LabVIEW™

FPGA Module User Manual

April 2003 Edition
Part Number 370690A-01

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 61 2 9672 8846, Austria 43 0 662 45 79 90 0, Belgium 32 0 2 757 00 20, Brazil 55 11 3262 3599,
Canada (Calgary) 403 274 9391, Canada (Montreal) 514 288 5722, Canada (Ottawa) 613 233 5949,
Canada (Québec) 514 694 8521, Canada (Toronto) 905 785 0085, Canada (Vancouver) 514 685 7530,
China 86 21 6555 7838, Czech Republic 420 2 2423 5774, Denmark 45 45 76 26 00,
Finland 385 0 9 725 725 11, France 33 0 1 48 14 24 24, Germany 49 0 89 741 31 30, Greece 30 2 10 42 96 427,
Hong Kong 2645 3186, India 91 80 51190000, Israel 972 0 3 6393737, Italy 39 02 413091,
Japan 81 3 5472 2970, Korea 82 02 3451 3400, Malaysia 603 9059 6711, Mexico 001 800 010 0793,
Netherlands 31 0 348 433 466, New Zealand 64 09 914 0488, Norway 47 0 32 27 73 00,
Poland 48 0 22 3390 150, Portugal 351 210 311 210, Russia 7 095 238 7139, Singapore 65 6 226 5886,
Slovenia 386 3 425 4200, South Africa 27 0 11 805 8197, Spain 34 91 640 0085, Sweden 46 0 8 587 895 00,
Switzerland 41 56 200 51 51, Taiwan 886 2 2528 7227, United Kingdom 44 0 1635 523545

For further support information, refer to the *Technical Support and Professional Services* appendix. To comment on the documentation, send email to techpubs@ni.com.

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Trademarks

Patents

For patents covering National Instruments products, refer to the appropriate location: **Help»Patents** in your software, the `patents.txt` file on your CD, or `ni.com/patents`.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Contents

About This Manual

Conventions ix
Related Documentation x

Chapter 1
Introduction

Custom Hardware from LabVIEW 1-1
Additional Advantages of the FPGA Module 1-2
FPGA Module Application Development 1-2
    Execution Targets 1-2
    Execution of FPGA VIs 1-3
    Communication with FPGA VIs 1-3
        Interactive Front Panel Communication 1-3
        Programmatic FPGA Interface Communication 1-5
FPGA Module Examples 1-7

Chapter 2
Creating FPGA VIs

Targeting FPGA Devices 2-1
Utilizing FPGA Space 2-1
Performing Basic I/O 2-2
    Analog I/O 2-3
        Analog Input 2-3
        Analog Output 2-3
    Digital I/O 2-4
Timing FPGA VIs 2-5
    Creating Timed I/O Applications 2-5
    Creating Delays between Events 2-6
    Measuring Time between Events 2-6
Customizing I/O 2-7
    Creating Triggers 2-8
    Creating Counters 2-9
Using Parallel Operations 2-11
    Parallel Operations on the FPGA 2-11
    SubVIs on the FPGA 2-13
Understanding How to Program FPGA VIs 2-14
    Restricted and Unavailable VIs and Functions 2-14
    Mathematical Operations 2-15

Arrays 2-16
Memory 2-16
Controlling I/O Power-On States 2-16
Communicating with a Host VI 2-18
Interrupt-Based Communication 2-18

Chapter 3
Managing Shared Resources

Resource Contention and Arbitration 3-1
Arbitration Options 3-3
Normal 3-3
Normal (Optimize for Single Accessor) 3-3
None 3-4
Available Arbitration Options for Specific Resources 3-4
Jitter 3-5
Timing 3-6
FPGA Utilization 3-7

Chapter 4
Running FPGA VIs

Compiling FPGA VIs 4-1
Compiling FPGA VIs Using the LabVIEW FPGA Compile Server 4-2
Compiling on a Remote Computer 4-2
Managing Compilation Files 4-3
Using Compiled FPGA VI Options 4-3
Changing the FPGA Device Clock Rate 4-3
Configuring FPGA VIs to Run Automatically 4-4
Downloading FPGA VIs to the FPGA Device 4-4
Running FPGA VIs 4-4
Running FPGA VIs at Power On 4-5
Setting Target Configurations 4-5

Chapter 5
Communicating with FPGA VIs

Establishing Communication with the FPGA VI 5-1
Selecting the FPGA VI 5-1
Selecting the FPGA Device 5-2
Setting Open and Run Options 5-2
Reading and Writing Data to the FPGA VI 5-3

Responding to FPGA VI Interrupts 5-4
Waiting for Interrupts 5-4
Acknowledging Interrupts 5-4
Closing a Reference to the FPGA VI 5-5

Chapter 6
Debugging FPGA VIs

Testing a VI Before Compiling 6-1
Building Debugging into an FPGA VI 6-2
Adding Indicators 6-2
Adding I/O 6-2

Appendix A
Technical Support and Professional Services

Glossary

About This Manual

This manual describes the LabVIEW FPGA Module software and techniques for building applications in LabVIEW with the FPGA Module. Use this manual to learn about FPGA Module programming features to help you build VIs that run on National Instruments Reconfigurable I/O (RIO) devices and VIs to communicate with RIO devices.

If you are not using the FPGA Module, refer to Chapter 5, *Communicating with FPGA VIs*, for information about communicating with the FPGA device.

Conventions

The following conventions appear in this manual:

» The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** 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.

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 and palette names.

italic Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font 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 should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and code excerpts.

Related Documentation

The following documents contain information that you might find helpful as you read this manual:

- *LabVIEW User Manual*
- *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**
- *LabVIEW FPGA Module Release Notes*
- *Where to Start with the NI PXI-7831R*
- *NI PXI-7831R User Manual*
- *LabVIEW Real-Time Module User Manual*

1 Introduction

With the LabVIEW FPGA Module and LabVIEW, you can create VIs that run on National Instruments Reconfigurable I/O (RIO) devices. Reconfigurable I/O devices, also known as *FPGA devices*, contain a reconfigurable FPGA (Field-Programmable Gate Array) surrounded by fixed I/O resources. Depending on the specific FPGA device, fixed I/O resources can include analog and digital resources—such as analog-to-digital converters (ADCs) and digital-to-analog converters (DACs)—that you can control from the FPGA.

With the FPGA Module, you configure the behavior of the reconfigurable FPGA to match the requirements of a specific measurement and control system. The VI you create to run on a FPGA device is called the *FPGA VI*. Use the FPGA Module to write FPGA VIs. When you download the FPGA VI to the FPGA, you are programming the functionality of the FPGA device. Each new FPGA VI you create and download is a custom timing, triggering, and I/O solution.

Custom Hardware from LabVIEW

When standard hardware did not meet your requirements for a specific application prior to the FPGA Module, you had to create a custom hardware design using low-level hardware description languages. With the FPGA Module, you do not need to know a hardware description language to design a specific hardware solution—you just need LabVIEW. With the FPGA Module, you can design and rapidly develop hardware components with the power of LabVIEW graphical programming.

The FPGA Module is ideal for programming applications that require functionality such as the following:

- **Custom I/O**—Modified digital and analog lines with custom counters, encoders, and pulse width modulators (PWMs)
- **On-board decision making**—Control, digital filtering, and Boolean decisions
- **Resource synchronization**—Precise timing of FPGA device resources, such as analog input (AI), analog output (AO), digital input

and output (DIO), counters, and PWMs, as well as synchronization among multiple devices

Additional Advantages of the FPGA Module

The FPGA Module expands the functionality of LabVIEW solutions. For example, you can design FPGA VIs that allow the FPGA device to operate independently of the rest of the system. You can create robust FPGA VIs that use the ability to operate independently and continue to run even if the *host computer*—the computer that controls and monitors the FPGA device—crashes. Furthermore, you can design the FPGA VI to store data on the FPGA until the host computer can retrieve the data.

Another advantage of the FPGA Module is parallel execution of block diagram operations in an FPGA VI. Portions of the block diagram that do not depend on other portions execute in parallel on the FPGA device. For example, multiple independent While Loops on a block diagram each have independent portions of hardware. Therefore, the multiple independent While Loops run simultaneously on the FPGA device.

FPGA Module Application Development

FPGA Module applications range from a single FPGA VI running on a FPGA device to large LabVIEW solutions that include multiple FPGA devices, the LabVIEW Real-Time Module, and LabVIEW for Windows. In any case, you need to create the FPGA VI that runs on the FPGA device. To create an FPGA VI, first select the FPGA device as the execution target in LabVIEW. An *execution target* is any location—including FPGA devices, RT targets, or the development computer—on which you can run a VI.

Execution Targets

By default, LabVIEW selects the development computer as the execution target. You must change the execution target to access the FPGA Module palettes, VIs, functions, and development tools. To change the execution target from the **LabVIEW** dialog box, select an FPGA device from the **Execution Target** pull-down menu. Even if the target device is not present, you still can target the FPGA device to develop an FPGA VI. If you are currently working on a VI and you want to change the execution target, you can select **Operate»Switch Execution Target** to set the execution target.

Refer to Chapter 2, *Creating FPGA VIs*, for information about good programming techniques and the VIs, functions, and tools in the FPGA Module that you need to create efficient FPGA VIs.

Execution of FPGA VIs

After you create an FPGA VI with the FPGA Module VIs, functions, and tools, use LabVIEW to compile and download the FPGA VI to the FPGA device. As you do with any other VI, click the **Run** button to automatically compile, download, and run the FPGA VI on the execution target, which in this case is the FPGA device. Refer to Chapter 4, *Running FPGA VIs*, for information about compiling, downloading, and running FPGA VIs on the FPGA device.

Communication with FPGA VIs

After you have an FPGA VI running on the FPGA device, you need a way to communicate with that VI. Depending on the application requirements, you can communicate with the FPGA VI interactively or programmatically. Use Interactive Front Panel Communication to communicate with the FPGA VI directly from the front panel of the FPGA VI. Use Programmatic FPGA Interface Communication to communicate with the FPGA VI from a VI running on the host computer. The VI running on the host computer is called the *host VI*.

Interactive Front Panel Communication

Use Interactive Front Panel Communication to communicate with an FPGA VI running on an FPGA device with no additional programming. With Interactive Front Panel Communication, the host computer displays the FPGA VI front panel and the FPGA device executes the FPGA VI block diagram, as shown in Figure 1-1.

