

# Getting Started with the NI PCI-5640R IF Transceiver and the LabVIEW FPGA Module

This guide demonstrates how to develop an NI PCI-5640R application using the LabVIEW FPGA Module. This document provides step-by-step instructions for using an NI PCI-5640R hardware module with the LabVIEW FPGA Module to make a simple spectrum analyzer. While developing this example solution, you will learn the concepts and techniques necessary to develop your LabVIEW FPGA NI PCI-5640R application.

This document assumes you are familiar with the LabVIEW FPGA graphical programming environment. Refer to the *Getting Started with LabVIEW* manual for more information about the basic LabVIEW features you use to build VIs and projects. For the most current versions of documentation, visit [ni.com/manuals](http://ni.com/manuals).



**Note** The NI-5640R software also includes an instrument driver with an easy-to-use LabVIEW API to create an application for your NI PCI-5640R hardware. NI-5640R instrument driver VIs are *not* supported in the LabVIEW FPGA Module. For more information about using the instrument driver, refer to the *NI 5640R Help* and the *NI 5640R IF Transceiver Getting Started Guide*.

## Contents

---



Conventions .....	3
What You Need to Get Started .....	4
Fundamentals of Reconfigurable Input/Output (RIO) .....	4
Reconfigurable I/O Concept .....	4
User-Defined I/O Resources .....	4
Device-Embedded Logic and Processing .....	5
Reconfigurable I/O Architecture .....	5
Reconfigurable I/O Applications .....	6
LabVIEW FPGA Module .....	7
Xilinx Virtex II Pro FPGA .....	7

Reconfigurable FPGA Resources.....	7
Clocking Block Diagram .....	8
Configuring IQ Clock Rates .....	9
I/O Resources .....	10
I/O Resources Available using the FPGA VI.....	11
Configuration I/O Resources .....	12
Data I/O Resources .....	12
Getting Started with FPGA VIs and Host VIs.....	12
NI-5640R FPGA Examples.....	14
The LabVIEW Project.....	14
Exploring an FPGA VI.....	15
Exploring a Host VI.....	18
Creating an FPGA Application.....	19
Overview of the Application .....	19
Using the ni5640R Template.....	20
Configuring the NI PCI-5640R Target.....	22
Developing the FPGA VI .....	23
Compiling the FPGA VI.....	26
Developing the Host VI.....	27
What You Have Learned .....	34
Using FPGAs for Signal Processing.....	34
Using Single-Cycle Timed Loops .....	34
Understanding Clock Domains.....	35
Pipelining.....	35
FPGA Signal Processing References.....	36
Tips and Troubleshooting .....	37
It takes a long time to access controls and indicators in the Read/Write Control.....	37
I get a pop-up indicating that an error was detected in the communication between the host computer and the FPGA target.....	37
When I open a Host VI, LabVIEW cannot find a file named filename.lvbit.....	38
When I Run an FPGA VI, a compilation error instantly occurs.....	38
Where to Go From Here .....	39

# Conventions

---

The following conventions are used in this document:

- » 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 note, which alerts you to important information.
-  This icon denotes a caution, which advises you of precautions to take to avoid injury, data loss, or a system crash. When this symbol is marked on the hardware product, refer to the *Read Me First: Safety and Radio-Frequency Interference* document, included with the device, for precautions to take.
- bold** Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names.
- italic* Italic text denotes variables, emphasis, a cross-reference, or an introduction to a key concept. Italic text also denotes text that is a placeholder for a word or value that you must supply.
- `monospace` Text in this font denotes text or characters that you 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.
- monospace italic* Italic text in this font denotes text that is a placeholder for a word or value that you must supply.
- NI PCI-5640R The NI PCI-5640R IF transceiver hardware module.
- NI-5640R The software you use to program the NI PCI-5640R IF transceiver hardware module.
- RIO device A device based on a reconfigurable FPGA core surrounded by fixed resources. The NI PCI-5640R is a RIO device.

# What You Need to Get Started

---

Your system must meet certain requirements to use the NI PCI-5640R with LabVIEW FPGA. For more information on minimum system requirements and supported application development environments (ADEs), refer to the *NI-5640R Readme*, which is available on the NI-5640R CD.



**Note** After you install NI-5640R, you can access the *NI-5640R Readme* at **Start» All Programs»National Instruments»NI-5640R»Documentation**.

## Fundamentals of Reconfigurable Input/Output (RIO)

---

This section introduces the concept of Reconfigurable Input/Output (RIO) and describes how to use the reconfigurable FPGA to build high-level functions in hardware. Refer to the *NI PCI-5640R Hardware Overview* section in the *NI 5640R Help* for descriptions of the physical input/output (I/O) resources available on the NI PCI-5640R.

### Reconfigurable I/O Concept

The NI PCI-5640R device is based on a reconfigurable FPGA core surrounded by fixed resources. Unlike traditional IF digitizer/source devices that use a fixed core with predetermined functionality, you can configure the behavior of the FPGA core in the NI PCI-5640R to closely match the requirements of your system. The behavior of this module is fully user-defined, and can be implemented as a VI, creating an application-specific I/O device.

Flexible functionality allows a RIO device, such as the NI PCI-5640R, to match individual application requirements. For example, you can process the data from the two A/D converters (ADCs) as I and Q components from a single RF-to-complex-IF demodulator (single antenna-to-bits receive chain) or as two IF channels representing a two-channel, multiple-input-multiple-output (MIMO) (two antenna-to-bits receive chain). Another example is implementing a new modulation/demodulation scheme that may be difficult to achieve with standard off-the-shelf IF transceivers.

### User-Defined I/O Resources

With the RIO device, you can define both the combination of I/O resources and the I/O resources themselves. You can also create new building blocks on top of fixed I/O resources. For example, one application might require an event counter that increments when a rising edge appears on any of three

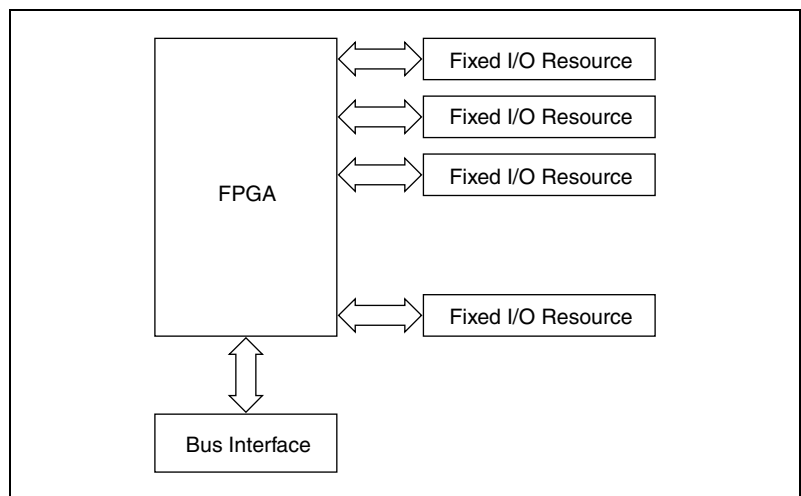
digital input lines. Another application might require a digital line to be asserted when an analog input exceeds a programmable threshold. You can implement these user-defined behaviors in the hardware for fast, deterministic performance.

## Device-Embedded Logic and Processing

You can embed logic and digital signal processing in the FPGA of the RIO device. Typical logic functions include Boolean operations, comparisons, and basic mathematical operations. For example, the FPGA can demodulate and decode a baseband signal. You can implement multiple functions efficiently in the same design, operating sequentially or in parallel. You can implement multiple signal processing blocks on the FPGA, with only the FPGA size to limit the scope of these algorithms.

