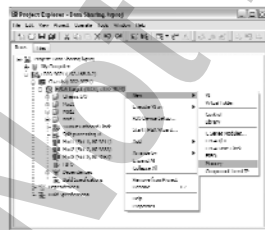
Memory Item Usage

- 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-Scoped
 - VI-Defined

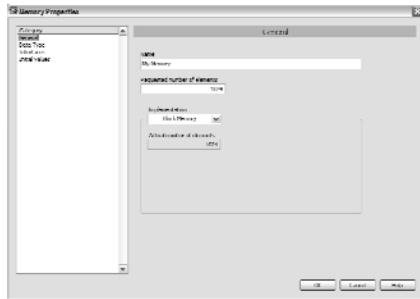

ni.com/training

Target-Scoped 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

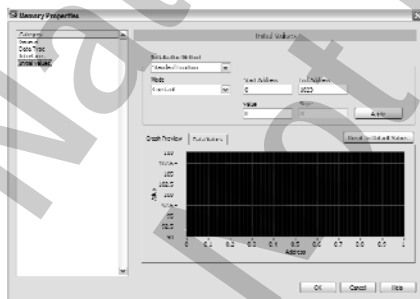
Use look-up tables when:

- 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

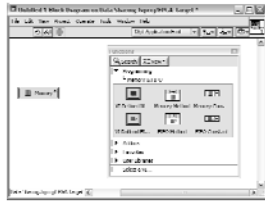
Memory Properties Dialog Box – Initial Values Page



ni.com/training

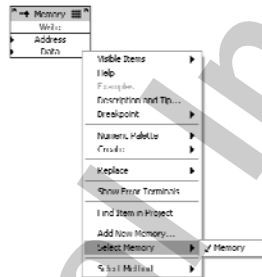
VI-Defined Memory Item

- 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
 - Double-click on node to launch Memory Properties Dialog Box


ni.com/training

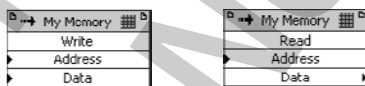
Memory Write and Read

- 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


ni.com/training

Memory Write and Read

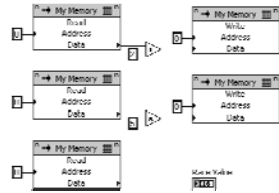
- Address – The location of the data in memory on the FPGA target
- Data – The data to be written to or read from the memory on the FPGA target
- Add error terminals to perform error handling


ni.com/training

E. Race Conditions

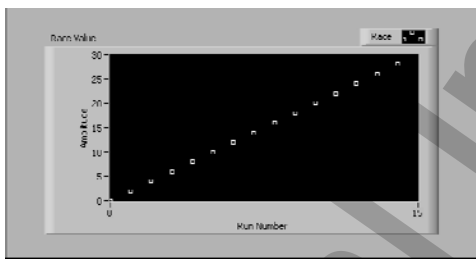
Race conditions can result from accessing a shared resource at multiple locations.

- What is the value of the Race Value after executing this VI the first time?
- After the second?
- After the third?



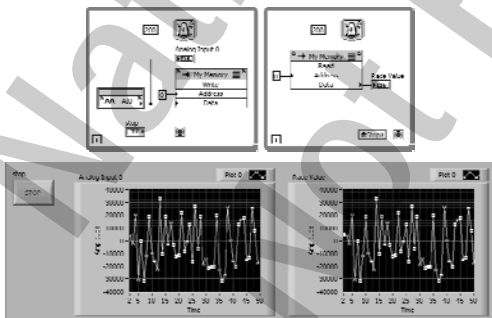
ni.com/training

Race Conditions – Possible Result



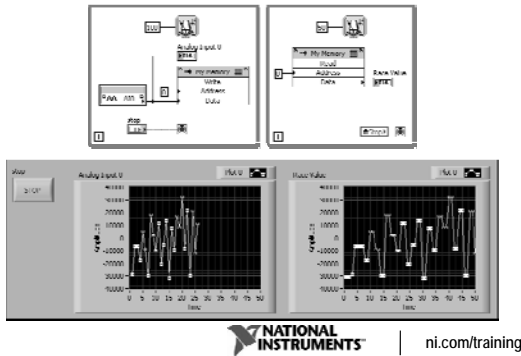
ni.com/training

Parallel Loops – Write and Read at Same Rate

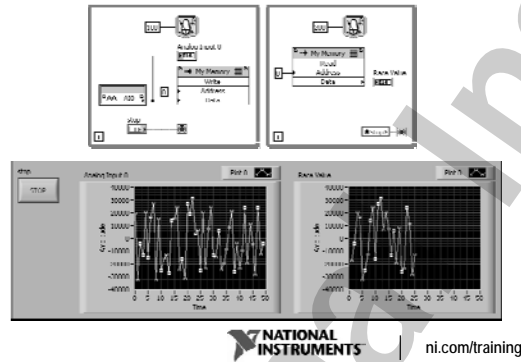


ni.com/training

Parallel Loops – Oversampling



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

F. FPGA FIFOs



FIFO – First-in first-out buffer. The first data item written to memory is the first item read and removed from memory.

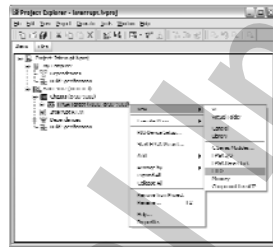
- Target-Scoped – A single FIFO transfers data between loops on multiple VIs running on the same target
- VI-Scoped – A single FIFO transfers data between multiple loops in the same VI



ni.com/training

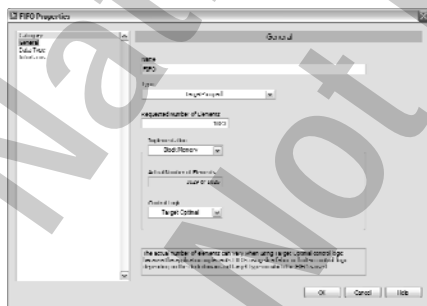
Target-Scoped FIFOs

- 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



ni.com/training

FIFO Properties Dialog Box



ni.com/training

FIFO – Number of Elements

- Number of elements FIFO can hold
- 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.


ni.com/training

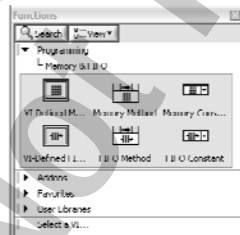
FIFO – Implementation

- **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 (2 per slice). Recommended only for small FIFOs < 300 bytes.
- **Block Memory** – Store data in block memory to preserve FPGA gates and LUTs for your VI.