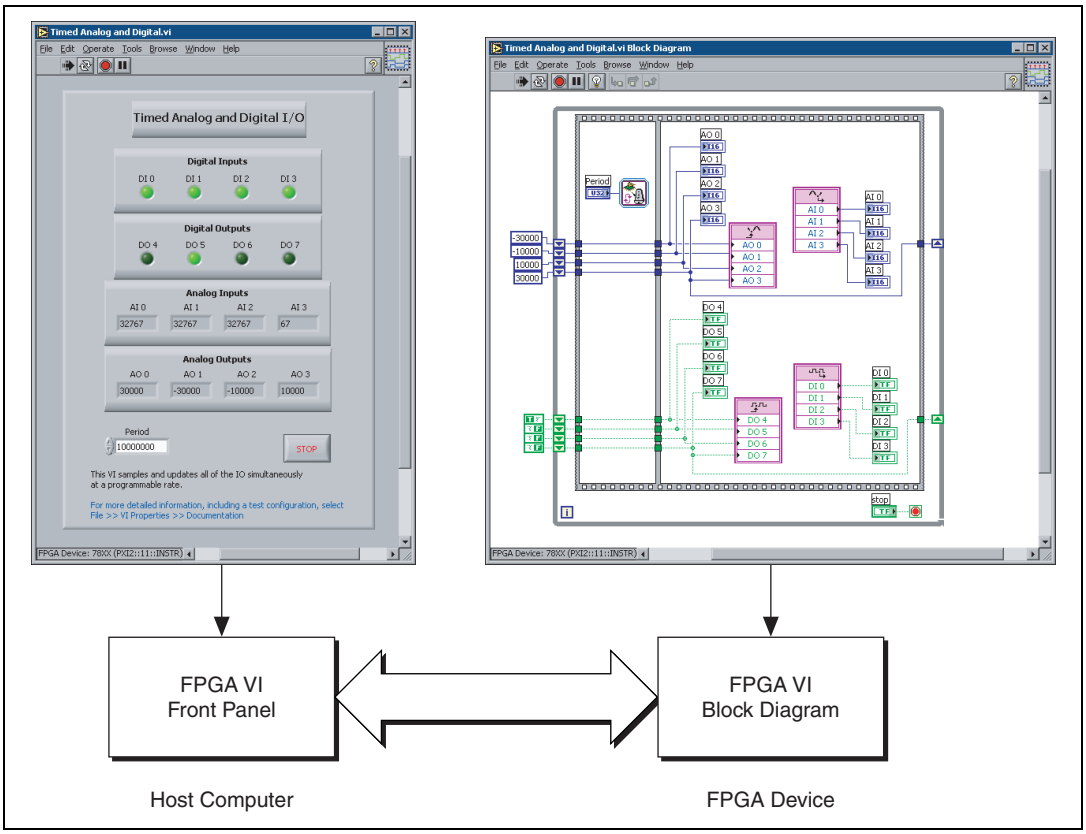

Figure 1-1. Interactive Front Panel Communication

The LabVIEW front panel communicates with the FPGA device block diagram to exchange the state of the controls and indicators. You can communicate with an FPGA device located in the host computer or with an FPGA device located in a remote system. As the FPGA device block diagram continues to run, the host computer updates values on the FPGA VI front panel as often as possible. The execution rate of the FPGA VI is not affected by the host computer updates to the controls and indicators.

Use Interactive Front Panel Communication between the FPGA device and the host computer to control and test VIs running on the FPGA device. After downloading and running the FPGA VI, keep LabVIEW open on the host computer to display and interact with the front panel of the FPGA VI.

During Interactive Front Panel Communication, you cannot use LabVIEW debugging tools—including probes, execution highlighting, breakpoints, and single-stepping. To identify errors before you compile, download, and

run the FPGA VI on the FPGA device, test the FPGA VI by targeting an FPGA device emulator. An *emulator* is an execution target that simulates the behavior of the FPGA VI running on the FPGA device. Refer to Chapter 6, *Debugging FPGA VIs*, for more information about testing FPGA VIs with emulators.

Programmatic FPGA Interface Communication

With Programmatic FPGA Interface Communication, you programmatically monitor and control an FPGA VI with a separate host VI running on the host computer. You might write a host VI to send information between the host computer and the FPGA device for the following reasons:

- You want to do more data processing than you can fit on the FPGA.
- You need to perform operations not available on the FPGA device, such as floating-point arithmetic.
- You want to create a multi-tiered application with the FPGA device as a component of a larger system.
- You want to log data.
- You want to run multiple VIs on the host computer. You cannot use LabVIEW on the host computer for any other task when you target an FPGA device or RT target while using Interactive Front Panel Communication.
- You want to control the timing and sequencing of data transfer.
- You want to control which components are visible on the front panel because some controls and indicators might be more important for communication than others.

When you use Programmatic FPGA Interface Communication, the FPGA VI runs on the FPGA device, and the host VI runs on the host computer, as shown in Figure 1-2. Use the FPGA Interface functions available when you target LabVIEW for Windows or an RT target to create a host VI that communicates with the FPGA VI and performs other required functions. Refer to Chapter 5, *Communicating with FPGA VIs*, for information about creating host VIs.

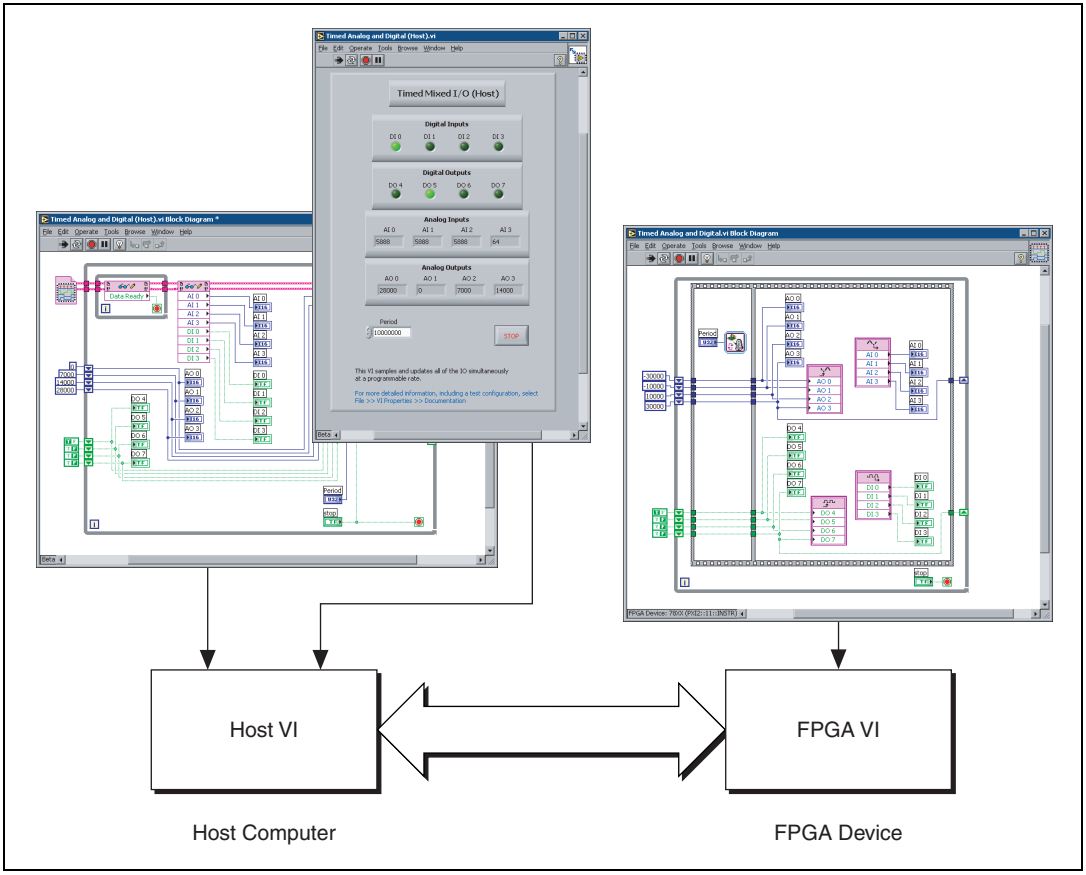

Figure 1-2. Programmatic FPGA Interface Communication

You also can use an RT target as the host computer. The RT target can use Programmatic FPGA Interface Communication to communicate with the FPGA device. You then can use a Windows computer to communicate with the RT target. The flexibility of FPGA devices integrates well with LabVIEW Real-Time Module applications, such as control and hardware-in-the-loop simulations, which require a significant amount of determinism.

FPGA Module Examples

The FPGA Module includes example FPGA VIs and example host VIs located in the `examples\FPGA` directory. The FPGA Module examples are divided into categories such as Getting Started, Timing and Triggering, Counters, and so on. The FPGA Module also includes VI templates to help you create specific FPGA VI solutions.

Begin with the Getting Started examples to learn about simplified functions based on actual application VIs. The Getting Started examples highlight key concepts, such as communication between the host VI and the FPGA VI as well as simplified timing, triggering, and data transfer. Continue through the other categories of FPGA Module examples for more detailed information.

Select **Help»Find Examples** to search the development computer and `ni.com` for FPGA Module examples.

2

Creating FPGA VIs

This chapter describes how to create an FPGA VI for an FPGA device. You will learn how to perform common tasks such as I/O, timing, and triggering, as well as more advanced tasks such as using parallel operations.

Targeting FPGA Devices

The LabVIEW FPGA Module provides the same graphical programming environment for the creation of FPGA VIs as LabVIEW does for standard VIs. The LabVIEW graphical programming environment includes front panels and block diagrams, powerful editing tools, and a wide range of included functions.

When you target LabVIEW to an FPGA device, you have access only to the LabVIEW VIs and functions that make sense on the FPGA device. For example, a typical FPGA device does not have access to a disk drive, so File I/O functions are not available on the **Functions** palette when you target that device. The LabVIEW VIs and functions available when you target an FPGA device have the same behavior and functionality in FPGA VIs as in VIs created for Windows. In addition to the subset of the standard LabVIEW VIs and functions, the FPGA Module provides FPGA device-specific VIs and functions. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for information about FPGA Device I/O functions and VIs.

Tip You can identify FPGA Device I/O functions on the palettes by their purple borders.

Utilizing FPGA Space

Every function or VI you place on the block diagram of an FPGA VI uses a certain number of logic cells on the FPGA. The FPGA on the FPGA device has a fixed number of logic cells. If the FPGA VI design exceeds the number of available logic cells, you must reduce the number of logic cells the FPGA VI uses on the FPGA. This manual contains information throughout to help you minimize the size of FPGA VIs.

When you compile the VI, LabVIEW displays a Compile Report of the FPGA usage. Refer to Chapter 4, *Running FPGA VIs*, for more information.

Performing Basic I/O

The FPGA Device I/O functions correspond to the fixed I/O resources on the FPGA device. Fixed resources can include analog input, digital output, and so on. When the FPGA VI runs on the FPGA device, it performs the I/O operations in hardware. For example, the Analog Input function initiates a conversion on the analog-to-digital converter (ADC) and returns the result to the FPGA VI. Because FPGA VIs run directly on the FPGA, you do not need driver calls or experience software delays.

Each FPGA Device I/O function corresponds to a specific type of fixed I/O resource. An FPGA device might include multiple I/O resources of the same type. Each individual I/O resource is a terminal on the FPGA device. You can configure the FPGA Device I/O functions to read or write to as many terminals as are available on the FPGA device. For example, you can use the Analog Input function to read the data input on any of the analog input terminals on the FPGA device.

Complete the following steps to configure an FPGA Device I/O function.

1. Place the appropriate FPGA Device I/O function on the block diagram. The FPGA Module offers functions for analog input and output, digital input and output, and digital port input and output.
2. Double-click or right-click the function on the block diagram and select **Properties** from the shortcut menu.

   Notice that the **Configure** dialog box contains one fixed I/O resource in the **Preview** listbox.
3. Select an available fixed I/O resource with which you want to associate inputs or outputs from the **Terminal** listbox on the **General Configuration** page.

   Refer to the hardware documentation for information about terminals and their connector assignments.
4. Type a name in the **Alias** field to specify an **Alias** for the fixed I/O resource. LabVIEW uses the terminal name as the default **Alias**.

5. To associate more inputs or outputs with a fixed I/O resource, click the **Add Input** or **Add Output** button and configure the fixed I/O resource as you did in the previous step.
6. Click the **OK** button to save the I/O configuration and close the **Configure** dialog box.