## Reconfigurable I/O Architecture

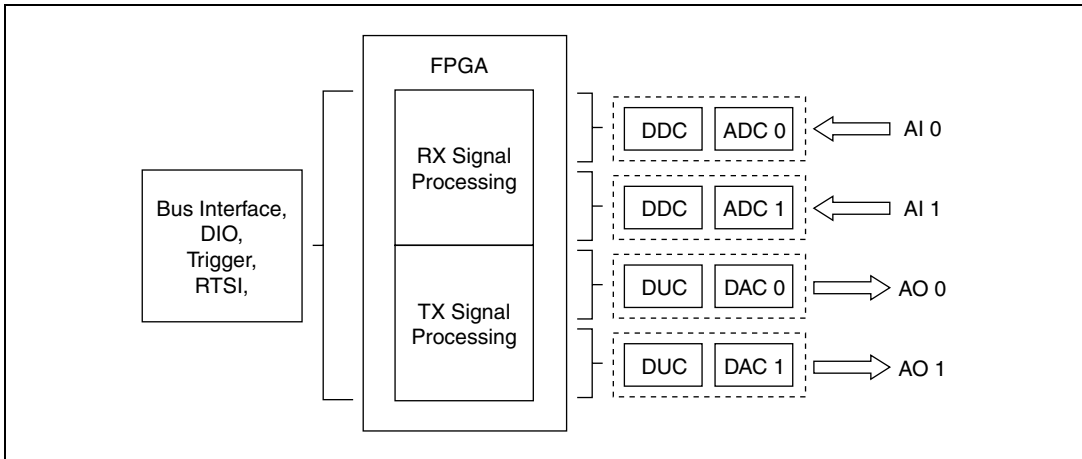
The following figure illustrates a generic representation of RIO device consisting of an FPGA connected to fixed I/O resources and a bus interface.



**Figure 1.** High-Level FPGA Functional Overview

The fixed I/O resources include analog-to-digital converters (ADCs), digital-to-analog converters (DACs), digital I/O (DIO) lines, and other input/output resources. Software accesses the RIO device through the bus interface, and the FPGA provides the connectivity between the bus interface and the fixed I/O, including any timing, triggering, processing, and custom I/O required by the application.

Triggering, processing, and custom I/O is provided by consuming logic in the FPGA. Each fixed I/O resource used by the application consumes a small portion of the FPGA logic, which performs basic control of the fixed I/O resource. The bus interface also consumes a small portion of the FPGA logic to provide software access to the device. The remaining FPGA logic is available for higher-level functions such as timing, triggering, and signal processing. Each of these functions consumes varying amounts of logic. For example, a typical 32-bit counter consumes 20 times more logic than a DIO resource, while an 8-bit counter consumes five times more logic than a DIO resource. The figure below illustrates using the FPGA for an IF transceiver application in a wireless communication system.



**Figure 2.** FPGA Logic for an IF Transceiver



**Note** On the NI PCI-5640R, the FPGA is volatile and does not retain its VI when powered off. The VI must be reloaded every time the device is powered on. The VI is downloaded to the FPGA from the host computer over the bus interface.

## Reconfigurable I/O Applications

To create new VIs for your application, you use the LabVIEW FPGA Module. The LabVIEW FPGA Module extends the LabVIEW development environment to allow you to design and use FPGA applications. Arbitrary functionality can be defined for the RIO device. If you are using the LabVIEW FPGA Module, refer to the included NI PCI-5640R FPGA examples.

## LabVIEW FPGA Module

The LabVIEW FPGA Module enables you to use LabVIEW to create VIs that run on the RIO device. The LabVIEW FPGA Module includes a function palette with VIs that can control the I/O, timing, and logic of the RIO device. The LabVIEW FPGA Module synthesizes your custom VI into a form that can be downloaded to the FPGA on the RIO device. The LabVIEW FPGA Module also includes a palette of functions for use in LabVIEW for Windows or when targeting a real-time (RT) target. These functions create applications that control the FPGA by programmatically reading and writing to the device.

## Xilinx Virtex II Pro FPGA

---

The Xilinx Virtex II Pro FPGA is used on the NI PCI-5640R to execute user-developed LabVIEW FPGA programs. This FPGA is connected to all the resources on the device (ADC, DAC, Clock Distribution Circuit [CDC], DIO, EXT TRIG) through point-to-point connections.

## Reconfigurable FPGA Resources

Table 1 lists the FPGA resources available for user configuration.

**Table 1.** User-Configurable FPGA Resources

FPGA Resource	Static	Reconfigurable
Combinatorial Logic	1%	99%
Memory Elements	2%	98%

## Component Datasheets

The information in the following component datasheets may be helpful for advanced users who are using the LabVIEW FPGA Module to create their NI PCI-5640R application. Refer to these documents for register maps, detailed information about functionality and programming, and for explanations of internal signals used in timing block diagrams.

- Datasheets for the AD 6654 ADC and AD 9857 DAC are available on the [www.analog.com](http://www.analog.com)<sup>TM</sup> Web site
- CDC 7005 Clock Distribution IC Datasheet available on the [www.ti.com](http://www.ti.com)<sup>TM</sup> Web site.

# Clocking Block Diagram

The NI PCI-5640R uses three categories of clocks: reference clocks, sample clocks, and IQ clocks. Figure 3 depicts the NI PCI-5640R clocking scheme.

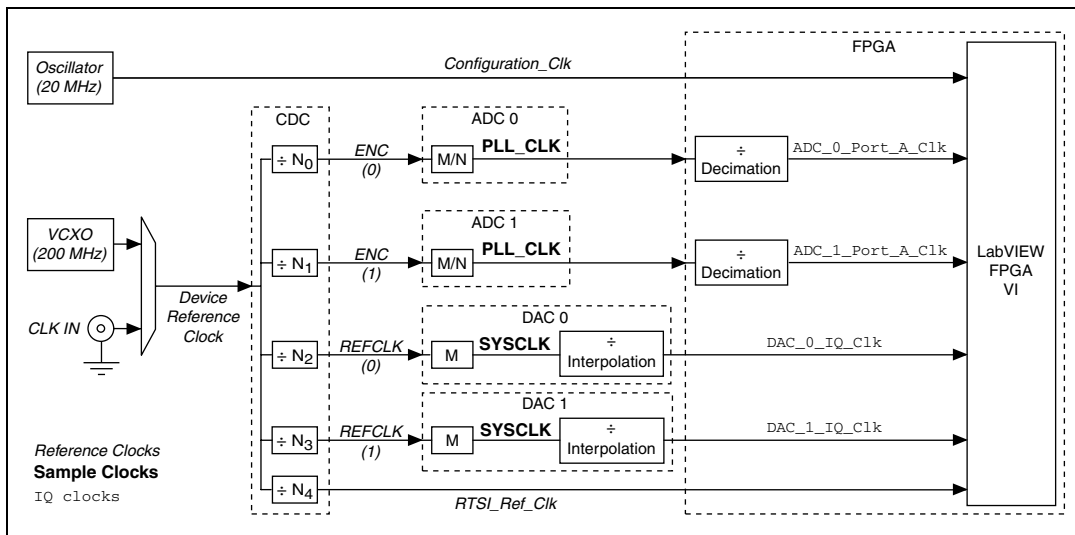


Figure 3. NI PCI-5640R Clocking Scheme

The 20 MHz Configuration\_Clk is not synchronized to the VCXO and runs independently of other clocks. The role of the 20 MHz Configuration\_Clk is to provide a fixed frequency configuration clock that is also used by the NI-STC2 ASIC for PCI DMA operations.

The Device Reference Clock can be configured as the internal 200 MHz VCXO or an external clock provided at the device front panel CLK IN connector. Specify the device reference clock source by calling the ni5640R Configure Timebase VI.

The valid values for CDC 7005 divisors (N0...N4) is 1, 2, 4, 8, and 16. *M*, *N*, interpolation values, and decimation values must be integers. Refer to the *Configuring IQ Clock Rates* section for more information.

# Configuring IQ Clock Rates

---

Observe the following guidelines when configuring IQ clock rates.

- $DAC_{<i>_i>}IQ\_Clk = 2 \times REFCLK_{DAC_{<i>_i>}} \times Clock\ multiplier_{DAC_{<i>_i>}} / Interpolation_{DAC_{<i>_i>}}$

Where

$REFCLK_{DAC_{<i>_i>}}$  is the specified *device reference clock* /  $N_{2\ or\ 3\ CDC}$ . Specify the divisor using the ni5640R Configure Timebase VI.

$Clock\ multiplier_{DAC_{<i>_i>}}(M)$  is equal to 1 or  $4 \leq M \leq 20$ . Configure the clock multiplier using the ni5640R DAC Program VI.

$Interpolation_{DAC_{<i>_i>}}$  is the hardware interpolation rate determined by the DAC fixed  $4\times$  interpolator times a programmable  $2\times$  to  $63\times$  CIC interpolating filter. The programmable CIC interpolator can be configured using the ni5640R DAC Profile VI.