ni.com/training

VI-Scoped FIFOs

- Create VI-Scoped 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

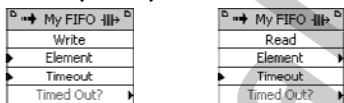
FPGA FIFO Write and Read

- Place a FIFO Method from the Memory & FIFO palette
- Right-click the object and choose **Select Method**
 - Choose Write or Read
- Right-click the object and choose **Select FIFO**
 - Associates node with a defined FIFO


ni.com/training

FPGA FIFO Write and Read

- Element** – The data element to written or read
- Timeout** – Sets number of ticks the function waits for space if the FIFO is full.
- Timed Out?** – True if attempt to write failed. Does not overwrite or add new element.
 - FIFO transfer may be lossy if write times out


ni.com/training

FIFO Overflow and Underflow

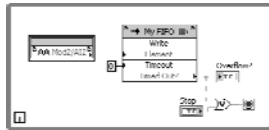
- Overflow** – Write loop executing faster than read loop
 - FIFO is filled and the FIFO Write method times out
 - Data is not written to the FIFO until space is available in the FIFO
 - Space can be created by reading data or resetting the FIFO
 - Data is lost until space is made available.
- Underflow** – Write loop executing slower than read loop
 - FIFO is empty and the FIFO Read method times out


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



ni.com/training

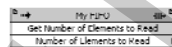
Demonstration – FIFO Bucket

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



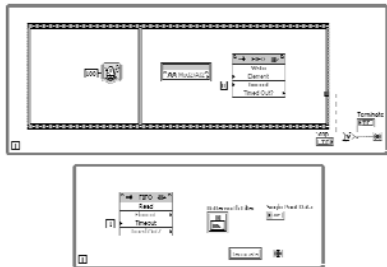
These methods can be used to avoid underflow/overflow conditions by monitoring the status of the FIFO.



ni.com/training

Separate I/O and Data Processing

- Top loop acquires data and writes to FIFO
- Bottom loop reads FIFO and processes data
- Results in faster I/O



ni.com/training

Clear Method



- FIFOs retain data if the FPGA is stopped and restarted.
- Use the Clear Method to empty a target- or VI-scoped FIFO
 - Place FIFO Method node
 - Right-click on object and choose Select Method»Clear



ni.com/training

Exercise 6-1 Accelerometer Threshold

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

GOAL

Exercise 6-1 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

G. Comparison of Data Sharing Methods

Transfer Method	FPGA Resource	Lossy?	Common Use
Variables	Logic	Yes	Control, Simulation
Memory Items	Memory	Yes	Control, Simulation, Variable sized data
Flip-Flop FIFOs	Logic	No	Datalogging (FIFOs smaller than 100 bytes)
Look-Up Table FIFOs	Logic	No	Datalogging (FIFOs smaller than 300 bytes)
Block Memory FIFOs	Memory and Logic	No	Datalogging (FIFOs 300 bytes or larger)



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. FIFO
 - d. 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. FIFO
 - d. 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 produces 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 produces 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 smaller than 300 bytes |
| 2. Flip-Flops | b. For FIFOs larger than 300 bytes |
| 3. Look-Up Table | c. For FIFOs smaller than 100 bytes |



ni.com/training

Summary—Quiz

Match each method of FIFO implementation with its recommended usage.

- | | |
|------------------|-------------------------------------|
| 1. Block Memory | a. For FIFOs smaller than 300 bytes |
| 2. Flip-Flops | b. For FIFOs larger than 300 bytes |
| 3. Look-Up Table | c. For FIFOs smaller than 100 bytes |




ni.com/training

Lesson 7

Single-Cycle Timed Loops

TOPICS

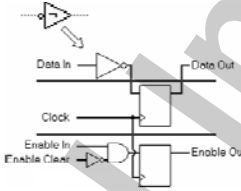
- A. Dataflow in FPGA
- B. Single-Cycle Timed Loop
- C. FPGA Clocks
- D. SCTL Errors



ni.com/training

A. Dataflow in FPGA

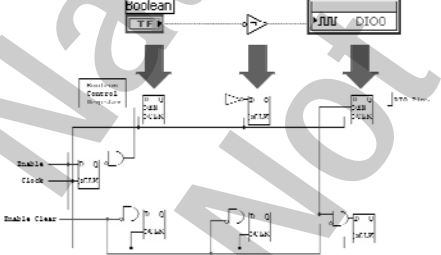
Three components necessary to maintain data flow:


- The corresponding logic function
- Synchronization
- The enable chain




ni.com/training

Dataflow in FPGA




ni.com/training

Dataflow in FPGA

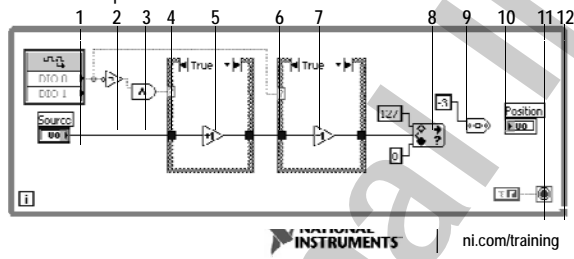
- Each function or VI takes a minimum of 1 clock cycle
- Functions can run in parallel
- Some dependent functions must run in sequence
- Application can only run as quickly as the sum of items in a sequence
- While Loops have a 2 clock cycle overhead



ni.com/training

Execution of a While Loop

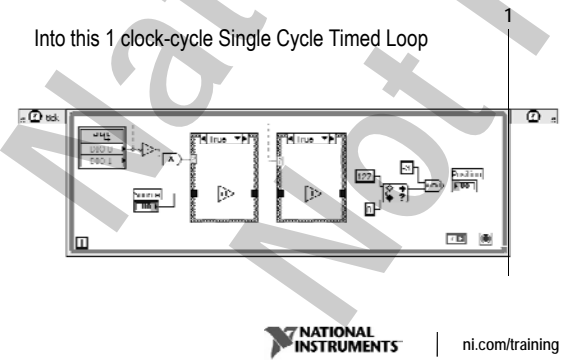
Use a single-cycle Timed Loop to convert this 12 clock-cycle While Loop



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

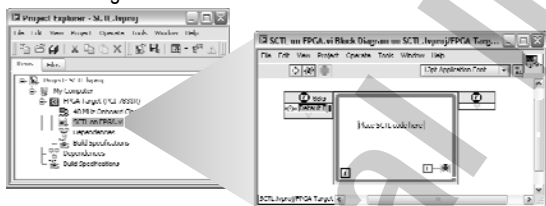


ni.com/training

B. Single-Cycle Timed Loop

Single-cycle Timed Loop

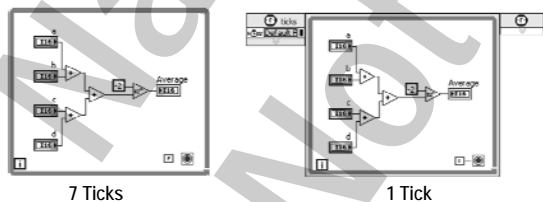
- Create by placing a Timed Loop on the block diagram of an FPGA target VI



ni.com/training

B. 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 Application Development* course



ni.com/training

Exercise 7-1: While Loop versus Single-Cycle Timed Loop

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

GOAL

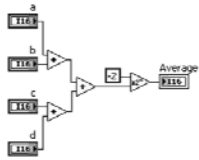
Exercise 7-1: While Loop versus Single-Cycle Timed Loop

- 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 = $1 / \text{clock frequency}$
- 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