Analog I/O

Analog Input

The Analog Input function initiates a conversion, waits for the result, then returns the binary representation of the voltage as a 16-bit signed integer. Typically you create the FPGA VI to use the binary representation for operations within the FPGA VI. You also can pass the binary representation back to the host VI and convert the binary representation back to a voltage.

The equation you use to convert the binary representation back to an actual voltage depends on the specific FPGA device. Refer to the hardware documentation for more information. For example, with an NI PXI-7831R device, use the following equation to convert the binary representation to voltage:

$$Input\ Voltage = \frac{Binary\ Code}{32768} \times 10.0\ \text{V}$$

Note Avoid executing this calculation in the FPGA VI because the FPGA only supports integer operations. Also, performing the equation on the FPGA uses additional space on the FPGA. Refer to the *Mathematical Operations* section for more information.

Analog Output

The Analog Output function writes the binary representation of the voltage as a 16-bit signed integer to the digital-to-analog converter (DAC), which sets the analog output voltage. You can generate voltage information in two sources—the host VI or the FPGA VI. Typically the host VI converts the voltage to a signed 16-bit binary representation before writing the value to the FPGA VI. If the FPGA VI determines the voltage, typically the FPGA VI performs the calculations using 16-bit binary representations. In both cases, the DAC passes the binary representation out as a voltage.

The equation you use to convert a voltage to a binary representation depends on the specific FPGA device. Refer to the hardware documentation for more information. For example, with an NI PXI-7831R device, use the following equation to convert the voltage to the binary representation:

$$Binary\ Code = \frac{Output\ Voltage \times 32768}{10.0\ \text{V}}$$

Note Avoid executing this calculation in the FPGA VI because the FPGA only supports integer operations. Also, performing the equation on the FPGA uses additional space on the FPGA.

Digital I/O

You can treat digital resources as individual lines or as predefined groups of eight digital lines, also known as ports. A terminal is either an individual digital line or a digital port depending on which FPGA Device I/O function you use. You can perform both digital input and digital output on any digital terminal. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for information about specific FPGA Device I/O functions and port assignments.

Use the Digital Input and Digital Port Input functions to read the state of a digital terminal. The state of the digital terminal is commonly determined by an external signal, such as the output generated by an external device. Use the Digital Output and Digital Port Output functions to set the state of a digital terminal or port. You can use the Digital Input functions to verify the state of the same terminal to which the Digital Output function writes.

Note If you have used a terminal for output, you must use the Digital Enable or Digital Port Enable function to disable the terminal for output before the Digital Input function can read the state of an external signal.

The Digital Output and Digital Port Output functions both write the data and enable the terminal for output. You also can use the Digital Data and Digital Port Data functions, which write data to a terminal but do not enable the output. Use the Digital Enable and Digital Port Enable functions to enable the digital terminal, which allows the data to be driven out. For example, you might have one portion of the block diagram continuously generating an internal signal. Use a Digital Enable or Digital Port Enable function in another portion of the block diagram to independently control when the internal signal is actually driven out to an external device.

Timing FPGA VIs

Every VI or function you place in an FPGA VI takes a certain amount of time to execute. You can allow operations to occur at the rate determined by the dataflow without additional programming. If you want to control or measure the execution timing, use the Time & Dialog VIs. You also can use the Time & Dialog VIs to create custom I/O such as counters and triggers.

Creating Timed I/O Applications

Applications often require the I/O to execute at a specific frequency. For example, the algorithms used in control loops typically require the inputs to be sampled at a known rate. Use the Loop Timer VI in a While Loop to control the execution rate of the I/O, as shown in Figure 2-1.

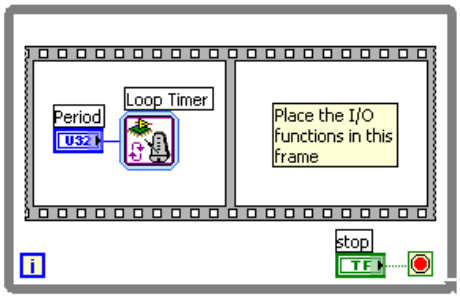

Figure 2-1. Controlling Execution Rate with the Loop Timer VI

To use the Loop Timer VI to control the execution rate of the I/O, place a sequence structure inside a While Loop. Place the Loop Timer VI in the first frame of the sequence structure. Configure the **Counter Units** and **Size of Internal Counter** in the **Configure Loop Timer** dialog box that appears. Place the LabVIEW code for the I/O in subsequent frames of the sequence structure.

Tip You can save space on the FPGA device by choosing the smallest **Size of Internal Counter** you can use for the application.

The I/O executes at the rate specified by the **Count** parameter of the Loop Timer VI. You can use the Timed Loop VI template to quickly create an FPGA VI that uses the Loop Timer VI.

The first call of the Loop Timer VI does not result in any wait or delay because it establishes a reference time stamp for subsequent calls. After the

first call of the Loop Timer VI, subsequent calls of the Loop Timer VI do not return until the time specified by the **Count** parameter has elapsed since the previous call. If the time specified by the **Count** parameter is less than the time it takes the FPGA device to execute the code in the While Loop, the Loop Timer VI returns immediately and establishes a new reference time stamp for subsequent calls.

Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for more information about the Loop Timer VI.

Creating Delays between Events

Use the Wait VI to create a delay between events in an FPGA VI. For example, you might want to create a delay between a trigger and a subsequent output. You can place the LabVIEW code for the trigger in the first frame of a sequence structure. Then place the Wait VI in the following frame. Finally, place the LabVIEW code for the output in the last frame of the sequence structure. You also can create a series of delays using multiple Wait VIs in a sequence structure, as shown in Figure 2-2.

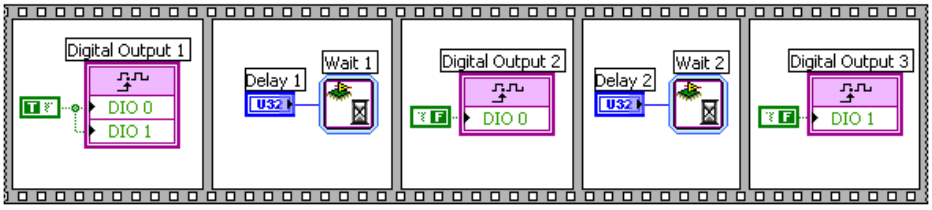

Figure 2-2. Using Wait VIs for a Series of Delays

Measuring Time between Events

Use the Tick Count VI to measure the time between events such as edges on a digital signal. You can use the Tick Count VI when you need to determine the period, pulse-width, or frequency of an input signal or if you want to determine the execution time of a section of LabVIEW code.

For example, each function or VI in an FPGA VI takes a certain amount of time to execute. To determine the amount of time it takes a function or a section of LabVIEW code to execute, use a sequence structure with two Tick Count VIs, as shown in Figure 2-3.

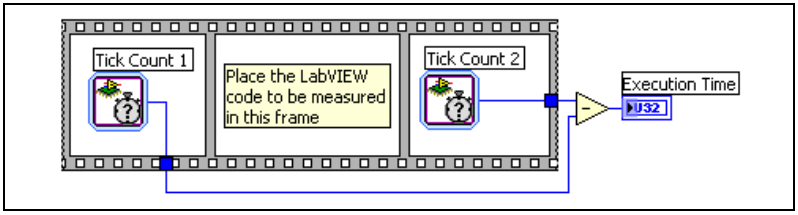

Figure 2-3. Measuring Execution Time with the Tick Count VI

Place one Tick Count VI in the first frame of the sequence structure. Then place the LabVIEW code you want to measure in the second frame of the sequence structure. Finally, place the other Tick Count VI in the last frame of the sequence structure. You then can calculate the difference between the results of the two Tick Count VIs to determine the execution time.

The Tick Count VI has an internal counter to track time. The internal counter for each Tick Count VI you place on the same block diagram shares the same start time. Therefore, every Tick Count VI that uses the same values for the **Counter Units** and **Size of Internal Counter** options tracks the same time. For example, if two Tick Count VIs that use the same **Configure Tick Count** options are called simultaneously, they return the same **Tick Count** value.

The **Tick Count** value returned by the Tick Count VI is in integer values of **Counter Units**. The **Tick Count** value cannot represent any fractional time periods that may occur when **Counter Units** is configured for **uSec** or **mSec**. This can result in timing measurements that have an accuracy of ±1 **Counter Unit** value. For example, the Tick Count VIs in Figure 2-3 can be configured to measure time in milliseconds. If the first Tick Count VI executes at 47.9 milliseconds, **Tick Count** returns a value of 47. If the second Tick Count VI executes at 53.2 milliseconds, **Tick Count** returns a value of 53. Although this example has a 5.3 milliseconds delay, the difference between the returned values is 6 milliseconds.

Customizing I/O

The FPGA Module includes functions for performing basic I/O. However, you might have applications that require more advanced or custom I/O functionality. Use the FPGA Device I/O functions as building blocks to create customized I/O functionality such as triggering and counters.

Creating Triggers

In many applications, you might need to wait for a trigger before performing an action. You can wait for a trigger on a single digital input using the Wait on Edge or Level function. This function waits until the specified condition is met on the digital input before continuing. Place the Wait on Edge or Level function in the first frame of a sequence structure and place the LabVIEW code for the task in the following frame, as shown in Figure 2-4.

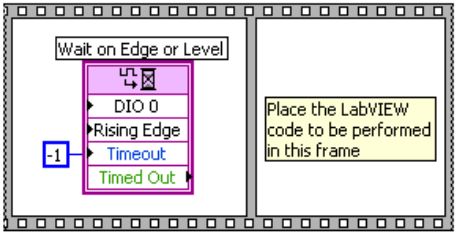

Figure 2-4. Creating a Trigger with the Wait on Edge or Level Function

You also can create more advanced triggering events from the FPGA Device I/O functions. For example, you might need an application that triggers only when multiple digital lines match a given condition, as shown in Figure 2-5.

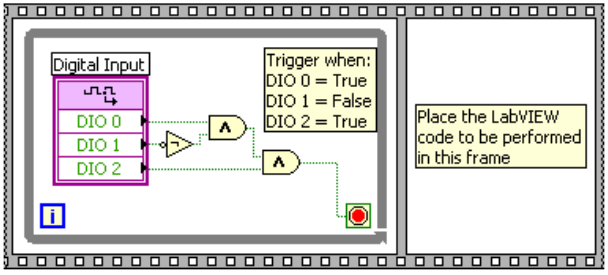

Figure 2-5. Triggering when Multiple Digital Lines Match a Condition

You can place the Digital Input function in a While Loop and exit the While Loop only when the digital inputs match the trigger pattern. Place the While Loop in the first frame of a sequence structure, just as you did for the Wait on Edge or Level function in the previous example.

You can implement analog triggers in the same manner. Place an Analog Input function and a Comparison function in a While Loop to trigger when the analog input value exceeds a programmable threshold.

Creating Counters

Counters can range from simple event counters to complex signal measurement with multiple inputs and outputs. You can build a simple event counter with the Wait on Edge or Level function in a While Loop. For example, you can use the Wait on Edge or Level function to wait for a rising edge to occur on a digital input terminal, as shown in Figure 2-6.