- $ADC_{<i>_i>}Port\_A\_Clk = ENC_{ADC_{<i>_i>}} \times Clock\ multiplier_{ADC_{<i>_i>}} / (Predivide\ Factor_{ADC_{<i>_i>}} \times Decimation_{ADC_{<i>_i>}})$

Where

$ENC_{ADC_{<i>_i>}}$  is the device reference clock /  $N_{0\ or\ 1\ CDC}$ . Specify the divisor using the ni5640R Configure Timebase VI.

$Clock\ multiplier_{ADC_{<i>_i>}}(M)$  is equal to 1 or  $4 \leq M \leq 20$ . Configure the clock multiplier using the ni5640R Input Port VI.

$Predivide\ Factor_{ADC_{<i>_i>}}(N)$  is equal to 1, 2, 4, or 8. Configure the predivide factor using the ni5640R Input Port VI.

$Decimation_{ADC_{<i>_i>}}$  is the decimation factor for a particular channel in the ADC. Decimation is performed in various filters throughout the processing channel. Each channel includes one CIC filter (decimates by 1 to 32), two FIR-HB filters (each decimates by 2), one DRC filter (decimates by 1 to 16) and one CRCF filter (decimates by 1 to 16). Configure all these filters using the ni5640R ADC Configure DDC VI.

You can have different clock-division ratios on the ADC and DAC clocks using the CDC 7005 clock distribution chip. For example, in internal clocking mode, the ADCs are normally clocked at 100 MHz (VCXO clock rate/2), and the DACs are normally clocked at 200 MHz (VCXO clock rate/1). Refer to the *I/O Resources* section for more details.

Refer to the *Timebase Block Diagram* section in the *NI 5640R Help* for a depiction of the NI PCI-5640R clocking scheme.

# I/O Resources

---

*I/O resources* consist of both onboard ADCs, both DACs, the NI-STC2 ASIC, the RTSI bus, DIO lines, the clock distribution chip (CDC), trigger lines, and timing circuitry.

I/O resources are configured in one of the two following ways, as shown in Table 2.

- Using the host computer—these resources are mostly configuration resources that are not intended to be changed during the execution of the application.
- Using the FPGA VI—these resources can be made accessible to the host computer by wiring them to controls and indicators.

**Table 2.** Configuring I/O Resources

<b>Resource</b>	<b>Configuration Method</b>
CDC and other timing circuitry	Host <i>only</i>
RTSI configuration	Host <i>only</i>
Analog input	FPGA VI
Analog output	FPGA VI
DIO	FPGA VI
Trigger	FPGA VI
RTSI	FPGA VI



**Note** The NI-5640R example host VIs can be used to appropriately configure all I/O resources.

# I/O Resources Available using the FPGA VI

Figure 4 depicts the I/O resources available using the FPGA VI.

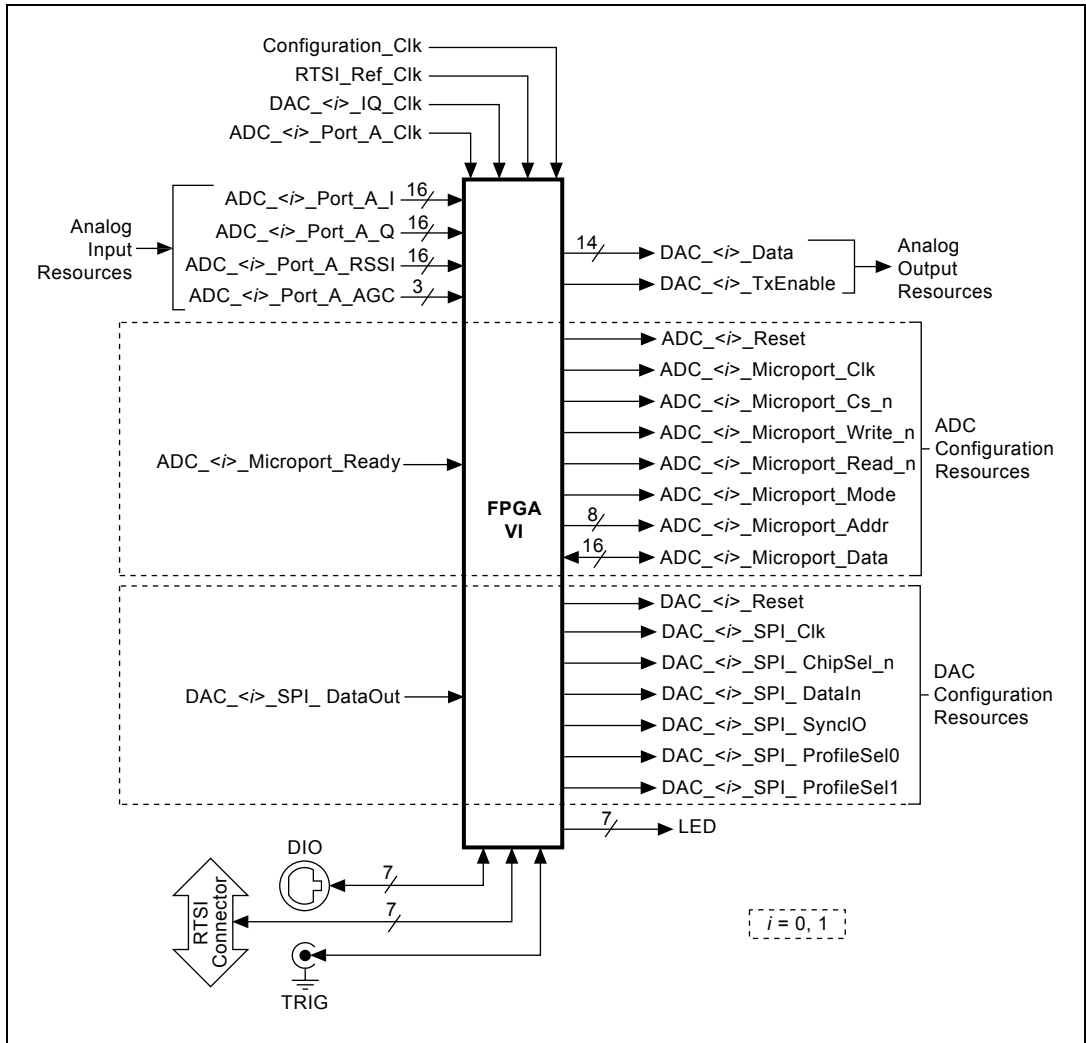


Figure 4. I/O Resources Available using FPGA VI

# Configuration I/O Resources

---

*Configuration I/O resources* consist of the configuration ports on ADCs and DACs. You must interface with configuration I/O resources to configure the ADCs and DACs from the FPGA VI.

To configure an ADC you must use its Microport Interface configured as an 8-bit port. To configure a DAC, use its Serial Interface configured as a 3-wire interface.



**Note** The NI-5640R example host VIs appropriately configure all I/O resources, so low-level direct access to ADC configuration resources and DAC configuration resources is not needed for most applications.

## Data I/O Resources

---

*Data I/O resources* consist of AI, AO, DIO lines, RTSI signals, and trigger signals as follows.

- AI—ADCs provide IQ data after downconversion. An IQ sample is supplied on the rising edge of every ADC IQ clock cycle
- AO—DACs require interleaved I and Q data on every DAC IQ clock cycle
- DIO lines—provide general-purpose I/O through seven lines configurable by pin as input or output. Refer to *Connecting Digital Input/Output (DIO) Signals* in the *NI 5640R IF Transceiver Getting Started Guide* for more information
- RTSI lines—can provide general-purpose digital I/O after appropriate host configuration
- Trigger lines—can be sent or received on DIO lines, RTSI lines, or the module front panel TRIG connector after appropriate host configuration

## Getting Started with FPGA VIs and Host VIs

---

Use the LabVIEW FPGA Module and NI-5640R examples to learn about building FPGA VIs and host VIs, communicating between VIs running on the FPGA target and on the host computer, and programming hardware functionality in the FPGA VI.

In this section you will learn about FPGA application components.