C. 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

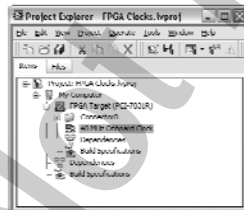


ni.com/training

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

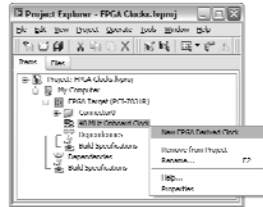


ni.com/training

Using FPGA Clocks – Derived Clock

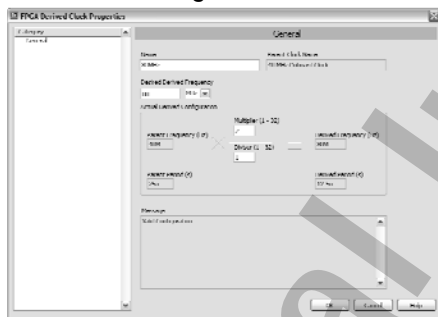
FPGA Derived Clock

- 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



[nvidia.com/training](https://www.nvidia.com/training)

Derived Clock Configuration

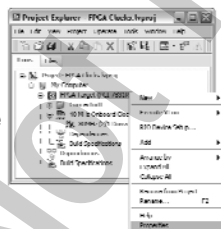


ni.com/training

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. SCTL Errors – Unsupported Objects

- 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

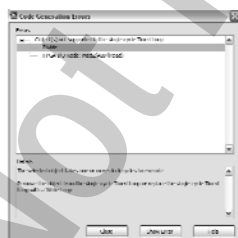


ni.com/training

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

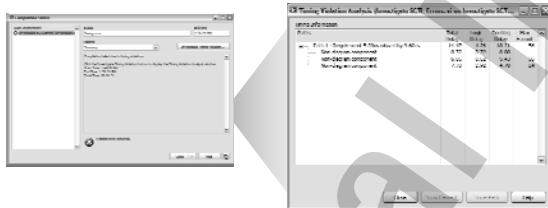


ni.com/training

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



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

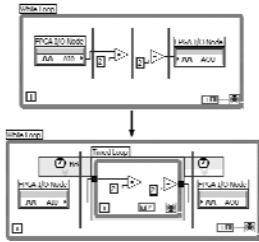


ni.com/training

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 7-2: Fixing SCTL Errors

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

GOAL

Exercise 7-2: Fixing SCTL Errors

- Why is the NI 9233 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

Summary—Matching Quiz

- | | |
|-------------------------|--|
| 1. FPGA base clock | A. Clock that an FPGA VI uses outside of SCTLs |
| 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 SCTLs |
| 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?
- Optimize the code in the SCTL
 - Move some code out of the SCTL
 - Select a faster clock for the SCTL timing source
 - Select a slower clock for the SCTL timing source

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?
- Optimize the code in the SCTL
 - Move some code out of the SCTL
 - Select a faster clock for the SCTL timing source
 - Select a slower clock for the SCTL timing source


ni.com/training

Lesson 8

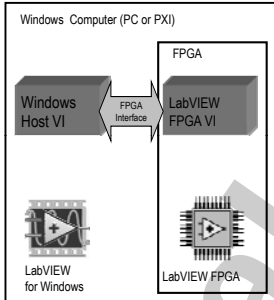
Basic Host Integration – PC/Real-Time


TOPICS

A. Windows Host Integration	F. Developing a Windows VI
B. Developing a Windows Host VI	G. Shared Variable Network Communication
C. Introduction to Real-Time	H. Prepare RT Host for Final Application
D. Developing an RT Host VI	
E. Front Panel Communication	

 | ni.com/training

A. Windows Host Integration




 | ni.com/training

A. Windows Host Integration

Common Windows host VI tasks:

- Transfer data between Windows computer and FPGA
- User interface
- Do more data processing than you can fit on the FPGA
- Perform operations not available on the FPGA target
- Floating-point math
- Sequence multiple FPGA VIs
- Log data
- 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

 | ni.com/training

B. Developing a Windows Host VI

Add a VI under My Computer



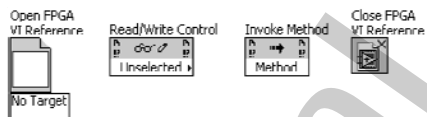
Note: The contents of right-click menus differ from those in this image depending on the software installed on your computer.



[nvidia.com/training](https://www.nvidia.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



ni.com/training

Open FPGA VI Reference

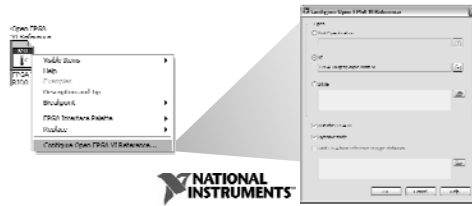
- Opens a reference to:
 - the FPGA build specification, FPGA VI, or bitfile
 - and FPGA target
- Must open a reference to establish communication between the host and the FPGA
- FPGA target may be specified
 - Explicitly, in all cases, by wiring the reference name input or
 - Implicitly, when opening a reference to an FPGA build specification or an FPGA VI



ni.com/training

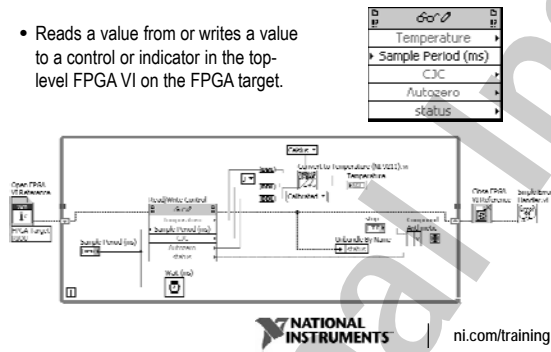
Open FPGA VI Reference Configuration

- Select build specification, VI, or bitfile
- Specify whether to run the FPGA VI if it is not already running when you run the host VI
- Enable/Disable Dynamic mode. When you enable Dynamic mode, you can write reusable subVIs based on the FPGA VI reference.



Read/Write Control

- Reads a value from or writes a value to a control or indicator in the top-level FPGA VI on the FPGA target.



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
 - Configure
 - Read
 - Start
 - Stop
 - Write
- Your target may support additional target-specific methods



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

Developing a Windows Host VI

- FPGA host interface functions on the Windows 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 8-1: Windows Host Integration

Create a user interface on a Windows host system that interacts with an FPGA VI on a simulated PCI-7831R R-Series board.

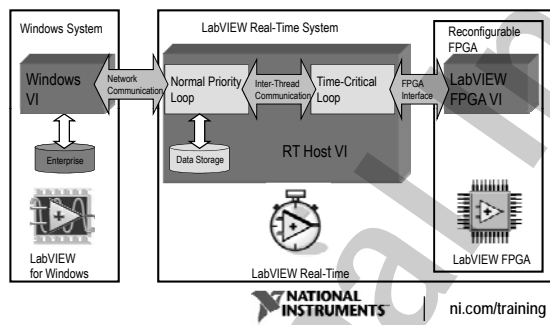
GOAL

Exercise 8-1: Windows Host Integration

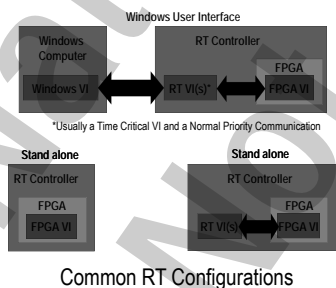
- What functionality does the Windows host VI accomplish that the FPGA VI could not?
- Will the same scaling calculations work for I/O channels that have a different resolution and voltage range?