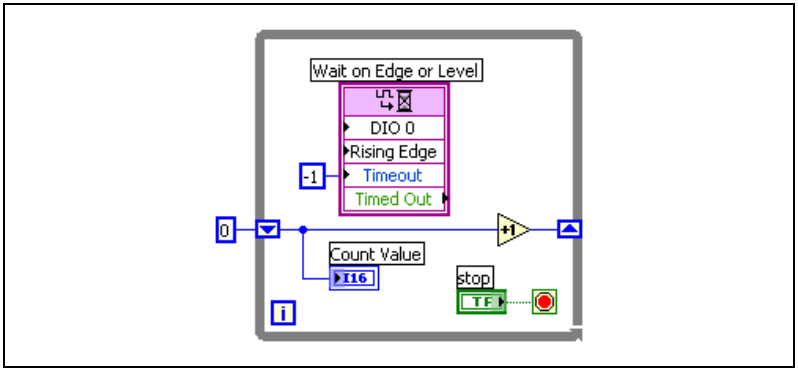

Figure 2-6. Counting Rising Edges

When the Wait on Edge or Level function detects an edge, the block diagram increments the counter value and stores the counter value in a shift register on the While Loop. You can use an indicator to view the counter value either on the front panel or by a local variable.

You also can build more advanced counters from the FPGA Device I/O functions. For example, an application might require a counter with independent count up, count down, and gate inputs and an output, as shown in Figure 2-7.

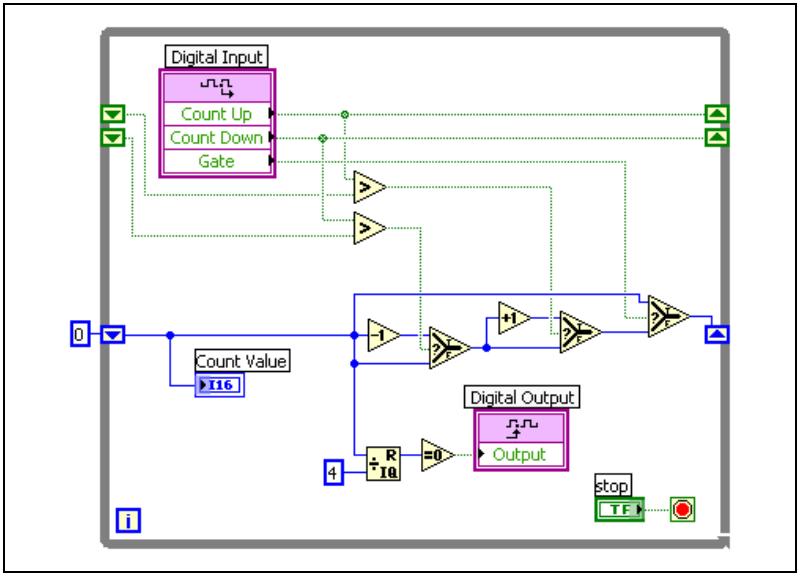

Figure 2-7. Building More Advanced Counters

In Figure 2-7, the counter value increments when a rising edge occurs on count up, the counter value decrements when a rising edge occurs on count down, and the gate prevents count up and count down from changing the counter value when the gate is high. The output gets asserted when the counter value is a multiple of four. You can make simple Boolean decisions in LabVIEW code to determine if the counter counts up, down, or stays the same. You also can make simple mathematical decisions in LabVIEW code to determine when the output asserts.

You also can make measurements on input signals, as shown in Figure 2-8. For example, you might need to measure the period of an input signal. You can place the Wait on Edge or Level function in the first frame of a sequence structure followed by the Tick Count VI in the second frame of the sequence structure. Then place the sequence structure in a While Loop. Store the current value returned by Tick Count in a shift register to create the previous value for the next iteration of the While Loop. Then subtract the previous time from the current time to determine the period of the input signal.

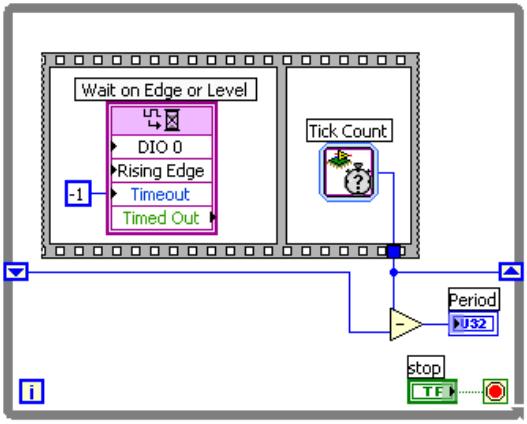

Figure 2-8. Measuring the Period of an Input Signal

Using Parallel Operations

As a fundamental part of the LabVIEW environment, LabVIEW allows you to create VIs that include parallel operations. When the VI executes on a processor-based target such as Windows, LabVIEW imitates parallel operation by serially executing portions of the block diagram. In FPGA VIs, parallel operations execute simultaneously on the FPGA device because the FPGA Module creates dedicated hardware for each independent VI or function in the FPGA VI.

Parallel Operations on the FPGA

Parallel operations on the FPGA typically increase determinism and execution rate when compared to a processor-based target. Because the parallel operations no longer contend over a common resource, such as the processor used by LabVIEW for Windows, you increase determinism. Because the overall execution time of multiple operations, with dedicated hardware for each operation, is the execution time of the slowest operation, you increase execution rate. With a single hardware resource, the overall execution time for multiple operations is the sum of the execution times.

To create parallel operations, use multiple independent While Loops on a single block diagram. For example, you can implement multiple data acquisition engines, each with an independent sampling rate, as shown in Figure 2-9.

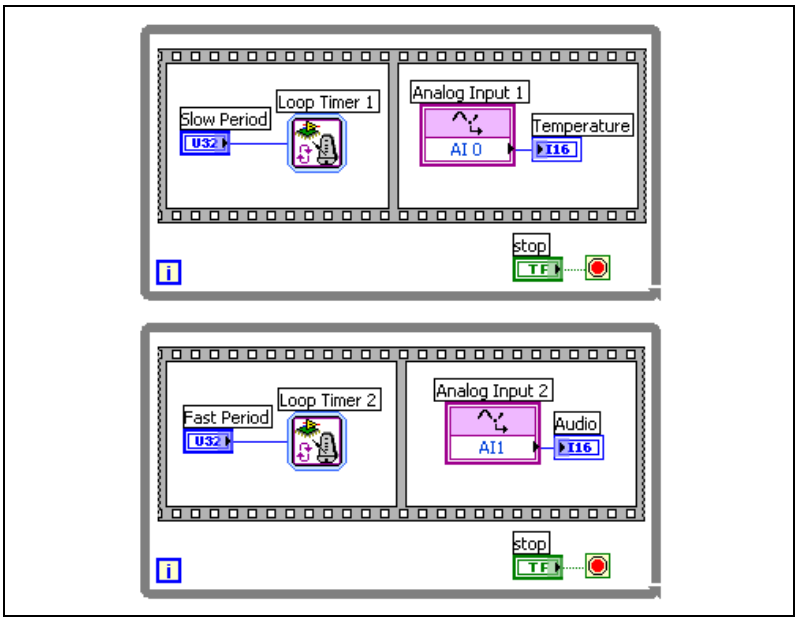

Figure 2-9. Implementing Multiple Data Acquisition Engines

You can use independent sampling rates to more efficiently acquire data in systems that contain both high frequency and low frequency signals. Configure one data acquisition engine with a fast sampling rate to measure a high frequency signal, such as audio signals. Configure the other data acquisition engine with a slower sampling rate to measure a low frequency signal, such as temperature.

If you use shared resources among parallel operations, you might lose the benefits of determinism and a higher execution rate. Possible shared resources include digital output lines, analog lines, memory blocks, the interrupt line, front panel controls, local variables, and non-reentrant subVIs. Refer to Chapter 3, *Managing Shared Resources*, for information about shared resources.

Tip Each parallel operation uses a certain amount of space on the FPGA. If you begin to run out of space on the FPGA and have identical parallel operations, you might save space by creating a subVI for the operation and making it non-reentrant. However, you lose parallel execution by creating a non-reentrant subVI for the operation.

SubVIs on the FPGA

LabVIEW allows you to encapsulate common sections of code as subVIs to facilitate its reuse on the block diagram. You can configure the subVI as a single instance shared among multiple callers, also known as a non-reentrant VI. You also can configure the subVI to replicate itself for each caller, also known as a reentrant VI. By default, LabVIEW subVIs are non-reentrant VIs. You can change the subVI to be a reentrant VI in the subVI **VI Properties** dialog box. Select **Execution** from the **Category** pull-down menu and place a checkmark in the **Reentrant Execution** checkbox.

If you use a non-reentrant subVI in an FPGA VI, only a single copy of the subVI becomes hardware and all callers share the hardware resource. If you use a reentrant subVI in an FPGA VI, each call of the subVI generates a dedicated hardware resource. For example, if you have five instances of an event counter configured as a reentrant subVI on the block diagram, LabVIEW implements five independent copies of the event counter hardware on the FPGA.

Be careful not to use shared resources in reentrant subVIs when you want to have dedicated hardware for each copy of the subVI. If you use any shared resource in a reentrant subVI, only one copy of the shared resource exists in hardware. Each reentrant subVI must use arbitration to access the shared resource. Refer to Chapter 3, *Managing Shared Resources*, for information about shared resources.

Although non-reentrant subVIs typically consume less space in the FPGA VI, the FPGA VI might run slower because it shares resources on the FPGA. Reentrant VIs typically consume more space in the FPGA VI, but the FPGA VI might run faster without shared resources. Table 2-1 summarizes the typical advantages and disadvantages of non-reentrant and reentrant subVIs.

Table 2-1. Non-Reentrant versus Reentrant SubVIs

VI Type	FPGA Speed	FPGA Utilization
Non-reentrant	Slower—Each call to the subVI waits until the previous call ends.	Lower—Only one instance of the subVI exists on the FPGA no matter how many times it is used.
Reentrant	Faster—Multiple calls to the same subVI run in parallel.	Higher—Each instance of the subVI on the block diagram uses space on the FPGA.

Understanding How to Program FPGA VIs

In addition to providing the I/O capabilities, the FPGA Module enables you to use the LabVIEW functions and VIs appropriate for FPGA devices.

Restricted and Unavailable VIs and Functions

Some LabVIEW functions and VIs are not available or have restrictions in FPGA VIs.

The following functions are not available for FPGA VIs:

- Floating-point functions
- Variable-size and multi-dimensional arrays
- Error clusters or strings
- Analyze VIs
- ActiveX
- Dialog boxes
- File I/O
- Printing
- Programmatic menus
- VI Server
- Property Nodes

Other LabVIEW features might not be supported depending on the FPGA device.

Mathematical Operations

The FPGA Module restricts the use of mathematical operations in FPGA VIs to integer numeric data types. You can perform integer math using the Numeric functions. You cannot use floating point operations in FPGA VIs.

Integer overflow occurs when the result of a mathematical operation exceeds the range of the output data type. For example, the range of a U8 integer is 0 to 255. Adding two U8 integers together that have a result greater than 255 results in overflow, such as 200 + 70. When overflow occurs the result rolls over the limit of the range and the modulo of the range is returned. For example, a result of 270 for a U8 integer rolls over 255 and returns as 14.