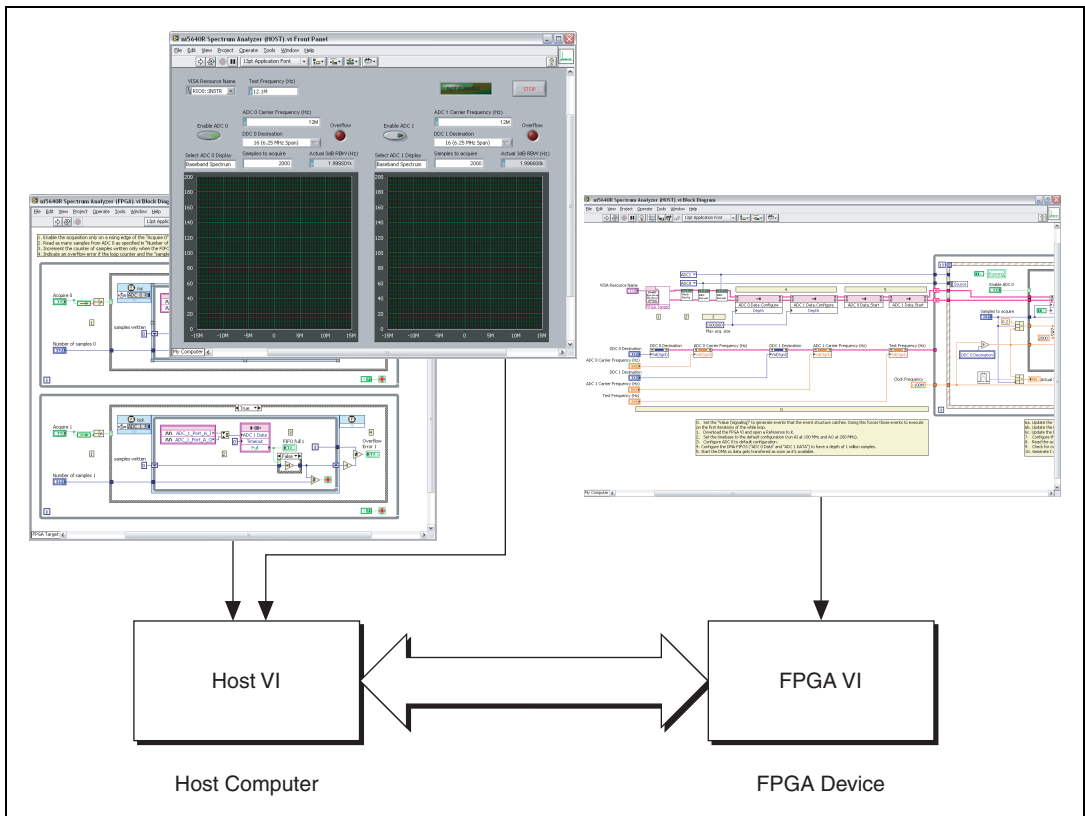
- An *FPGA target* is the NI hardware that includes a reconfigurable field-programmable gate array (FPGA) and on which you can run FPGA VIs you develop in LabVIEW with the LabVIEW FPGA

Module. In this case the FPGA target is the NI PCI-5640R IF transceiver.

- An *FPGA VI* is an NI-5640R VI that you download to an FPGA target that determines the functionality of the hardware using the FPGA I/O Nodes and VIs.
- A *host VI* is an NI-5640R VI that runs in software on the host computer and uses the FPGA Interface VIs to control and monitor the FPGA VI on the FPGA target.



**Note** ni5640R example host VIs are *not* the same as the NI-5640R instrument driver. Instrument driver VIs do *not* work if you are developing an application with the LabVIEW FPGA Module.



**Figure 5.** Programmatic FPGA Interface Communication

In this section, you will learn to download and run an FPGA VI and to communicate with an NI-5640R FPGA VI from a NI-5640R host VI.

## NI-5640R FPGA Examples

You can use NI-5640R FPGA examples to learn various concepts and programming techniques in LabVIEW. The NI-5640R examples demonstrate a variety of tasks and concepts:

- Using the NI PCI-5640R with the LabVIEW FPGA Module
- Performing basic I/O
- Performing basic processing in the FPGA
- Performing analog input
- Performing analog output
- Communicating between the FPGA VI running on the NI PCI-5640R and the host VI running on a host computer
- Using new and advanced features, including timed loops, multiple clock domains, and direct memory access (DMA).



**Note** ni5640R example host VIs are *not* the same as the NI-5640R instrument driver. Instrument driver VIs do *not* work if you are developing an application with the LabVIEW FPGA Module.

You can find more information about these topics in the *LabVIEW Help*, available within LabVIEW by selecting **Help»Search the LabVIEW Help**. Refer to the *NI 5640R Help*, available from the **Start** menu at **All Programs»National Instruments»NI-5640R»Documentation**, for information specific to the NI PCI-5640R.

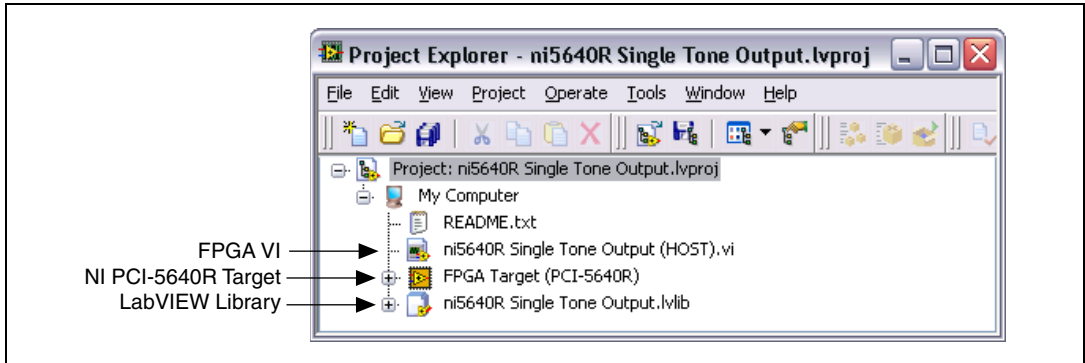
## The LabVIEW Project

A *LabVIEW project* is a file that allows you to manage VIs and targets on the development computer. You must configure the FPGA target to appear in the **Project Explorer** window. Complete the following steps to open a project in the **Project Explorer** window and add an FPGA target to the project.

1. Launch LabVIEW.
2. Access the NI-5640R FPGA examples directory by navigating to the **Start** menu and selecting **All Programs»National Instruments»NI-5640R»Examples**.
3. Open the `Fpga` folder to view the LabVIEW FPGA examples.
4. Open the `ni5640R Single Tone Output` folder.

5. Double-click the `ni5640R Single Tone Output.lvproj` to open the project in LabVIEW.

Notice that the NI PCI-5640R target appears in the **Project Explorer** window. Also notice that some I/O resources and the FPGA VI appear in the **Project Explorer** window below the NI PCI-5640R target, as shown in Figure 6.



**Figure 6.** The Project Explorer Window

You now can open and explore an FPGA VI example in the **Project Explorer** window.

## Exploring an FPGA VI

The FPGA VI is the VI that runs on the FPGA target. Complete the following steps to view an example FPGA VI for the NI PCI-5640R:

1. Open the `ni5640R Single Tone Output (FPGA) VI` from the **Project Explorer** window. The target on which the VI runs (`My Computer` or `FPGA Target`), appears in the lower left corner of the VI as shown in Figure 7.

- Open the block diagram and locate the Configuration Timed Loop. As shown in Figure 7, there are two subVIs inside this timed loop: the ni5640R Template Config ADC (FPGA) VI and the ni5640R Template Config DAC (FPGA) VI.