DISCUSSION

LabVIEW FPGA Architecture with LabVIEW Real-Time: RT Host VI



C. Introduction to Real-Time



Introduction to Real-Time

Advantages of Using a Real-time Host

- Code execution is more deterministic
- 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

The LabVIEW Real-Time 1 Course

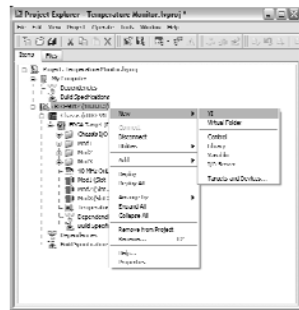
- | | |
|---------------------------------|--------------------------------------|
| • Configuring CompactRIO system | • Event response |
| • Real-time application design | • Shared variables with RT FIFO |
| • Multithreading | • Network-published shared variables |
| • Passing data between threads | • Verifying timing and memory usage |
| • Improving determinism | • Execution Trace Tool Kit |
| • Priority levels | • Deploying an application |
| • Memory management | • Remote panels |
| • Timed structures | |



ni.com/training

D. Developing an RT Host VI

Add a VI under the RT target



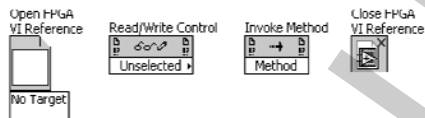
**NATIONAL
INSTRUMENTS**

[nvidia.com/training](https://www.nvidia.com/training)

Developing an RT Host VI

FPGA Interface Functions

- Used the same way whether programming a Windows Host VI or RT Host VI



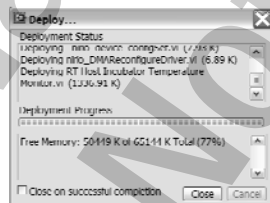
**NATIONAL
INSTRUMENTS**

ni.com/training

Developing an RT Host VI

After developing the RT host VI, run it

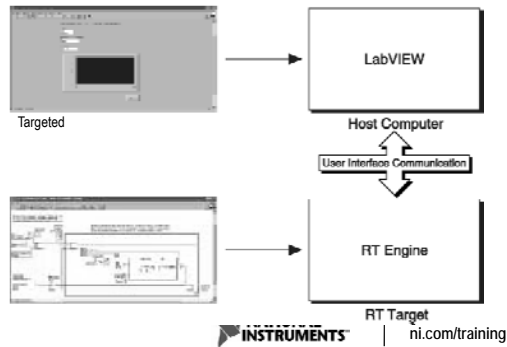
- Deploying Status Window:



**NATIONAL
INSTRUMENTS**

ni.com/training

E. Front Panel Communication in LabVIEW Real-Time



Front Panel Communication in LabVIEW Real-Time

Advantages

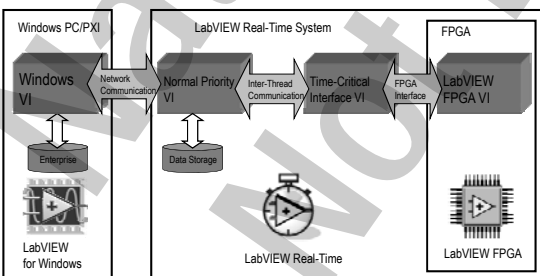
- Debug VIs
- Quickly monitor VIs running on RT target during development

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

NATIONAL INSTRUMENTS | ni.com/training

LabVIEW FPGA Architecture with LabVIEW Real-Time: Windows VI



NATIONAL INSTRUMENTS | ni.com/training

F. Developing a Windows VI

Windows VI possible tasks:

- Act as a user-interface
- Send data and commands to RT host VI
- ActiveX and .NET 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
 - Control the data exchanged
 - which front panel objects on the PC get updated and when
 - which RT target VI components are visible on the front panel of the Windows VI
 - Control timing and sequencing of the data transfer
 - Perform additional data processing or logging



ni.com/training

G. 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

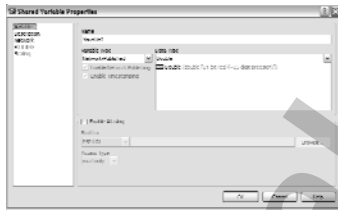
- Create Shared Variables from the Project Explorer window
- Shared Variables must be inside project libraries
- LabVIEW automatically creates the library with the shared variable inside



ni.com/training

Shared Variable Configuration

- Configuring Network-published shared variables
 - Set Variable Type to Network-published

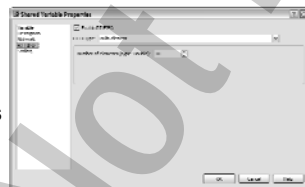


ni.com/training

Shared Variable with RT FIFO

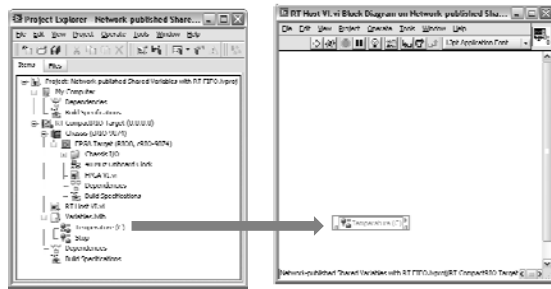
RT FIFO category

- Enable RT FIFO
 - Shares data across a network without affecting the determinism of the VIs
- FIFO Type
 - Single Element
 - Multi-element
 - Number of elements



ni.com/training

Shared Variable Programming



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
- This is covered in the *LabVIEW Real-Time 1* course



ni.com/training

Exercise 8-2: 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 8-2: 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

H. 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* course



ni.com/training

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. 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. 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. You should use Interactive Front Panel Communication in your final RT application.
- True
 - False



ni.com/training

Summary—Quiz

4. You should use Interactive Front Panel Communication in your final RT application.
- True
 - False



ni.com/training

National Instruments
Not For Distribution

Lesson 9 DMA Data Transfers

TOPICS

- A. LabVIEW FPGA and Host Communication
- B. DMA FIFOs
- C. Lossless Data Transfer
- D. Interleaving



ni.com/training

A. LabVIEW FPGA and Host Communication

- FPGA VI and host VI are inherently asynchronous
 - Each VI runs independently of the other
- Data transfer synchronization needs to be implemented based on the application needs
 - Communication using front panel control and indicators is sufficient for applications that don't require tight synchronization for control or data processing
 - Data buffering is required for tight synchronization and transferring large amounts of data for processing



ni.com/training

Limitations of Front Panel Controls/Indicators

- Only the most current data is transferred
- Because FPGA VI and host VI are inherently asynchronous, data could be lost if one VI writes faster than the other reads
 - 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



ni.com/training

Data Buffering



Buffer – An area of computer memory that stores multiple data items

- To transfer larger amounts of data at high rates between the target and the host you must buffer your data
- The best way to buffer data is by using Direct Memory Access (DMA) data transfer methods



ni.com/training

B. DMA FIFOs



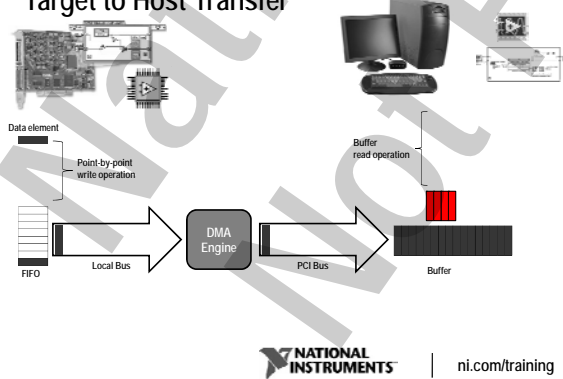
DMA – A single FIFO transfers data to or from Host VIs by directly accessing memory.