You can take advantage of the rollover behavior that occurs with overflow in some applications. For example, the execution time measurement in Figure 2-3 relies on the rollover behavior of overflow for proper operation. Suppose the Tick Count VIs are configured with an 8 bit **Size of Internal Counter** and milliseconds for **Counter Units**. When the internal counter of the Tick Count VI reaches 255 ms it rolls over to 0. If the first Tick Count VI returns a **Tick Count** of 132 ms and the execution time of the LabVIEW code to be measured takes 140 ms, the internal counter has rolled over and the second Tick Count VI returns a **Tick Count** value of 16 ms. When the block diagram subtracts 132 from 16, overflow occurs and results in the value of 140.

Note The Tick Count VI takes a single cycle to execute. In this example, if you set **Counter Units** as **Ticks** instead of **mSec**, the returned result from the subtraction is 141 even though the LabVIEW code in the middle sequence takes only 140 ticks to execute.

If you want to avoid overflow, you have two options—using larger data types and using the Saturation Arithmetic VIs. You might avoid data overflow with a larger data type, but larger data types consume more space on the FPGA.

Tip Use the smallest data type possible in FPGA VIs to minimize space used on the FPGA.

You also can use the Saturation Arithmetic VIs to handle signed integer overflow. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for information about the Saturation Arithmetic VIs.

Arrays

You can use only fixed-size, one-dimensional arrays in FPGA VIs. You can make any array constant, control, or indicator fixed-size by right-clicking the array index and selecting **Set Dimension Size** from the shortcut menu.

You cannot use an array function that returns a variable-size array. However, if you use appropriate constants with many array functions, the resulting array is fixed-size. For example, if you use the Array Subset function, you must wire constants to the **index** and **length** parameters so that the resulting subarray is fixed-size.

Tip Arrays consume significant amounts of space on the FPGA. To optimize compile time, avoid using arrays larger than 32 elements.

Memory

You can use FPGA memory for data storage in the FPGA VI. You access the FPGA memory using the Memory Read and Memory Write VIs available with the FPGA Module. You can use these VIs to perform basic read and write operations to the FPGA memory, and as building blocks to create more advanced memory functions such as FIFOs, dual ported memory, look-up tables, and so on.

You can create look-up tables with constant or variable entries in FPGA VIs. You can use fixed-size arrays for smaller look-up tables with variable entries. You can use constant fixed-size arrays when the look-up table entries do not need to change and you want to limit FPGA usage. For larger look-up tables, use the Memory Read and Memory Write VIs available with the FPGA Module to create look-up tables with variable entries in the FPGA memory.

Controlling I/O Power-On States

An application might require that the I/O on the FPGA device be set to a known value when the system powers on. For example, if an FPGA device controls hydraulic valves with the digital outputs, the FPGA device must keep the valves turned off until the host VI is launched and starts to control the system. You can create an FPGA VI and configure the FPGA device to set the power-on states of the FPGA device.

You must program the FPGA VI so that the block diagram sets the output states without any dependencies on the host VI. For example, you can place

the digital and analog output functions in the first frame of a sequence structure. You then place the rest of the LabVIEW code in the subsequent frames of the sequence structure, as shown in Figure 2-10. Then configure the FPGA VI to start executing as soon as it is loaded in the FPGA. Compile and download the FPGA VI to the flash memory on the FPGA device and configure the FPGA device to automatically load the FPGA VI from the flash memory when the FPGA device powers on. When the FPGA device powers on, the FPGA VI loads into the FPGA from the flash memory, and the FPGA VI starts executing immediately. The output functions in the first frame of the sequence structure on the FPGA VI set the output states. Refer to Chapter 4, *Running FPGA VIs*, for information about automatically loading and running FPGA VIs.

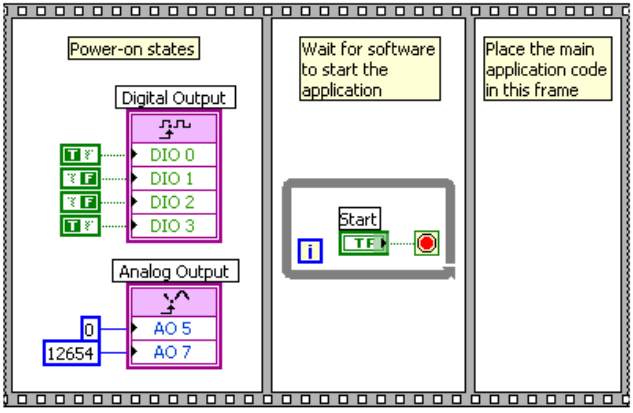

Figure 2-10. Setting the Output State without Host VI Dependency

You can create more than a static power-on state on the outputs of the FPGA device. You can create arbitrary power-on functionality that performs complex actions. For example, you can set outputs based on the state of the inputs, use serial communication with an external device, and so on.

Note If you use an I/O resource only once after the power-on state, you can select the **None** arbitration option to save space. Refer to Chapter 3, *Managing Shared Resources*, for information about arbitration.

Communicating with a Host VI

You can control and monitor data directly from the FPGA device using Interactive Front Panel Communication. You also can use a host VI running on the host computer or on an RT target to control or monitor the FPGA VI through Programmatic FPGA Interface Communication. With Interactive Front Panel Communication, you can use a polling-based method of communicating between the host VI and the FPGA VI by reading and writing indicators and controls. With Programmatic FPGA Interface Communication, you can use an interrupt-based method of communication where, in addition to communicating using indicators and controls, the FPGA VI can generate hardware interrupts that the host VI can wait for and acknowledge. You can use FPGA Interface functions available with the FPGA Module to create host VIs that communicate with the FPGA VI. Refer to Chapter 5, *Communicating with FPGA VIs*, for information about using the FPGA Interface functions.

A host VI can control and monitor only data passed through the FPGA VI front panel. For example, if you want the host VI to monitor the data from an Analog Input terminal, you must wire an indicator to the Analog Input function on the FPGA VI block diagram.

Interrupt-Based Communication

You can use interrupts to notify the host VI of events, such as data being ready, an error occurring, or a task finishing. An interrupt is a physical hardware line to the host that the FPGA device asserts.

Use the Interrupt VI in FPGA VIs to generate any of the 32 independent logical interrupts available on the FPGA device. Each logical interrupt specifies the reason for causing the interrupt and allows you to handle it differently in software. You can set the Interrupt VI to wait until the host VI acknowledges the interrupt on the FPGA device by wiring the **Wait Until Cleared** input. In this case, the Interrupt VI waits until the host VI controlling the device acknowledges the interrupt. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for more information about the Interrupt VI.

Use caution when you include simultaneous interrupt calls on the FPGA device. The interrupt becomes a shared resource if you use more than one, and this can induce jitter. Refer to Chapter 3, *Managing Shared Resources*, for more information about resolving resource contention.

The advantage of using interrupt-based communication instead of polling-based communication is that the host VI can perform other operations while waiting for the interrupt. In contrast, if the host VI uses polling-based communication, the host VI does not have time to perform other operations while waiting for a specific data value from the FPGA device.

3

Managing Shared Resources

This chapter describes how to use arbitration on shared resources in FPGA VIs. If the FPGA VI design fits on the FPGA and if the FPGA VI meets the performance expectations, keep the default **Arbitration** options.

Resource Contention and Arbitration

Many applications contain resources that are accessed from multiple functions or VIs in an FPGA VI. For example, an application might use the FPGA memory to temporarily store data from two independently operating data acquisition loops. The FPGA Module includes arbitration to determine which location can access the resource if the locations request access at the same time.

Resource contention occurs when you include two or more functions or VIs on the FPGA VI block diagram that simultaneously request access to the same shared resource. A *requestor* becomes an *accessor* when it actively requests information from a specific resource and is granted access by a special component called an *arbiter*. The arbiter determines which requestor becomes an accessor when resource contention occurs. Possible shared resources include digital output lines, analog lines, memory blocks, the interrupt line, front panel controls, local variables, and non-reentrant subVIs.

Figure 3-1 illustrates an FPGA VI with arbitration between the first and second requestor of AI0.

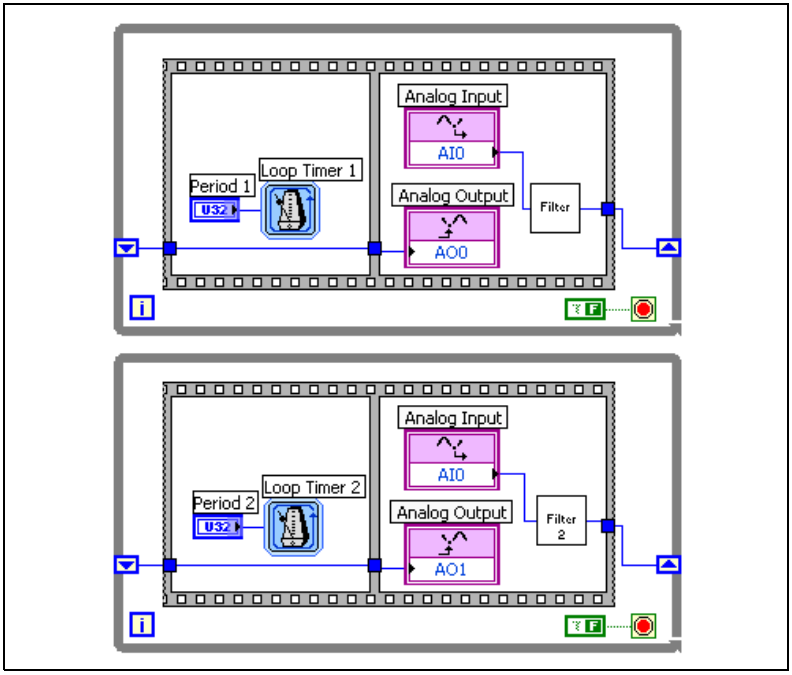

Figure 3-1. Arbitration between Two Analog Input Requestors

Notice that the two While Loops might simultaneously request access to AI0, depending on the values of the **Period** and **Period 2** controls. Similarly, one While Loop might request access to AI0 just after the other While Loop was granted access but before AI0 finishes executing. Because LabVIEW can allow only one accessor at a time, LabVIEW uses arbitration to ensure sequential access to the shared resource.

By default, LabVIEW performs arbitration for all shared resources. However, you can customize the arbitration options for FPGA Device I/O functions if you need to optimize the FPGA VI. The default arbitration option varies according to the type of shared resource.

Note The arbitration process can take several clock cycles to execute. Arbitration takes additional time and FPGA space and can add jitter to an application.

Arbitration Options

The following arbitration options are available with the FPGA Module:

- **Normal**
- **Normal (Optimize for Single Accessor)**
- **None**

An arbiter performs the following general steps during arbitration.

1. Waits for one or more requestors. If multiple requestors request access, the arbiter determines which requestor becomes the accessor.
2. Passes data from the accessor to the resource.
3. Begins resource execution.
4. Waits for the resource to complete execution.
5. Passes data back to the accessor.
6. Prepares the resource for another execution.
7. Waits for the next requestor.

Normal

A resource with the **Normal** arbitration option always uses an arbiter, even if only one requestor requests access. The **Normal** arbiter is a fair round robin arbiter that ensures sequential access to a shared resource. The arbiter does not allow a requestor to become an accessor again until all other waiting requestors have become accessors. Consequently, jitter occurs if you have more than one simultaneous requestor. Refer to the *Jitter* section for more information.