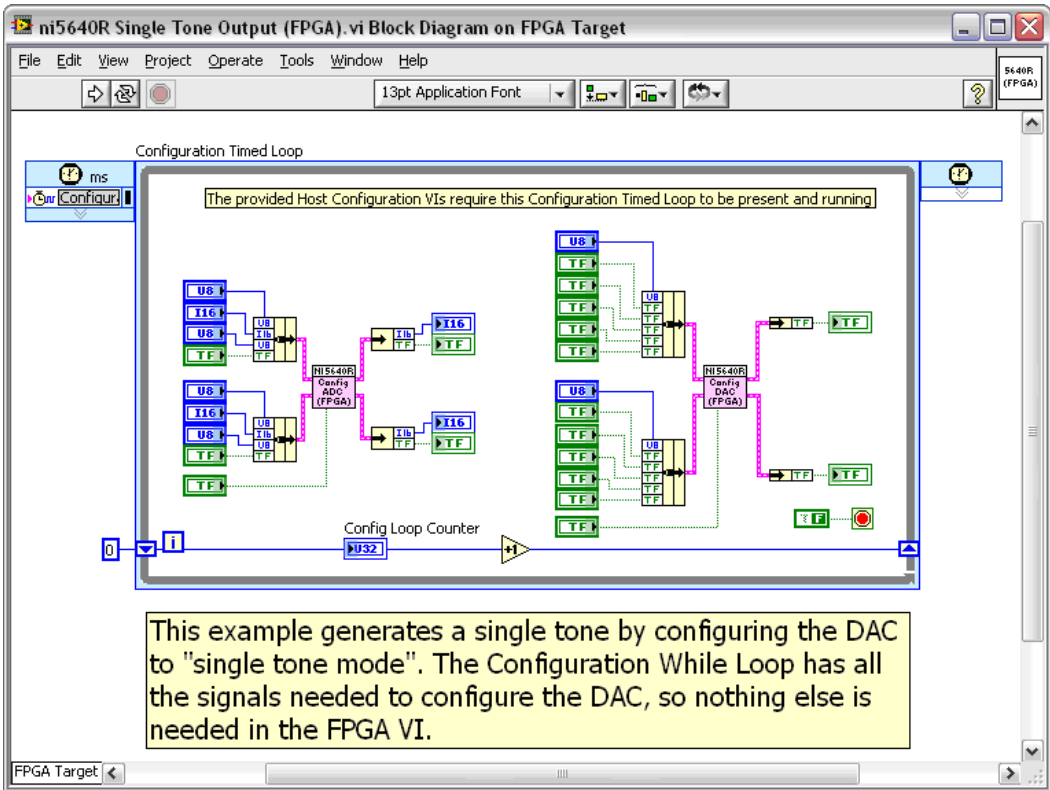
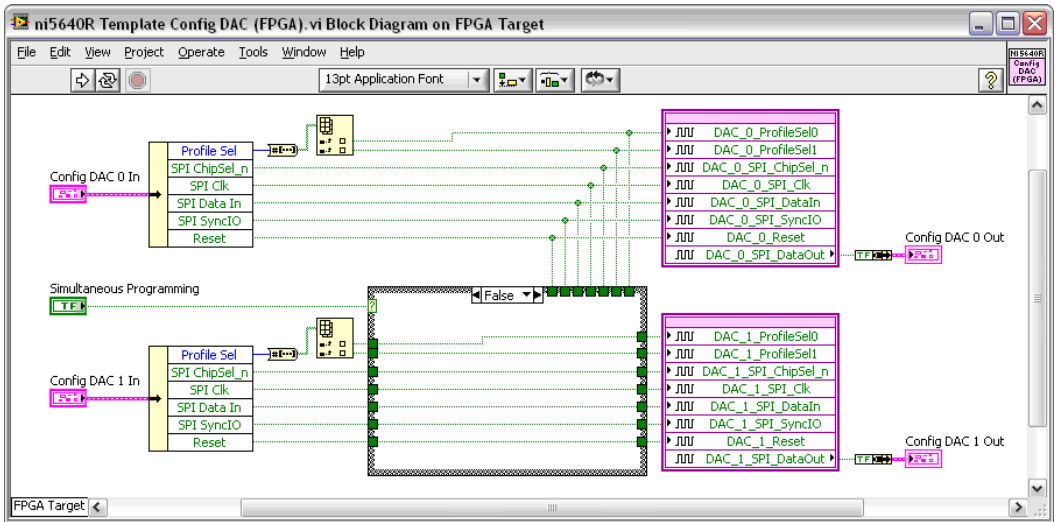


Figure 7. ni5640R Single Tone Output (FPGA) VI Block Diagram

- Open these subVIs and open their block diagrams. Notice the I/O Nodes—these are required to configure the ADCs and DACs. You can read from and write to registers in the ADCs and DACs using these nodes. The I/O Nodes are wired to controls and indicators that are written to or read from the calling VI. The ni5640R Template Config DAC (FPGA) VI block diagram is shown in Figure 8.



**Figure 8.** ni5640R Template Config DAC (FPGA) VI Block Diagram

- Close the subVIs. Notice there are controls and indicators wired to the subVIs. These controls and indicators are accessed by the provided ni5640R example host VIs in order to configure the ADCs and DACs. You must specify settings such as the decimation/interpolation factor, filters, and modes of operation to configure the ADCs and DACs before the VI can write any to or read any data from them. You will not need to write data directly to these controls and indicators because the ni5640R example host VIs provide a high-level interface to them.



**Note** ni5640R example host VIs are *not* the same as the NI-5640R instrument driver. Instrument driver VIs do *not* work if you are developing an application with the LabVIEW FPGA Module.

Notice that the Configuration Timed Loop is clocked by the Configuration\_Clk (shown in the upper left corner of the loop in Figure 7). The Configuration\_Clk is an onboard 20 MHz free-running oscillator that is not synchronized to the VCXO and runs independently of other clocks. Refer to the *Clocking Block Diagram* section for more information about the Configuration\_Clk.

The Configuration Timed Loop is always running (because there is a **FALSE** constant wired to the **Loop Condition** terminal of the loop). This continuous operation allows the ni5640R host VIs to configure the ADCs and DACs at any point during runtime.

No specific code generates a sine tone because the DACs can internally generate tones at the specified frequency. To configure this feature, use the provided ni5640R example VIs to specify the frequency and configure the DAC for single-tone mode.

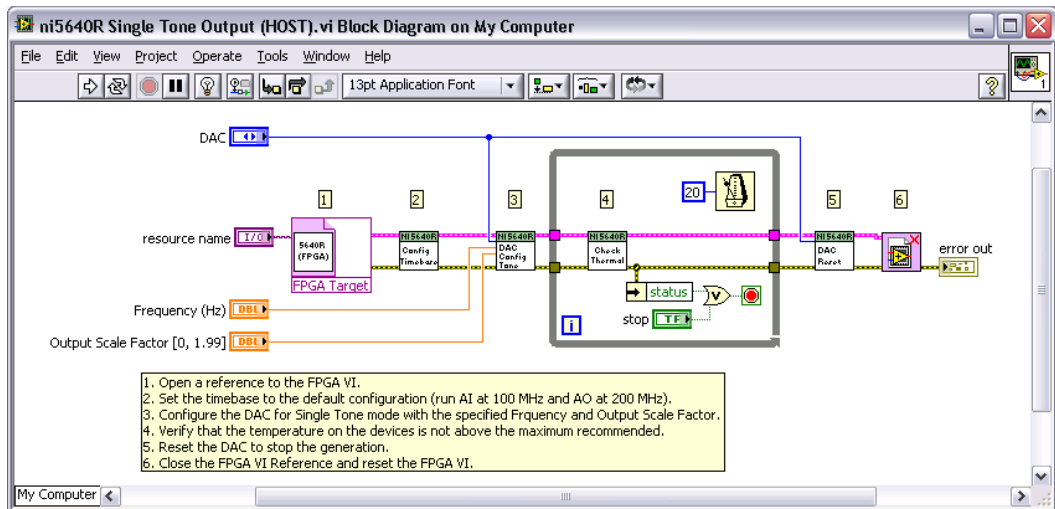


**Note** The Configuration Timed Loop is required by the ni5640R host VIs to be present and running.

## Exploring a Host VI

The host VI runs in software on the host computer and communicates with the FPGA VI on the NI PCI-5640R. You can place the host VI in the same project as the FPGA target and the FPGA VI. Complete the following steps to open a host VI:

1. Open the ni5640 Single Tone Output (HOST) VI from the **Project Explorer** window. Notice that the host VI appears above the FPGA target in the **Project Explorer** window; the host VI appears under My Computer because it runs on the Windows computer. As shown in Figure 9, the target on which the VI runs (in this case, My Computer), appears in the lower left corner of the VI.



**Figure 9.** ni5640R Single Tone Output (HOST) VI Block Diagram

2. The following four VIs appear on the block diagram:



- Open FPGA VI Reference—This VI downloads an FPGA VI to the FPGA (if it has not been previously downloaded) and opens a reference to it.
- pni5640R Configure Timebase—This VI configures the device timebase. If all inputs are unconnected, this VI configures the hardware to use the internal VCXO (200 MHz), a divide factor of 2 for the ADCs to set their reference clock to 100 MHz, and a divide factor of 1 for the DACs to set their reference clock to 200 MHz.
- ni5640R DAC Configure for Single-Tone Mode—This VI configures the specified DAC to operate in single-tone mode, configures the DAC frequency, and configures the DAC's output scale factor.
- ni5640R DAQ Reset—This VI stops the generation.
- Close FPGA VI Reference—Stops and resets the FPGA VI running on the FPGA target and closes the reference to the VI.