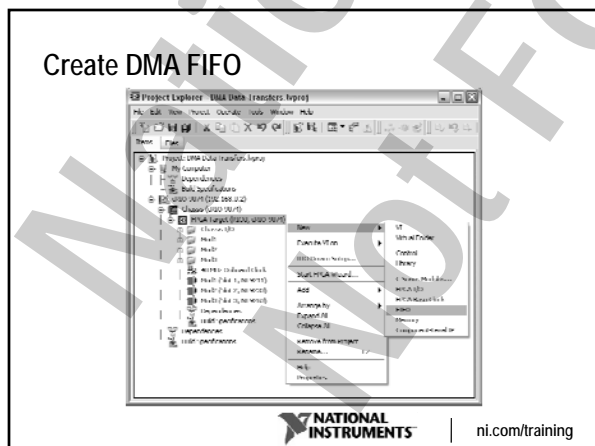
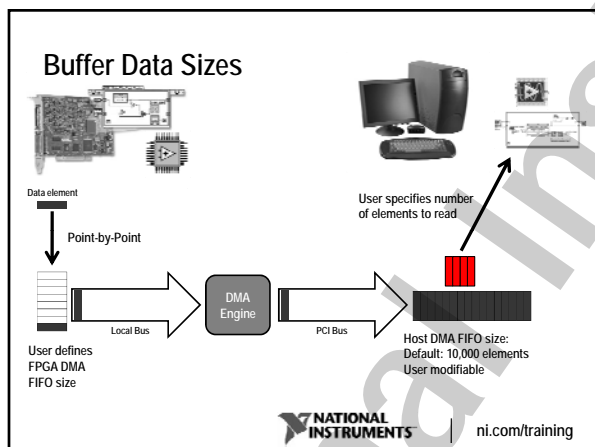
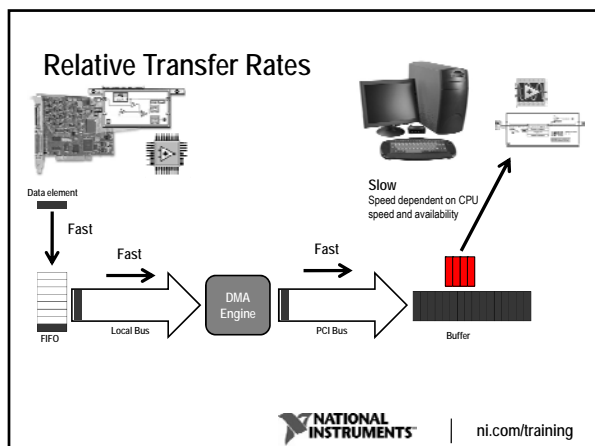
- Consists of two parts – FPGA and host
- Streams large amounts of data between computer memory and the FPGA
- Done mostly without the CPU
- Provides better performance than using the CPU to read and write data to the indicators and controls



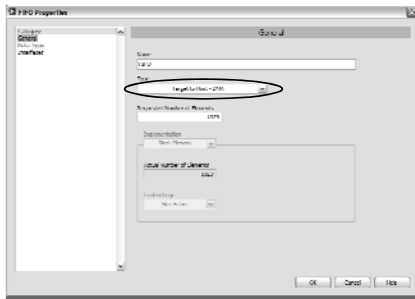
ni.com/training

Target to Host Transfer



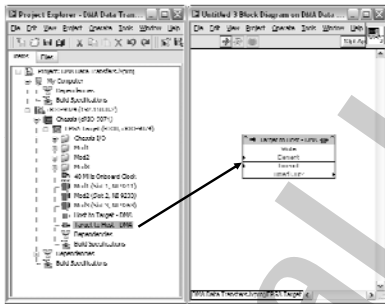


Configure DMA FIFO



ni.com/training

Use DMA FIFO in FPGA VI



ni.com/training

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



ni.com/training

Use DMA FIFO in Host VI

- Open a reference to the FPGA
- Add an Invoke Method function
- For a Host to Target FIFO
 - Choose X»Read to read data from FPGA
- For a Host to Target FIFO
 - Choose X»Write to write data to FPGA
- X is the name of the FIFO that you created in the Project Explorer



ni.com/training

Host VI: DMA FIFO Read and Write

Target to Host Transfer

Target to Host - DMA.Read
Number of Elements
Timeout (ms)
Data
Elements Remaining

Host to Target Transfer

Host to Target - DMA.Write
Data
Timeout (ms)
Empty Elements Remaining



ni.com/training

Target to Host Architectures



Blocking – Host DMA FIFO Read waits indefinitely to read a fixed number of elements from the FIFO



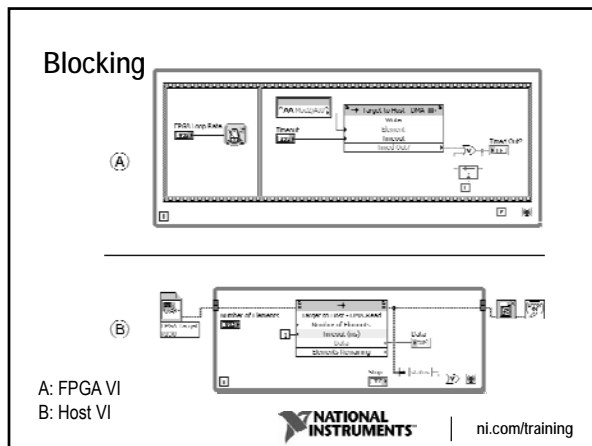
Polling – Host DMA FIFO Read reads all of the available elements in the FIFO at a user-defined rate.