Normal (Optimize for Single Accessor)

A resource with the **Normal (Optimize for Single Accessor)** option does not use an arbiter if the FPGA VI contains only one requestor. If the FPGA VI has multiple requestors, LabVIEW uses **Normal** arbitration even if the requests are not simultaneous. You can save time and space in FPGA VIs if you use the **Normal (Optimize for Single Accessor)** arbitration option if the FPGA VI contains only one requestor.

Use the **Normal (Optimize for Single Accessor)** option in the following situations:

- You have a large FPGA VI and need to save space.
- You have only one accessor for a resource.

- You do not need single requestor channels synchronized with multiple requestor channels. Refer to the *Timing* section for information about synchronized channels.

None

A resource with the **None** option does not arbitrate simultaneous requests, which saves significant space on the FPGA. To use the **None** option, you must guarantee sequential access to the resource in the data flow of the FPGA VI. If you attempt to make simultaneous requests in the FPGA VI, you make simultaneous accesses and corrupt data.

Available Arbitration Options for Specific Resources

You can select arbitration options for most FPGA Device I/O resources, as shown in Table 3-1. D is the default option, and O indicates other available options.

Table 3-1. Arbitration Options for I/O

Arbitration Option	Analog Input	Analog Output	Digital Input	Digital Output	Digital Enable	Digital Data
Normal Arbitration	D	D	—	D	D	D
Normal (Optimize for Single Accessor)	O	O	—	O	O	O
None	—	O	D	O	O	O

Use the default I/O arbitration options for most applications. You can change arbitration options to optimize some designs. To access arbitration options, double-click or right-click the function icon on the block diagram and select **Properties** from the shortcut menu. Select an arbitration option on the **Arbitration** tab of the FPGA Device I/O function **Configure** dialog box. Each configured **Alias** is associated with an **Arbitration** option. References to the same **Alias** in the block diagram have the same arbitration options. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for more information about configuring aliases.

None is the only arbitration option available for the Digital Input and Digital Port Input functions. In addition to minimizing FPGA usage, the **None** option allows the Digital Input and Digital Port Input functions to execute in a single clock cycle. Use the **None** option to minimize FPGA

usage and allow single clock cycle execution for the other digital I/O functions.

Note Use the arbitration options with caution. Incorrect use can cause incorrect execution of a block diagram or unintended data. For example, use the **None** option for an analog output only if you are certain that the resource is not accessed from two functions or VIs at the same time. If you do access the shared resource from two locations simultaneously, the data presented to the resource is the logical OR of the data from the individual requestors.

Shared resources other than FPGA Device I/O resources—such as interrupts, non-reentrant VIs, global variables, written local variables, and Memory VIs—use the **Normal (Optimize for Single Accessor)** arbitration option. You cannot change the arbitration option for shared resources other than FPGA Device I/O resources.

Note The default mode for subVIs in LabVIEW is non-reentrant, but you might need reentrant subVIs for parallel execution and no arbitration. Refer to Chapter 2, *Creating FPGA VIs*, for information about reentrant subVIs.

Jitter

Jitter occurs if a requestor is delayed in becoming an accessor due to resource contention with one or more additional requestors. For example, you might have an application performing a timed While Loop that samples analog input at a fixed rate. Each time the Analog Input function executes, the function becomes an accessor as soon as it requests the analog input resource. If you add a second timed While Loop that samples the same analog input resource, the two Analog Input functions might simultaneously request the analog input resource. In this case, the arbiter delays one of the requestors while allowing the other requestor to become an accessor. The delayed requestor has jitter because the access does not occur immediately after the request was made.

To avoid jitter, design the FPGA VI block diagram to make sure a requestor does not access the shared resource when the shared resource is busy or to make sure two requests do not occur during the same clock cycle. Jitter occurs most often when you have a shared local variable with multiple writers or a shared subVI from two independently running loops or unrelated parts of the VI as shown in Figure 3-2.

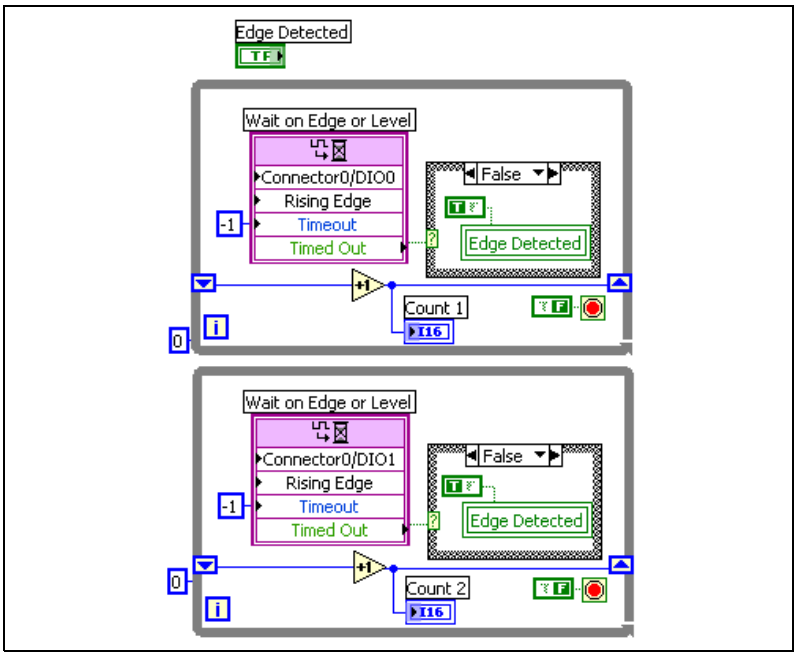

Figure 3-2. Arbitration Jitter

The VI in Figure 3-2 shows two While Loops that might attempt to write to the **Edge Detected** local variable simultaneously. The arbiter allows one While Loop to access **Edge Detected** at a time. The other While Loop does not access **Edge Detected** until after the first While Loop finishes. Jitter is introduced into the delayed While Loop.

The possibility of jitter grows with the number of accessors. If you do not schedule simultaneous requests, the delay through the arbiter is constant regardless of the number of potential accessors.

Timing

Not all arbitration options take the same amount of time to execute. If you want accesses to multiple resources of the same type to occur simultaneously, you must choose arbitration options for each resource that take the same amount of time to execute. Figure 3-3 illustrates an FPGA VI that might have a timing problem depending on the arbitration options selected.

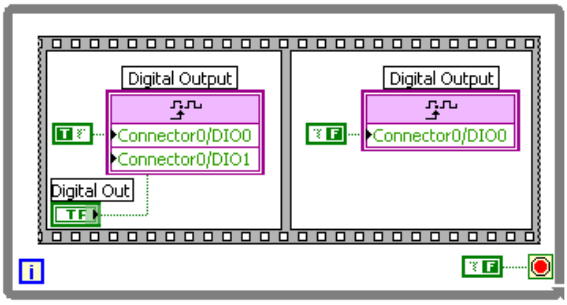

Figure 3-3. Arbitration Timing

The Digital Output functions shown in Figure 3-3 have three arbitration options. If you choose the **Normal** arbitration option, LabVIEW implements an equivalent arbiter for both Connector0/DIO0 and Connector0/DIO1. Both arbiters take the same amount of time to execute, so Connector0/DIO0 and Connector0/DIO1 output simultaneously in the first frame of the Flat Sequence structure.

If you choose the **Normal (Optimize for Single Accessor)** arbitration option, LabVIEW implements a different arbiter for each of Connector0/DIO0 and Connector0/DIO1. Connector0/DIO0 uses a normal arbiter because the block diagram requests access to Connector0/DIO0 twice. Connector0/DIO1 uses no arbiter because the block diagram requests access to Connector0/DIO1 only once. Connector0/DIO0 takes longer to execute than Connector0/DIO1, so Connector0/DIO0 and Connector0/DIO1 do not output simultaneously in the first frame of the Flat Sequence structure.

If you choose the **None** arbitration option for both Digital Output functions, LabVIEW does not implement an arbiter for either Connector0/DIO0 or Connector0/DIO1. Therefore, Connector0/DIO0 and Connector0/DIO1 output simultaneously.

FPGA Utilization

Arbitration also can use a significant amount of space on the FPGA. If you can decrease the number of requestors of a resource to one, use the **Normal (Optimize for Single Accessor)** arbitration option. The single requestor requires no arbitration. If you have two requestors, LabVIEW uses **Normal** arbitration, even if **Normal (Optimize for Single Accessor)** is selected.

4

Running FPGA VIs

This chapter describes compiling, downloading, and running FPGA VIs, as well as FPGA device configuration options.

Compiling FPGA VIs

You can compile an FPGA VI by clicking the **Run** button while targeted to an FPGA device or by clicking the **Build** button in the **FPGA Project Builder** dialog box. You also can compile an FPGA VI without running the FPGA VI by clicking <Ctrl>-**Run** while targeted to an FPGA device. Compiling FPGA VIs can take from a few minutes to a few hours.

Before you can run an FPGA VI on an FPGA device, the LabVIEW FPGA Compile Server must convert the VI to a bitstream that LabVIEW can download to the FPGA device. The LabVIEW FPGA Compile Server executes independently of the LabVIEW development system, so you can run it on a remote computer.

LabVIEW prompts you to compile new or changed FPGA VIs. If you made only cosmetic changes to the FPGA VI and you did not change the front panel controls and indicators, you do not need to recompile the FPGA VI. Click the **Use Old Bitstream** button when the **Warning: Beginning compile for FPGA** dialog box appears to avoid a new compile of an already compiled FPGA VI. If you make non-cosmetic changes to the FPGA VI and do not recompile, the most recently compiled FPGA VI downloads and runs but you might receive incorrect results.

You can compile FPGA VIs if an FPGA device is not installed. However, you cannot run the FPGA VI or use an emulator without an FPGA device installed.

You can test an FPGA VI before compiling it. Refer to Chapter 6, *Debugging FPGA VIs*, for information about testing FPGA VIs using emulators.

Compiling FPGA VIs Using the LabVIEW FPGA Compile Server

LabVIEW and the LabVIEW FPGA Compile Server have a client-server architecture, where LabVIEW is a client to the LabVIEW FPGA Compile Server. The client-server architecture allows you to disconnect LabVIEW from the LabVIEW FPGA Compile Server during a compile. You then can continue to use LabVIEW while the FPGA VI compiles. You must not modify the FPGA VI being compiled. To reconnect, you again run the FPGA VI targeted to the FPGA device. LabVIEW displays a compile report when the compile is complete. You can view the compile report if you are connected to the LabVIEW FPGA Compile Server. After you click the **OK** button in the **Successful Compile Report** window, LabVIEW embeds the new bitstream into the FPGA VI and downloads the bitstream to the FPGA. The FPGA VI then runs on the FPGA device and you can interact with it through the front panel using Interactive Front Panel Communication.

The LabVIEW FPGA Compile Server launches automatically when you run an FPGA VI that is not compiled or has been modified since the last compile. LabVIEW converts the VI into intermediate files to send to the LabVIEW FPGA Compile Server. The LabVIEW FPGA Compile Server converts the intermediate files into a bitstream.

The compile time depends on the size of the VI, the processor speed, and amount of memory in the computer on which you are compiling. National Instruments recommends at least 512 MB of memory for the LabVIEW FPGA Compile Server. If you have less memory, smaller block diagrams might compile quickly, but larger block diagrams might use large amounts of virtual memory, which can be very slow, and compiles can take over ten times longer to complete.

The LabVIEW FPGA Compile Server does not close automatically. You can close it by clicking the **Stop Server** button.