3. Run the Single Tone Output (HOST) VI and notice that it generates a sine tone at the specified frequency.

In the host VI, the ni5640R DAC Configure for Single-Tone Mode VI configures the DAC to generate a sine tone at a particular frequency with a specific power level. The ni5640R DAC Configure for Single-Tone Mode VI writes the controls within the FPGA VI Configuration Timed Loop to access the DAC registers. The sole purpose of the FPGA VI in this example is to provide an interface that the NI 5640R VIs use to access the DAC registers.

## Creating an FPGA Application

---

Now that you have explored a project, an FPGA VI, and a host VI, you will develop an application that includes all three components.

### Overview of the Application

To develop and use an FPGA VI with the NI PCI-5640R, you must perform the following steps.

1. Create the LabVIEW project
2. Add the NI PCI-5640R hardware target

3. Develop and compile the FPGA VI
4. Create the host VI to communicate with the target

In this case, you will develop a simple spectrum analyzer using one of the analog input channels on the NI PCI-5640R. You will be able to configure the center frequency and the bandwidth to analyze. The application also generates a sine tone that can be used as the signal to analyze.

## Using the ni5640R Template

The ni5640R Template contains all the elements needed to develop an application using the NI PCI-5640R. It contains a LabVIEW project that has been preconfigured to target the NI PCI-5640R. It also contains a Template FPGA VI and a Template Host VI. Complete the following steps to use the ni5640R Template:

1. From Windows Explorer, copy the folder named ni5640R Template (installed at <LabVIEW>\Examples\instr\ni5640R\Fpga) to a new location.
2. Rename the folder My Simple Spectrum Analyzer.
3. Open the My Simple Spectrum Analyzer folder and rename the ni5640R Template.lvproj file to My Simple Spectrum Analyzer.lvproj.
4. Double-click My Simple Spectrum Analyzer.lvproj to open the project in LabVIEW.

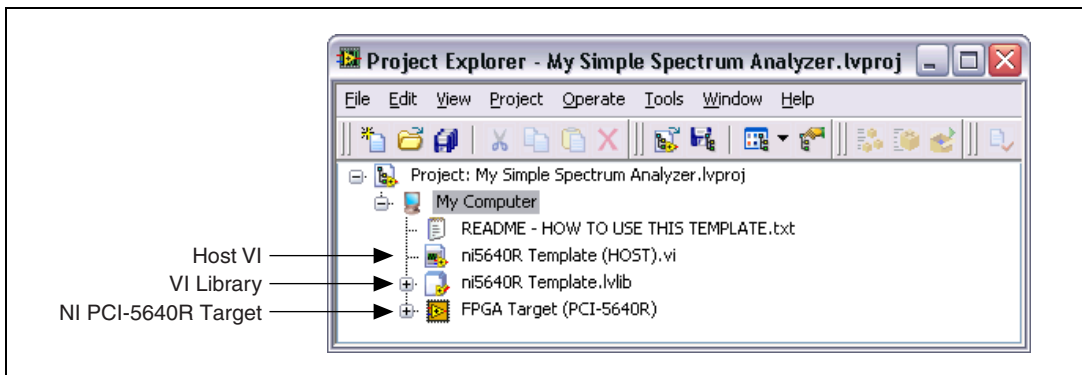


Figure 10. My Simple Spectrum Analyzer Project

5. Double-click ni5640R Template (HOST) .VI to open the VI.



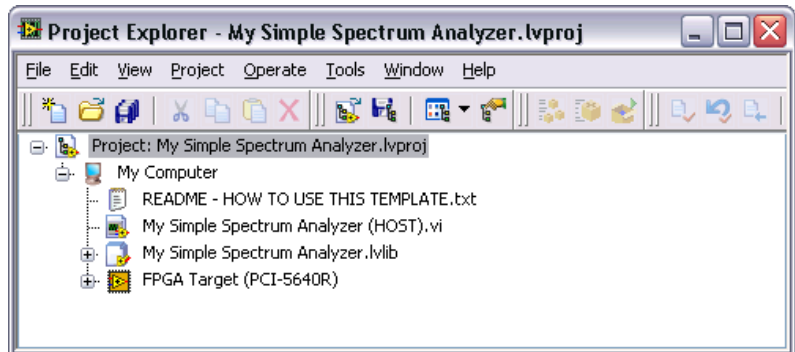
**Caution** In later steps, you will update some components of this VI. It is important to keep this VI open during updates to automatically propagate changes to affected components. If the VI is closed while updating, it cannot propagate changes properly and might be unable to find updates upon reopening.

6. Return to the **Project Explorer** window. Right-click `ni5640R Template.lvlib` and select **Rename** from the shortcut menu to launch the Save As dialog box.
7. Select **Rename** and then **Continue** from the dialog box.
8. Rename the library to `My Simple Spectrum Analyzer.lvlib` and click **OK**.



**Note** The `.lvlib` file contains all the high-level VIs required to configure the NI PCI-5640R.

9. Right-click `My Simple Spectrum Analyzer.lvlib`, and select **Save All** from the shortcut menu.
10. Right-click the `ni5640R Template (HOST) VI`, and select **Save As**.
11. Save the `ni5640R Template (HOST) VI` with a new name: `My Simple Spectrum Analyzer (HOST).vi`.
12. At the LabVIEW **File** dialog box, select **Save All**. Your LabVIEW project should now look like Figure 11.



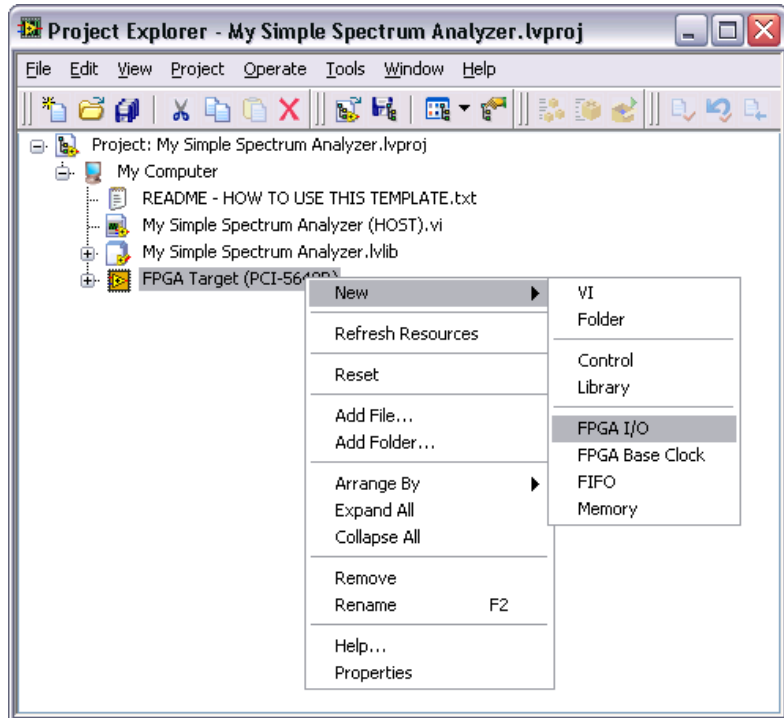
**Figure 11.** My Simple Spectrum Analyzer Project

## Configuring the NI PCI-5640R Target

All resources used by the FPGA VI must first be added to the project. In this section, you will add FPGA I/O, a DMA FIFO, and an FPGA base clock.

### Adding FPGA I/O for Analog Input

1. From the **Project Explorer** window, right-click **FPGA Target (PCI-5640R)** and select **New»FPGA I/O** from the shortcut menu, as shown in Figure 12. The **New FPGA I/O** dialog box appears, displaying the I/O resources available on the NI PCI-5640R.



**Figure 12.** Adding New FPGA I/O Resources

2. Under the Analog Input folder, select **ADC\_0\_Port\_A\_I** and **ADC\_0\_Port\_A\_Q**.
3. Click **Add** to add these resources to the **New FPGA I/O** table.
4. Click **OK**. The FPGA I/O items you configured appear in the **Project Explorer** window in a folder below the NI PCI-5640R FPGA target.