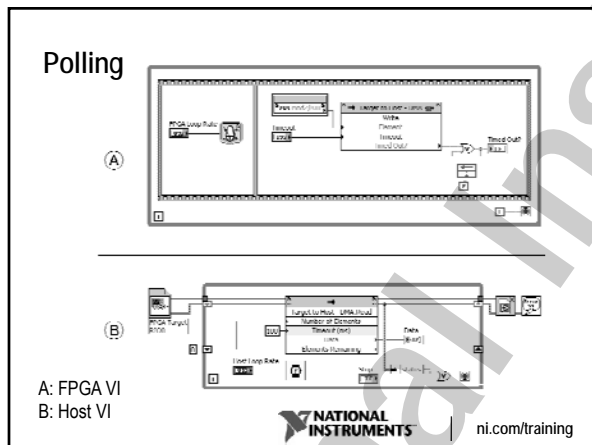


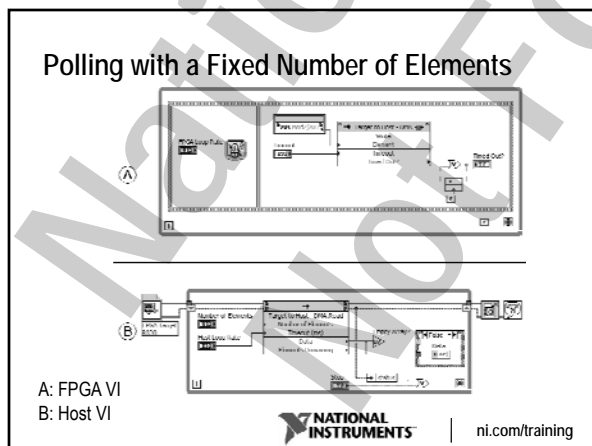
Polling with a Fixed Number of Elements – Host DMA FIFO Read reads a fixed number of elements at a user-defined rate.



ni.com/training







C. Lossless Data Transfer

- Selecting FIFO Size
 - FPGA VI
 - Host VI
- Ensuring lossless data transfer
 - Handling overflow
 - Handling underflow


ni.com/training

Selecting FIFO Size

- FPGA VI
 - In the FPGA FIFO Properties dialog box, use Number of Elements to set the size of the FPGA FIFO on the FPGA
 - Save resources by choosing the smallest size that is reasonable for your application
- Host VI
 - Default is 10,000 elements
 - Use a Configure Method to specify a different size


ni.com/training

Ensuring Lossless DMA Transfer

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/host buffer sizes
- Reduce load on CPU
 - Speed of CPU and competing tasks impact transfer rate from DMA to application memory


ni.com/training

Ensuring Lossless DMA Transfer

Underflow – FIFO is emptied as a result of data being read faster than it is written, timeout is generated

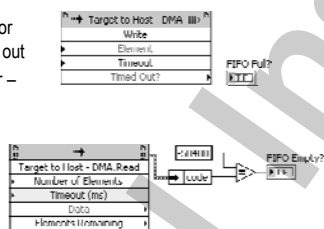
- Increase timeout
- Read less frequently
- Read smaller sets of elements



ni.com/training

Handling Overflow and Underflow

- Overflow – Check whether or not the Write Method timed out
- Underflow – Check for error – 50400 on the DMA Read method



ni.com/training

Recovering from a FIFO Overflow

- There are two primary methods for clearing the FIFO when an overflow occurs, depending on whether or not you want to keep the data that is in the FIFO
 - Reset the FPGA VI – Discards the data in the FIFO
 - Flushing the buffer – Reads the remaining data in the FIFO
- Either method can be performed to initialize the FIFO at the beginning of execution



ni.com/training

Resetting the FPGA VI

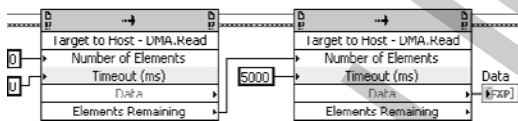
- Execution on development computer
 - FIFOs automatically reset when the VI stops and restarts
- Execution on FPGA Target
 - FIFOs do not automatically reset
- To reset programmatically:
 - Use the Invoke Method function
 - Or
 - Use the Close FPGA VI function configured to Close and Reset if Last Reference



ni.com/training

Flushing the Buffer

- Find out how much data remains in the FIFO
- Read all of the remaining elements from the FIFO



ni.com/training

Exercise 9-1 Custom Triggering

Develop an FPGA VI that waits for multiple trigger conditions to be met before acquiring analog data and writing to a DMA FIFO for transfer to the RT Host.

GOAL

Exercise 9-1 Custom Triggering

- What are the benefits of implementing a custom trigger on the FPGA?
- Which polling or blocking method was used to read the FIFO in the Host VI?
- How did your code handle overflow/underflow?

DISCUSSION

D. Interleaving

- There is a limited number of DMA FIFO channels available
 - Three channels available for NI 7831R 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

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



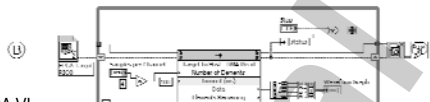
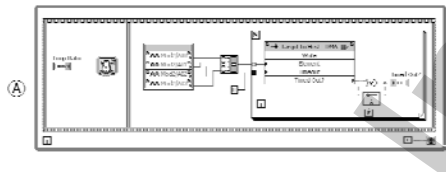
ni.com/training

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



Interleaving - Example



A: FPGA VI
B: Host VI



Exercise 9-2 AI Interleaved DMA

Create an FPGA VI that uses DMA FIFOs to transfer analog data from multiple devices and channels from the FPGA target to the RT Host using the blocking method of data transfer.

GOAL

Exercise 9-2 AI Interleaved DMA

- Which polling or blocking method was used to read the FIFO in the Host VI?
- What purpose does the Feedback Node serve in the FPGA VI?
- What is the largest Samples per Channel value that you could acquire before causing an overflow condition to occur?

DISCUSSION

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

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)
- Reduce the speed of acquisition on the FPGA VI
 - Increase the number of elements to read on the host
 - Decrease the rate at which the host reads data
 - 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)
- Reduce the speed of acquisition on the FPGA VI
 - Increase the number of elements to read on the host
 - Decrease the rate at which the host reads data
 - Increase the rate at which the host reads data


ni.com/training

Summary Quiz

Match each target to host architecture to the description that best fits it.

- | | |
|--|--|
| 1. Polling with Fixed Number of Elements | a. Host DMA FIFO Read reads a fixed number of elements at a user-defined rate |
| 2. Polling | b. Host DMA FIFO Read waits indefinitely to read a fixed number of elements from the FIFO |
| 3. Blocking | c. Host DMA FIFO Read reads all of the available elements in the FIFO at a user-defined rate |


ni.com/training

Summary Quiz Answer

Match each target to host architecture to the description that best fits it.

- | | | |
|--|------|---|
| 1. Polling with Fixed Number of Elements | → a. | Host DMA FIFO Read reads a fixed number of elements at a user-defined rate |
| 2. Polling | → b. | Host DMA FIFO Read waits indefinitely to read a fixed number of elements from the FIFO |
| 3. Blocking | → c. | Host DMA FIFO Read reads all of the available elements in the FIFO at a user-defined rate |




ni.com/training

Lesson 10

Modular Programming


TOPICS

A. Review of SubVIs	D. Using Name Controls and Constants
B. Using SubVIs on the FPGA	E. Testing FPGA SubVIs
C. Reentrancy and Non-reentrancy in FPGA	F. LabVIEW FPGA IPNet

 | 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

 | ni.com/training

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

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

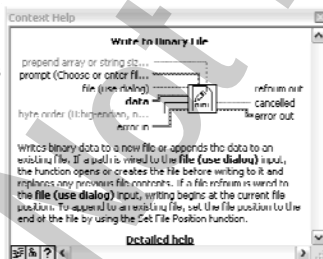
- Right-click the icon in the upper right corner of the front panel and select Show Connector
 - Each rectangle on the connector pane represents a terminal
 - Use the terminals to assign inputs and outputs
- Select a different pattern by right-clicking the connector pane and selecting Patterns from the shortcut menu



ni.com/training

Review - Using SubVIs – Terminal Setting

- Bold: Required terminal
- Plain: Recommended terminals
- Dimmed: Optional terminals