Compiling on a Remote Computer

You can install the LabVIEW FPGA Compile Server on a remote computer. You might want to do this if your development computer is slow and does not have enough memory to compile for the FPGA device. By default, LabVIEW assumes the LabVIEW FPGA Compile Server is installed on the local computer. To select a remote LabVIEW FPGA Compile Server, select **Tools»FPGA Target Options** when you target the FPGA device and enter the name or IP address and server port of the remote computer running the LabVIEW FPGA Compile Server. Depending on the network, you also might need to increase the network timeout.

Launch the LabVIEW FPGA Compile Server manually on the remote computer by selecting **Start»Programs»National Instruments» LabVIEW»LabVIEW FPGA Utilities»CompileServer**. You must launch the LabVIEW FPGA Compile Server manually when LabVIEW clients on other computers are configured to connect to the remote computer for compiling.

Managing Compilation Files

The LabVIEW FPGA Compile Server stores all files it uses to compile a VI in a directory. Configure the directory by clicking the **Configure** button in the **LabVIEW FPGA Compile Server** window. Click the **Compile List** button to view the compile history and delete compile files you no longer need. Typically, you do not need compile files after the bitstream is embedded in the FPGA VI. Delete compile files that you no longer need to save space on your hard drive.

Using Compiled FPGA VI Options

This section describes the clock rate and auto run options available with the FPGA Module to compile into FPGA VIs. Each type of FPGA device might have specific options available. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for more information about device-specific options.

Changing the FPGA Device Clock Rate

The FPGA device provides a 40 MHz clock to control the internal operations on the FPGA. The internal clock determines the execution time of the individual functions and VIs on the FPGA VI block diagram. Most FPGA VIs can execute properly using this clock. You also can compile FPGA VIs with faster clock rates for higher performance. However, not all FPGA VIs can compile properly with faster clock rates. If you select a clock rate that is too fast for the FPGA VI, the compile report window tells you the compile failed. You must select a lower clock rate and try the compile again.

Change the global default clock rate for an FPGA device by selecting **Tools»FPGA Target Options**. All subsequent FPGA VIs you create for that FPGA device have the new default clock rate. You can change the clock rate for a specific FPGA VI by selecting **Tools»Build for FPGA**. The clock rate for the specific FPGA VI is compiled into the bitstream. Each time you compile the same FPGA VI, the FPGA VI retains the same clock rate.

Configuring FPGA VIs to Run Automatically

Some FPGA devices have flash memory that can store FPGA VIs. If you want a VI that is loaded to the FPGA from flash memory to run automatically on an FPGA device, change the **Default Auto Run** option for the FPGA device by selecting **Tools»FPGA Target Options**. All subsequent FPGA VIs you create for that FPGA device have the new **Default Auto Run** setting. You can change the **Auto Run VI** option for every instance of a specific FPGA VI by selecting **Tools»Build for FPGA**. Refer to the *Running FPGA VIs at Power On* section for information about storing the FPGA VI in flash memory.

Downloading FPGA VIs to the FPGA Device

When you click the **Run** button on a new or changed FPGA VI while targeting an FPGA device, LabVIEW downloads the FPGA VI to the FPGA automatically after the compile completes. LabVIEW automatically downloads a previously compiled VI when you target an FPGA device and click the **Run** button. LabVIEW does not download the FPGA VI if it is already on the FPGA device. You can force a download by selecting **Operate»Download Application**. You might force a download if you want to reinitialize the FPGA to its default state.

When you target LabVIEW to Windows or an RT target, you can programmatically download FPGA VIs to FPGA devices. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for information about programmatically downloading FPGA VIs.

Running FPGA VIs

After you compile and download an FPGA VI, you can run the FPGA VI on the targeted FPGA device. When you click the **Run** button, the FPGA VI runs using Interactive Front Panel Communication.

Note If you click the **Run** button and are not targeted to an FPGA device, LabVIEW generates random data for the FPGA Device I/O functions.

If you want to close LabVIEW but leave the FPGA VI running, select **File»Exit without closing FPGA VIs**. If you later restart LabVIEW, you can reconnect to the running FPGA VI by opening the FPGA VI, targeting LabVIEW to the FPGA device, and clicking the **Run** button.

You can build host VIs to programmatically read and write to the front panel of the FPGA VI by targeting LabVIEW to Windows or an RT target. Refer to Chapter 5, *Communicating with FPGA VIs*, for more information.

After you run the FPGA VI, you might need to debug the block diagram. Refer to Chapter 6, *Debugging FPGA VIs*, for more information.

Running FPGA VIs at Power On

You can store FPGA VIs on the flash memory of certain FPGA devices. You can configure the FPGA device to automatically load the FPGA VI from flash memory into the FPGA when the FPGA device powers on. Select **Tools»FPGA Utilities»Flash VI or Setup Board** to store the FPGA VI on the flash memory and to configure the FPGA VI to load when the FPGA device powers on.

You must use this feature when you want the FPGA VI to run automatically when the FPGA device is first powered on or after a power failure. Refer to the *Configuring FPGA VIs to Run Automatically* section for more information.

Setting Target Configurations

Some FPGA devices have configuration information stored in flash memory. Select **Tools»FPGA Utilities»Flash VI or Setup Board** to configure the flash memory options.

For example, the NI PXI-7831R stores two configuration options in flash memory—**Sync to PXI Clock** and **Analog Signal Connection**. The FPGA clock source is internal by default, or you can synchronize the FPGA device to the 10 MHz clock of a PXI chassis. Use this feature when you want multiple FPGA devices to synchronize the FPGA device clocks to the same PXI clock. Refer to the hardware manual for more information about the configuration options.

5

Communicating with FPGA VIs

Use the FPGA Interface functions to communicate with an FPGA VI from a host VI. A host VI is a VI that communicates with the FPGA VI to control the FPGA device. A host VI can run in Windows or on an RT target.

To write host VIs, you first target LabVIEW for Windows or an RT target. Refer to the *LabVIEW User Manual* for information about developing VIs that run on Windows. Refer to the *LabVIEW Real-Time Module User Manual* for information about developing VIs for RT targets.

You can use the FPGA Interface functions to programmatically control and communicate with an FPGA VI. Use the FPGA Interface functions to perform the following operations in host VIs:

- Establishing communication with the FPGA VI.
- Reading and writing data to the FPGA VI.
- Waiting for and acknowledging FPGA VI interrupts.
- Closing the VI reference.

Establishing Communication with the FPGA VI

You must open a reference to the FPGA VI before you can communicate with the host VI. Use the Open FPGA VI Reference function, available when you target LabVIEW for Windows or an RT target, to do the following:

- Select the FPGA VI with which the host VI communicates.
- Select the FPGA device on which the FPGA VI runs.
- Determine whether the host VI opens and runs the FPGA VI or just opens the FPGA VI.

Selecting the FPGA VI

To select the FPGA VI, right-click the Open FPGA VI Reference icon on the block diagram, and select **Select Target VI** from the shortcut menu. Type the path or navigate to the FPGA VI on the host computer.

Selecting the FPGA Device

You can select the FPGA device when you create the host VI. Right-click the Open FPGA VI Reference icon on the block diagram and select **FPGA** from the shortcut menu to display the **Select Board** dialog box. Choose from the list of available FPGA devices.

You also can programmatically select an FPGA device when you use the Open FPGA VI Reference function using the **VISA Resource Name** input. Right-click the Open FPGA VI Reference icon on the block diagram and select **External VISA Input** from the shortcut menu to add the **VISA Resource Name** input to the Open FPGA VI Reference function.

LabVIEW lists the FPGA devices in the **Select Board** dialog box using relative addressing. Relative addressing shows the current assignments of FPGA devices to labels. Each FPGA device is labeled RIO0, RIO1, RIO2, and so on. If you select RIO1, the Open FPGA VI Reference function downloads the FPGA VI to the FPGA device currently labeled RIO1 in the **Select Board** dialog box. If you then remove the FPGA device currently labeled RIO1 from the system, LabVIEW reassigns the labels so that the FPGA device that was labeled RIO2 becomes RIO1, RIO3 becomes RIO2, and so on. The Open FPGA VI Reference function now downloads the FPGA VI to the new RIO1 FPGA device. You can use absolute addressing or relative addressing with the **VISA Resource Name** input.

To use absolute addressing, use a **VISA Resource Name Constant** and click the down arrow with the Operating tool to select an absolute address from the pull-down menu. To use relative addressing, right-click inside the Open FPGA VI Reference function and select **FPGA** from the shortcut menu. You can select an available board in the **Select Board** dialog box that appears. You also can use the **VISA Resource Name Constant** or a **String Constant** and type the relative address in the constant.

Setting Open and Run Options

When you run the host VI, the Open FPGA VI Reference function creates the reference and checks the specified FPGA device to determine if the FPGA VI is on the FPGA device. If the selected FPGA VI does not match the VI on the FPGA device, the Open FPGA VI Reference function downloads the selected FPGA VI to the FPGA device. You can choose to open the reference and run the FPGA VI by right-clicking the Open FPGA VI Reference icon on the block diagram and selecting **Open and Run** from the shortcut menu. **Open and Run** is the default behavior.

You also can choose to open the reference without running the FPGA VI by right-clicking the Open FPGA VI Reference icon on the block diagram and selecting **Open** from the shortcut menu. If you open an FPGA VI reference without running the FPGA VI, you can use the Invoke Method function from the host VI to programmatically run the FPGA VI. You also can use the Invoke Method function to download and abort FPGA VIs. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for more information about the Open FPGA VI Reference and Invoke Method functions.

Reading and Writing Data to the FPGA VI

A host VI can control and monitor only data passed through the FPGA VI front panel. You cannot access any wires on the block diagram that do not have controls or indicators. First use the Open FPGA VI Reference function to open a reference to the FPGA VI. Then wire the **HW Exec Ref Out** parameter to the Read/Write Control function to access controls and indicators on the FPGA VI. You can read indicators and write controls. You also can write indicators and read controls. You can expand the Read/Write Control function to read or write multiple controls and indicators. When you run the host VI, the Read/Write Control function reads and writes controls and indicators in the order they appear in the function icon on the block diagram.

The Read/Write Control function supports scalar data, such as numeric controls, and complex data, such as arrays and clusters. You can program the FPGA VI to bundle scalar data into arrays or clusters and then read the arrays or clusters of data as a single block with the host VI to make sure all data is read at a single time. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for information about arrays and clusters.

Note The FPGA Module creates a register map specific to the FPGA VI that includes a hardware register for every control and indicator. LabVIEW uses the register maps internally to communicate with the FPGA VI for both Interactive Front Panel Communication and Programmatic FPGA Interface Communication.

Responding to FPGA VI Interrupts

You can generate interrupts from the FPGA VI to notify the host VI of events, such as data being ready, an error occurring, or a task finishing. Use the Invoke Method function in a host VI to interact with an interrupt on an FPGA device. You can wait on interrupts with the Wait on IRQ method. You also can acknowledge interrupts with the Acknowledge IRQ method. Refer to Chapter 2, *Creating FPGA VIs*, for information about generating interrupts in FPGA VIs.

Waiting for Interrupts

Use the Wait on IRQ method in the host VI to wait for logical interrupts generated by the FPGA VI. For example, you might want the host VI to perform independent operations while the FPGA VI collects data. You can use the Wait on IRQ method in the host VI to detect when the FPGA VI data is ready and the host VI can retrieve the data.