## Adding DMA FIFO

5. Right-click the NI PCI-5640R FPGA target in the **Project Explorer** window and select **New»FIFO**. You will use this FIFO to transfer the acquired data to the host computer.
6. On the **FPGA FIFO Properties** dialog box, select **Target to Host – DMA** as the **Type** and set **Depth** to 32767 elements.
7. Click **OK**.

## Adding a Base Clock for Analog Input

8. Right-click the NI PCI-5640R FPGA target in the **Project Explorer** window and select **New»FPGA Base Clock**.
9. From the **FPGA Base Clock Properties** dialog box, select the **Resource** named **ADC\_0\_Port\_A\_Clk**. The **ADC\_0\_Port\_A\_Clk** ticks every time the ADC has a new IQ sample.
10. Click **OK**.

## Developing the FPGA VI

FPGA VIs run on FPGA targets and define the functionality and features of the targets on which they run. In this case, you will use an included template VI to build an FPGA VI that acquires data using one of the analog input channels and then sends packets of data to the host VI using DMA.

1. From the **Project Explorer** window, right-click the ni5460R Template (FPGA) VI, and select **Save As** from the shortcut menu.
2. Select **Rename** and then **Continue** from the dialog box.
3. Rename the VI to `My Simple Spectrum Analyzer (FPGA).VI`.
4. Open the block diagram for `My Simple Spectrum Analyzer (FPGA).VI`. Create a new Timed loop beneath the Configuration Timed Loop. The Timed loop is located on the **Functions** palette under **Programming»Structures»Timed Structures**.
5. Wire a **FALSE** constant to the **Loop Condition** terminal, so that the new loop is always running.

- Select both Analog Input FPGA I/O items in the **Project Explorer** window and drag them onto the block diagram inside the timed loop you created. Two FPGA I/O Nodes appear on the block diagram configured with the specific FPGA I/O item. Your block diagram should now look like Figure 13.

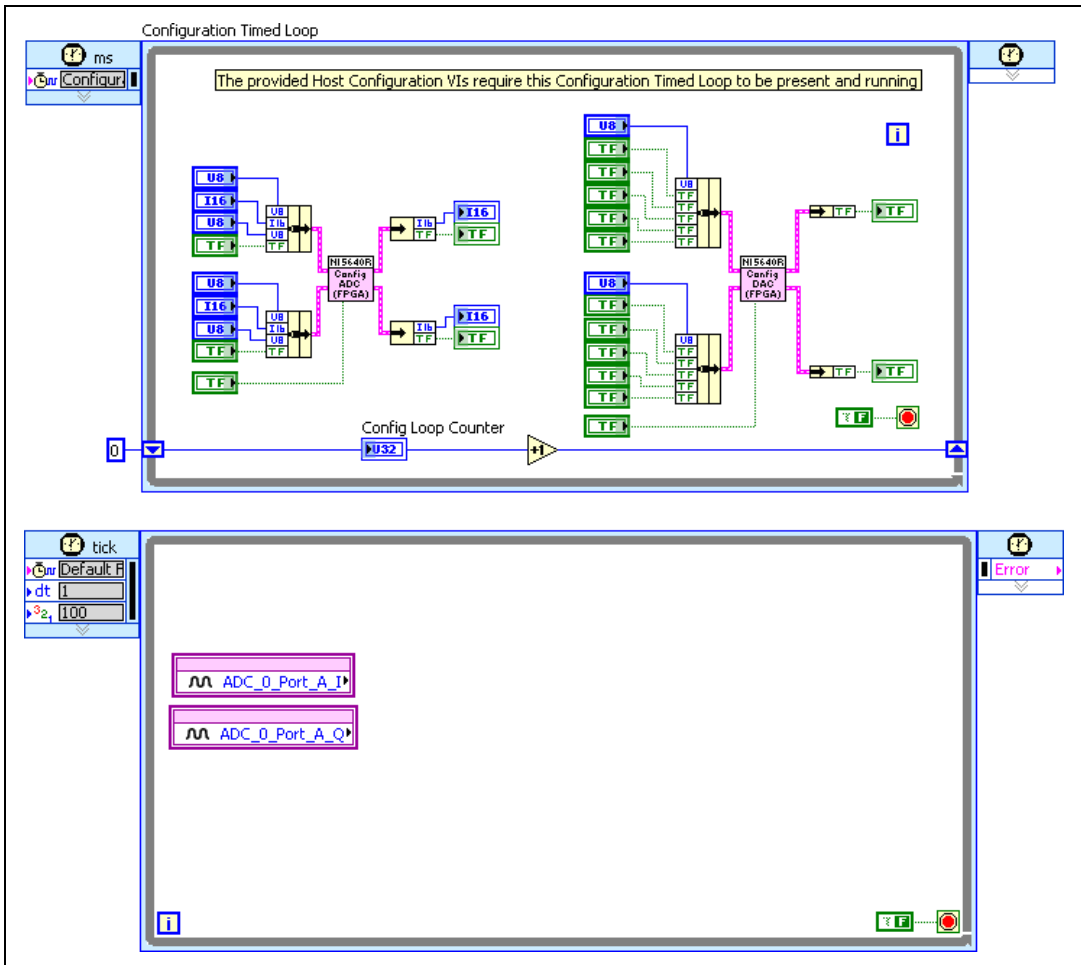
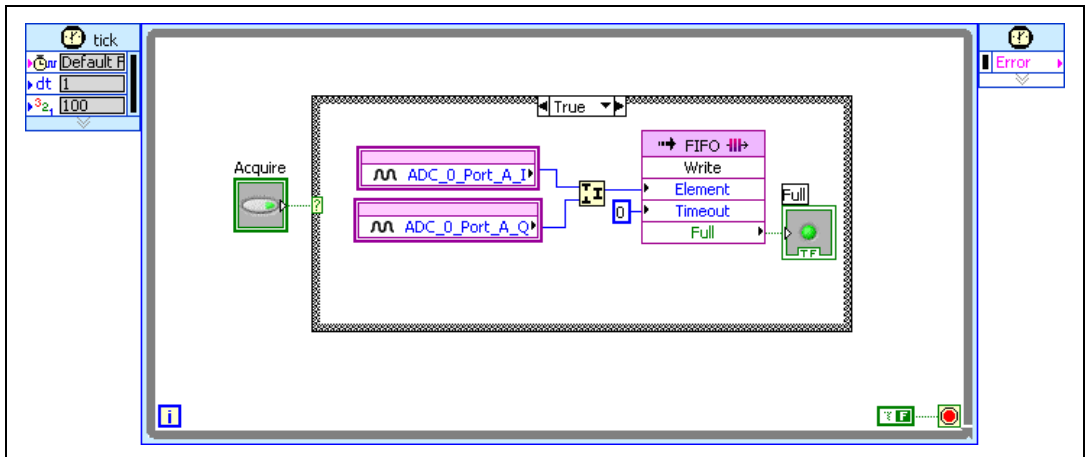


Figure 13. Adding I/O Nodes to the FPGA VI

7. Select the FIFO item in the **Project Explorer** window and drag it onto the block diagram inside the timed loop, to the right of the FPGA I/O Nodes.
8. Place a Join Numbers VI between the FPGA I/O Nodes and the FIFO. The Join Numbers VI is located on the **Functions** palette under **Programming»Numeric»Data Manipulation**.
9. Wire the outputs of the FPGA I/O Nodes to the input of the Join Numbers VI. Then wire the output of the Join Numbers VI to the input of the FIFO.
10. Create a numeric constant of zero and wire it to the **Timeout** input of the FIFO.
11. Create a Boolean indicator and wire it to the **Full** output of the FIFO.
12. Create a case structure around the code as shown in Figure 14. This image does not show the Configuration Timed Loop.
13. Create a Boolean control called **Acquire** and wire it to the case structure.



**Figure 14.** Timed Loop Setup

The acquired I and Q data, which are 16-bit values, are packed into a 32-bit value by the Join Numbers VI. The packed value is then sent to the FIFO, which transfers the data to the host computer using DMA. The only remaining task is specifying how often to acquire I and Q samples.