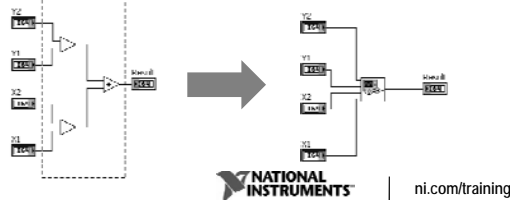


ni.com/training

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



B. Using SubVIs on the FPGA

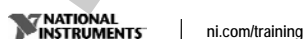
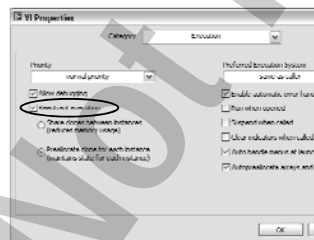
- You can run only one top-level FPGA VI
- You can use multiple VIs on the FPGA by placing subVIs on the block diagram of the top-level FPGA VI
- Placing large controls and indicators on the front panel of a subVI is acceptable
 - SubVI controls and indicators are not exposed to the host



C. Reentrancy and Non-reentrancy in FPGA

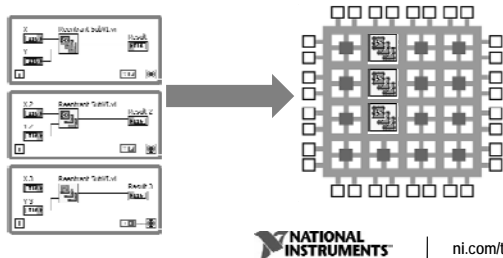
Reentrant FPGA SubVI Configuration

- Default configuration of VIs created under an FPGA target
- Multiple calls to the same subVI run in parallel using separate FPGA resources



Reentrant FPGA SubVI

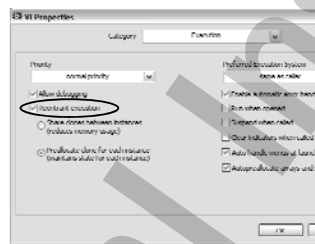
- Each instance of the reentrant FPGA subVI on the block diagram becomes a separate hardware resource



ni.com/training

Nonreentrant FPGA SubVI

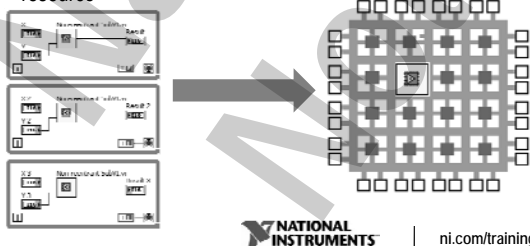
- Single instance shared among multiple callers
- Each call to the subVI waits until the previous call ends
- Does not allow parallel execution



ni.com/training

Nonreentrant FPGA SubVI

- Only a single copy of the nonreentrant FPGA subVI becomes hardware and all callers share the hardware resource



ni.com/training

Reentrancy and Non-reentrancy in FPGA

Tradeoffs

- Reentrant is the default VI Type for FPGA VIs

Comparison	Reentrant SubVI	Nonreentrant 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. Using Name Controls and Constants

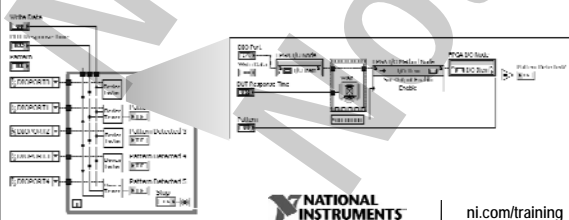
Using FPGA Name Controls in SubVIs

- FPGA I/O control
- FPGA Clock control
- Memory control
- FIFO control


ni.com/training

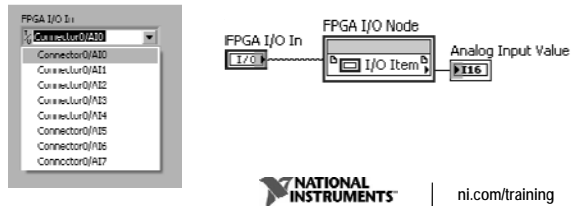
FPGA I/O Control

- Passes FPGA I/O items to subVIs
- Create VIs that can be used as reentrant subVIs with configurable I/O items


ni.com/training

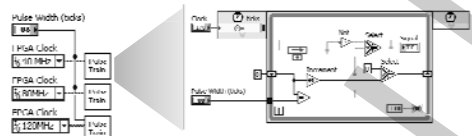
FPGA I/O Control

- 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



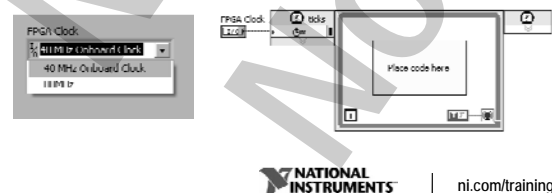
FPGA Clock Control

- Pass FPGA Clocks to subVIs



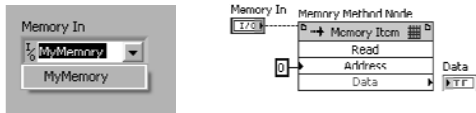
FPGA Clock Control

- 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 Control

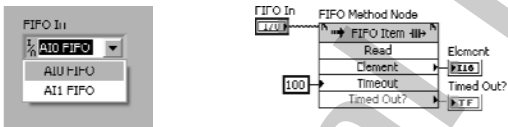
- Pass memory items to subVIs
- Right-click the Memory In input of a Memory Method node and select Create»Control
- Wire the memory control to the subVI connector pane



ni.com/training

FPGA FIFO Control

- Pass FPGA FIFO items to subVIs
- Right-click the FIFO In input of a FIFO Method node and select Create»Control
- Wire the FIFO control to the subVI connector pane



ni.com/training

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 controls into clusters with other FPGA controls, but not with other data types



ni.com/training

E. Testing FPGA SubVIs

- Test subVIs as top-level VIs
 - Execute on development computer
 - Execute on FPGA target



ni.com/training

F. LabVIEW FPGA IPNet

LabVIEW FPGA IPNet

- Resource for downloading and sharing LabVIEW FPGA functions and intellectual property
 - Acquire IP needed for your application
 - Download examples to learn programming techniques
 - Share your LabVIEW FPGA IP



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 nonreentrant?

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



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




ni.com/training

National Instruments
Not For Distribution

Appendix A: Pipelining

TOPICS

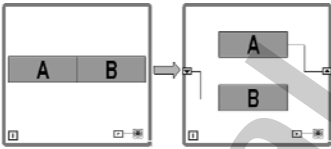

- A. Pipelining
- B. Using Pipelining in SCTLs



A. Pipelining

Within a loop, you can split your code into different loop iterations to reduce the duration of each iteration

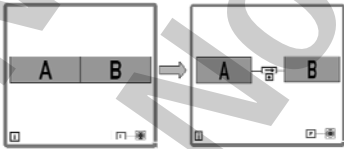

- Handle different parts of the process flow in parallel within one loop iteration
- Pass data to next piece of code using shift registers

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

Pipelining

Pipelining is a technique you can use to increase the throughput or speed of the FPGA VI.