You can use arrays with the Wait on IRQ method if you want to wait on multiple logical interrupts. Depending on the logical interrupt or combination of logical interrupts, the host VI can execute different code to handle the interrupts. You also can generate occurrences in the code that enable other parts of the host VI.

Use only one Wait on IRQ method at a time per FPGA device in a host VI. For example, if you control two FPGA devices with one host VI, you can use two simultaneous Wait on IRQ methods, one for each device.

Note You cannot use more than one Wait on IRQ method at a time in a time-critical VI running on an RT target. When you use the Wait on IRQ method in a time-critical priority thread, the entire thread sleeps while the Wait on IRQ method waits on an interrupt.

Acknowledging Interrupts

You must use the Acknowledge IRQ method to acknowledge the logical interrupts returned by the Wait on IRQ method to allow the FPGA VI to assert those interrupts again. Acknowledge the interrupt only when the host VI is ready to accept the next interrupt from the FPGA VI. If you use the **Wait Until Cleared** option on the Interrupt VI in the FPGA VI, acknowledging the interrupt allows the FPGA VI to continue executing. For example, you might not want to acknowledge the interrupt until a certain value is written to the FPGA device.

Closing a Reference to the FPGA VI

Use the Close FPGA VI Reference function to close the reference to the FPGA VI after you finish communicating with the FPGA VI from the host VI. Close the VI reference before exiting the host VI.

By default, the Close FPGA VI Reference function aborts the FPGA VI and closes the FPGA VI reference. You can close the FPGA VI reference without aborting the FPGA VI by right-clicking the Close FPGA VI Reference function icon on the block diagram and selecting **Close** from the shortcut menu. If you do not abort the FPGA VI, the FPGA VI continues executing on the FPGA device after you close the FPGA VI reference.

6

Debugging FPGA VIs

This chapter describes debugging techniques you can use to test FPGA VIs. You can use traditional LabVIEW debugging techniques only when you target the FPGA VI to an emulator or run the FPGA VI on the host computer. You cannot use traditional LabVIEW debugging techniques, such as execution highlighting and probing, with LabVIEW targeted to an FPGA device.

Refer to the *LabVIEW User Manual* for information about traditional LabVIEW debugging techniques.

Testing a VI Before Compiling

You can test the logic of an FPGA VI before compiling it by targeting an emulator. To target an emulator rather than the FPGA device, select the execution target that matches the FPGA device. For example, if the device appears in the **Operate»Switch Execution Target** menu as **FPGA Device: 78XX**, the emulator for that device appears in the same menu as **FPGA Device: 78XXEmulator**. You can use an emulator with any available FPGA device.

When you run an FPGA VI with an emulator, LabVIEW downloads the pre-compiled emulation VI included with the FPGA Module to the FPGA device to provide I/O, and the FPGA VI runs on the host computer. LabVIEW then communicates with the emulation VI on the FPGA while both VIs run. The FPGA Module includes a pre-compiled emulation VI for every FPGA device target.

You can use all traditional LabVIEW debugging tools, such as probes, execution highlighting, breakpoints, and single-stepping. You cannot test certain behavior, such as timing and determinism, with an emulator because the FPGA VI runs on the host computer instead of the FPGA. The emulator tries to preserve the timing of the Loop Timer, Wait, and Tick Count VIs as much as possible. Other functions and VIs execute as quickly as possible on the host computer.

Note You must have an FPGA device installed to use an emulator. However, you can still debug the FPGA VI without an FPGA device by targeting LabVIEW for Windows. When you run an FPGA VI while targeting LabVIEW for Windows, LabVIEW generates random data for the FPGA Device I/O functions.

Building Debugging into an FPGA VI

In addition to using emulators, you can build debugging functionality into the FPGA VI with additional indicators or additional I/O. You can use additional indicators and additional I/O with Interactive Front Panel Communication or Programmatic FPGA Interface Communication.

Adding Indicators

You can add indicators to the FPGA VI block diagram to monitor the internal state of the FPGA VI. Use indicators as you would probes. Place them anywhere on the block diagram where you need to see data to verify the functionality of the VI. You also can perform more advanced debugging by using controls to change execution of the FPGA VI.

If you use additional indicators with Programmatic FPGA Interface Communication, you must program the host VI to read the additional indicators.

Note Adding indicators to the FPGA VI takes up more space on the FPGA. Be sure to remove debugging indicators if you encounter space constraints on the FPGA.

Note An indicator consumes a small amount of execution time, which can affect the performance of the FPGA VI. Whenever possible, add debugging indicators in parallel to other operations in the FPGA VI to minimize the effect on execution time.

Adding I/O

If you have unused I/O resources on the FPGA device, you can add additional I/O terminals to the FPGA VI block diagram to aid debugging. You can easily monitor the internal state of Boolean logic, triggers, and so on. You can create more advanced debugging tools by adding LabVIEW code to analyze data and events and to control flow.

A

Technical Support and Professional Services

Visit the following sections of the National Instruments Web site at ni.com for technical support and professional services:

- **Support**—Online technical support resources include the following:
  - **Self-Help Resources**—For immediate answers and solutions, visit our extensive library of technical support resources available in English, Japanese, and Spanish at ni.com/support. These resources are available for most products at no cost to registered users and include software drivers and updates, a KnowledgeBase, product manuals, step-by-step troubleshooting wizards, conformity documentation, example code, tutorials and application notes, instrument drivers, discussion forums, a measurement glossary, and so on.
  - **Assisted Support Options**—Contact NI engineers and other measurement and automation professionals by visiting ni.com/support. Our online system helps you define your question and connects you to the experts by phone, discussion forum, or email.
- **Training**—Visit ni.com/custed for self-paced tutorials, videos, and interactive CDs. You also can register for instructor-led, hands-on courses at locations around the world.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, NI Alliance Program members can help. 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.

Glossary

Symbol	Prefix	Value
p	pico	10^{-12}
n	nano	10^{-9}
µ	micro	10^{-6}
m	milli	10^{-3}
k	kilo	10^{3}
M	mega	10^{6}
G	giga	10^{9}
T	tera	10^{12}

A

accessor — A hardware component that has been granted access to a specific shared resource by an arbiter.

alias — A user defined name for an I/O terminal, displayed on the FPGA Device I/O function icon on the block diagram. For example, you can create an alias for AI0 named **Oven Temperature** that appears in the Analog Input function icon on the block diagram. The complete list of aliases created in an FPGA VI appears in the **Configure** dialog box for each FPGA Device I/O function. Aliases configured in one FPGA VI do not appear in other FPGA VIs.

ADC — Analog-to-digital converter—an electronic device, often an integrated circuit, that converts an analog voltage to a digital number.

arbiter — A hardware component that controls access to a shared resource and determines which requestor becomes the accessor of the shared resource. The arbiter resolves resource contention over the shared resource.

arbitration — The process of resolving resource contention by determining which requestor of a shared resource is granted access to the shared resource.

B

bitstream	Programming information that is downloaded to an FPGA device to determine its behavior.

C

compile for FPGA	The process of creating a bitstream from an FPGA VI.

D

DAC	Digital-to-analog converter—an electronic device, often an integrated circuit, that converts a digital number into a corresponding analog voltage or current.
determinism	Characteristic of a system that describes how consistently it can respond to external events or perform operations within a given time limit.
development computer	The computer on which you develop LabVIEW VIs. The VIs can run on different execution targets.
device	An instrument or controller you can access as a single entity that controls or monitors real-world I/O points. A device often is connected to a host computer through some type of communication network.

E

emulation VI	The VI that the emulator downloads to the FPGA device so you can test and debug an FPGA VI without compiling and downloading the FPGA VI.
emulator	A target you can select from the **Switch Execution Target** list that mimics the behavior of an FPGA device. The emulator runs the FPGA VI on the host computer and accesses the emulation VI running on the FPGA device to provide real I/O. You can use the emulator to test and debug FPGA VIs without compiling and downloading the FPGA VIs.
execution target	The computer or device that runs a LabVIEW VI. An execution target can be an FPGA device, an RT target, or the development computer.

F

flash memory	Non-volatile storage that retains its contents even when the device powers off.
FPGA	Field Programmable Gate Array. A programmable logic device (PLD) with a high density of gates.
FPGA device	A Reconfigurable I/O device that contains a reconfigurable FPGA surrounded by fixed I/O resources.
FPGA Interface function	A type of function that enables communication between a host VI and an FPGA VI. Available with the FPGA Module targeted to LabVIEW for Windows or an RT target.
FPGA VI	A VI that is downloaded to the FPGA device that determines the functionality of the hardware.

H

host computer	The computer that controls and monitors the FPGA device.
host VI	A VI that runs in software on the host computer and controls and monitors the FPGA VI on the FPGA device using FPGA Interface functions.

I

Interactive Front Panel Communication	A method of communicating with the FPGA VI that allows you to interact directly with the FPGA VI front panel controls and indicators. The front panel of the FPGA VI displays on the host computer while the block diagram executes on the FPGA device.
interrupt	A hardware signal that allows a peripheral device to alert the host computer to perform some action.
I/O	Input/output—the transfer of data to/from a computer system involving communications channels, operator interface devices, and/or data acquisition and control interfaces.

J

jitter	The amount of time that the loop cycle time varies from the desired time.

L

logical interrupt	A hardware alert that allows multiple interrupt sources to simply and efficiently share a single hardware interrupt in an application.

N

non-reentrant VI	A subVI that occurs as a single instance shared among multiple callers.

O

operating system	Base-level software that controls a computer, runs programs, interacts with users, and communicates with installed hardware or peripheral devices.

P

power-on state	The state at which a device is set when the system powers on.
Programmatic FPGA Interface Communication	A method of communicating with the FPGA VI that allows you to use a host VI to communicate programmatically with an FPGA VI using the FPGA Interface functions. LabVIEW on the host computer communicates directly with LabVIEW on the FPGA device.
PWM	Pulse width modulation. Typically refers to a signal whose high period and low period can be varied in a controlled fashion.

R

real time	A property of an event or system in which data is processed with high determinism as it is acquired instead of being accumulated and processed at a later time.
reentrant VI	A subVI that replicates itself for each caller.

register	A location in hardware on the FPGA device that you can read or write to pass data between the FPGA device and the host computer. Every control and indicator in an FPGA VI has an associated register.
register map	A collection of registers that define the hardware interface for communicating between the host computer and an FPGA device.
requestor	A LabVIEW or hardware component that has requested access to a shared resource.
resolution	The smallest signal increment that can be detected by a measurement system. Resolution can be expressed in bits, in proportions, or in percent of full scale. For example, a system has 12-bit resolution, one part in 4,096 resolution, and 0.0244% of full scale.
resource	A hardware component that can be accessed from a block diagram. A resource might be a component connected to the FPGA, such as an ADC or DAC. It also can be a component within the FPGA, such as FPGA memory or a local variable.
resource contention	A situation that occurs when two requestors simultaneously attempt to access a shared resource or when a requestor attempts to access a resource that is currently in use by an accessor.
round robin arbitration	An arbitration scheme where no requestor has priority over any other requestors. The current accessor does not become an accessor again until any other pending requestors have become accessors.
RT target	A National Instruments RT Series device you can target to run host VIs.

T

terminal	A specific I/O resource on an FPGA device, such as Connector0/DIO0.

V

VI	*See* virtual instrument (VI).
virtual instrument (VI)	Program in LabVIEW that models the appearance and function of a physical instrument.