- 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



ni.com/training

Pipelining Drawbacks

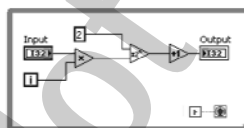
- Requires code to be restructured
 - Greatly reduced by using Feedback Nodes
- N-cycle delay before valid output data
 - Where $N = \#Discrete\ Steps\ Used - 1$
- May increase latency
 - Where $Latency\ Increase\ (clock\ ticks) = 2 \times N$
 - Usually negligible



ni.com/training

Basic Example of Pipelining

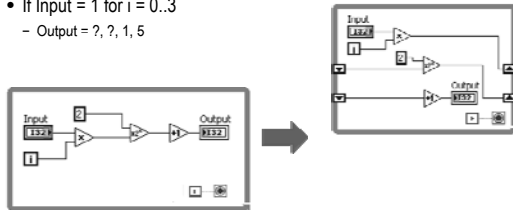
- Simple set of code with inputs and outputs
- Takes 7 clock cycles to execute
 - 5 cycles for the code, 2 for the loop
- If Input = 1 for $i = 0..3$
 - Output = 1, 5, 9, 13



ni.com/training

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 = ?, 2, 1, 5



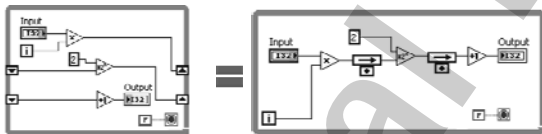
NATIONAL INSTRUMENTS

ni.com/training

Basic Example of Pipelining

Maintain look and feel of original application by using Feedback Nodes in sequence with code

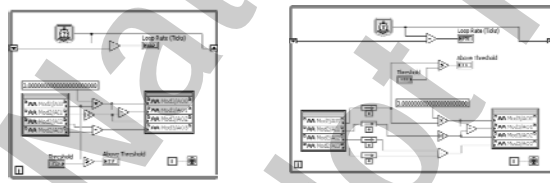
- Same functionality as a Shift Register
- Maintains more congruous VI appearance



NATIONAL INSTRUMENTS

ni.com/training

Pipelining



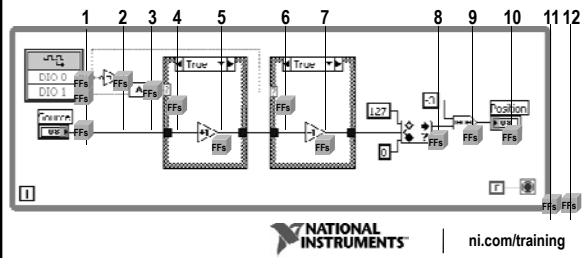
Analog Input 1/8 Ticks
 212 clock cycles (5.3 μ s)
 Scaling Analog Output 38 Ticks
 172 clock cycles (4.3 μ s)
 ~19% Faster

NATIONAL INSTRUMENTS

ni.com/training

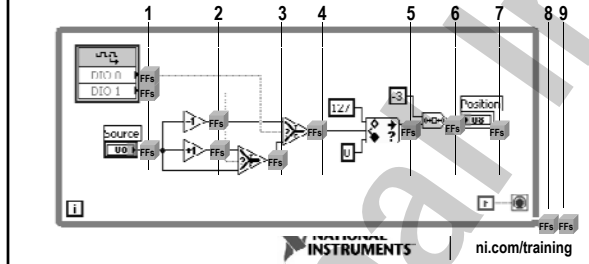
Pipelining Optimization

- What to do if your diagram executes too slowly?
- 12 clock cycles



Pipelining Optimization

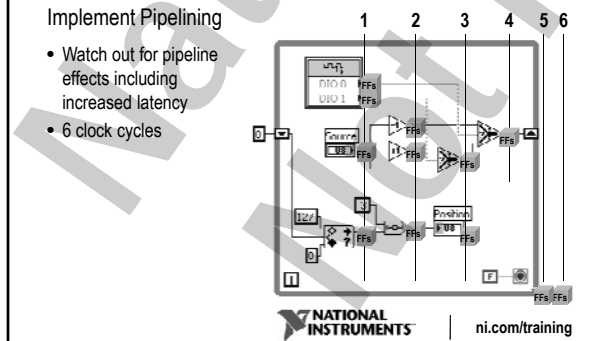
- Shorten the longest path using basic optimizations
- 9 clock cycles



Pipelining Optimization

Implement Pipelining

- Watch out for pipeline effects including increased latency
- 6 clock cycles



B. Using Pipelining in SCTLs

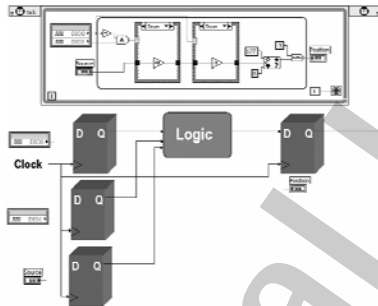
SCTL limited by propagation delays through the FPGA circuitry

- 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



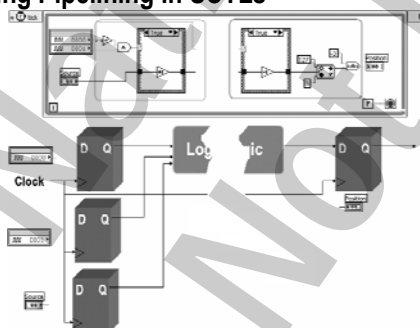
ni.com/training

Using Pipelining in SCTLs

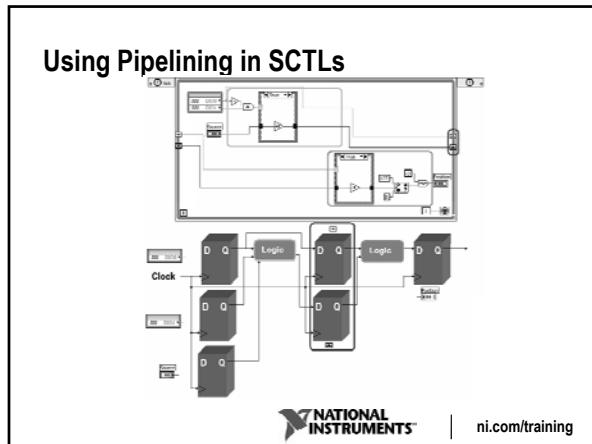


ni.com/training

Using Pipelining in SCTLs



ni.com/training



National Instruments
Not For Distribution



Additional Information and Resources

This appendix contains additional information about National Instruments technical support options and LabVIEW resources.

National Instruments Technical Support Options

Visit the following sections of the award-winning National Instruments Web site at 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 on demand training modules via the Services Resource Center. NI offers complementary membership for a full year after purchase, after which you may renew to continue your benefits.

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. areas. Visit ni.com/training for more information about the NI certification program.