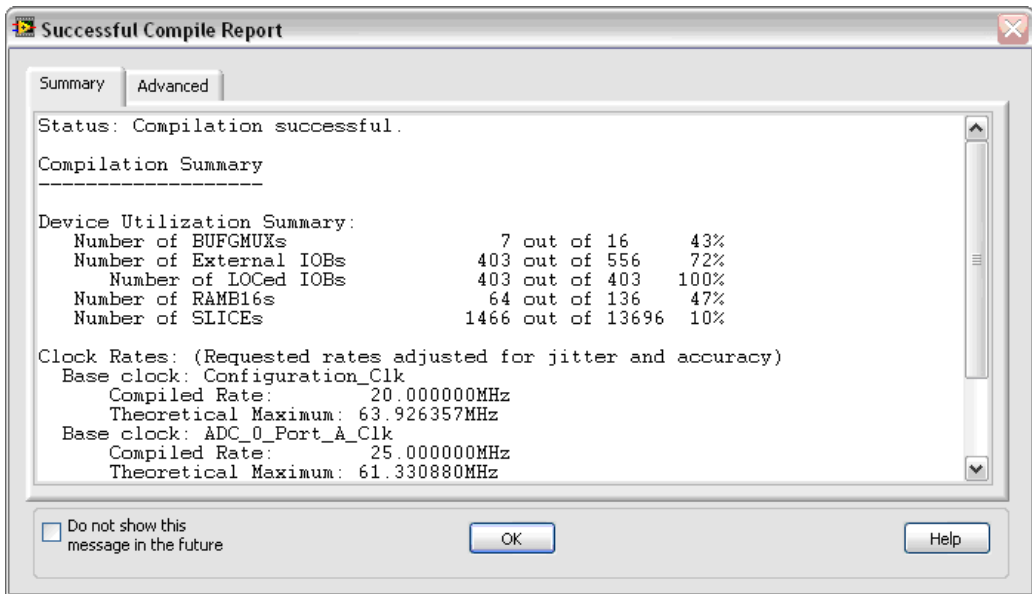
14. On the FPGA VI block diagram, double-click the input node (top left corner of the loop) on the Timed loop to view the **Configure Timed Loop** dialog box. This dialog box allows you to configure which clock runs the Timed Loop. Notice that only those clocks that have been added to the project appear in the list of Available Timing Sources.

15. In the **Configure Timed Loop** dialog, click **Select Timing Source** and select **ADC\_0\_Port\_A\_Clk**. Your FPGA VI is now complete.
16. Save the VI and the project.

## Compiling the FPGA VI

You now can compile the FPGA VI. In the **Project Explorer** window, right-click the FPGA VI and select **Compile** from the shortcut menu.

When the compilation process is complete, the LabVIEW FPGA Compile Server displays a report indicating that the compilation was successful. Click the **OK** button on the dialog box shown in Figure 15.



**Figure 15.** FPGA Download Complete Dialog



**Note** Compilation might take up to 30 minutes, depending on your system and VI complexity.

## Developing the Host VI

To communicate programmatically with the FPGA VI, you must develop a host VI that runs on a Windows PC. Complete the following steps to create a host VI that communicates with the FPGA VI running on the NI PCI-5640R hardware module.

### Opening a Reference to the FPGA VI

1. Open the block diagram for the My Simple Spectrum Analyzer (HOST) VI. Right-click the Open FPGA VI Reference VI and choose **Select VI** to launch the **Select VI** dialog box.
2. Select the My Simple Spectrum Analyzer (FPGA) VI you created in the *Developing the FPGA VI* section.
3. Click **OK**.

## Adding NI-5640R VIs

- In the **Project Explorer** window, expand the My Simple Spectrum Analyzer library and double-click the ni5640R VI Tree.VI to open it. Navigate to the ni5640R VI Tree VI block diagram.

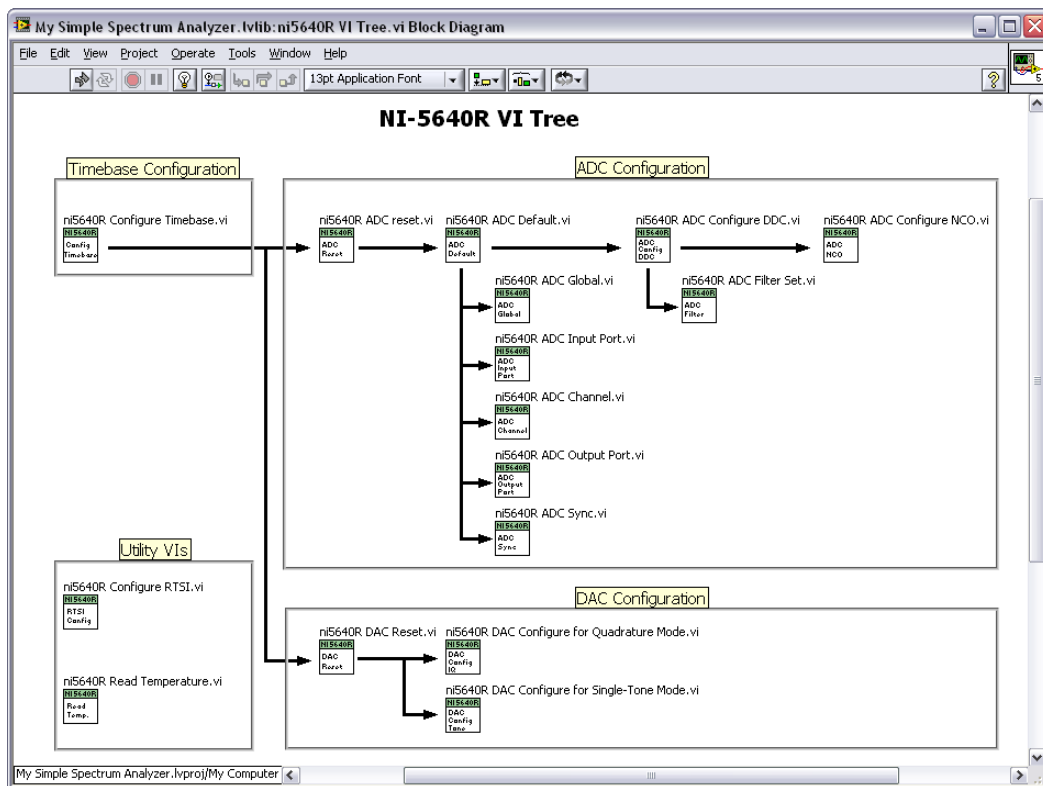


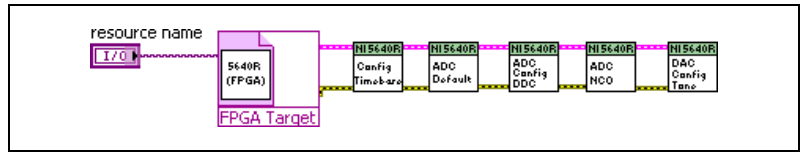
Figure 16. ni5640R VI Tree Block Diagram

- Copy the following VIs from the ni5640R VI Tree VI block diagram to the block diagram of My Simple Spectrum Analyzer (HOST) VI.
  - ni5640R ADC Default.VI—configures analog input
  - ni5640R ADC Configure DDC.VI—configures analog input
  - ni5640R ADC Configure NCO.VI—configures analog input
  - ni5640R DAC Configure for Single-Tone Mode.VI—generates sine tone stimulus

6. Connect the preceding VIs as shown in Figure 17.



**Note** Delete any existing VIs or data structures, such as loops, that do not appear in Figure 17 from the block diagram.



**Figure 17.** Host VI Block Diagram Connections

## Acquiring Data From the FPGA VI

7. Place an Invoke Method Node, available on the **Functions»FPGA Interface** palette, on the block diagram.
8. Wire the Invoke Method Node to the ni5640R DAC Config for Single Tone VI.
9. Left-click the Invoke Method Node and select **Method»FIFO»Configure**. This method configures the size of the software FIFO. It limits the maximum number of elements that your VI can read.
10. Place another Invoke Method Node, available on the **Functions»FPGA Interface** palette, on the block diagram.
11. Left-click the Invoke Method Node and select **Method»FIFO»Start**. This method starts DMA transfer from the DMA FIFO that you created in the FPGA VI to a software DMA FIFO.
12. Place a Read/Write Node, available from located on the **Functions** palette under **FPGA Interface** palette, on the block diagram.
13. Left-click the Read/Write Node and select **Acquire**.
14. Place another Invoke Method Node on the block diagram.
15. Left-click the Invoke Method Node and select **Method»FIFO»Read**. This method reads the contents of the software DMA FIFO.
16. Create a control named **Number of Elements** and wire it to the **Depth** parameter in the **FIFO»Configure** method and to the **Number of Elements** parameter in the **FIFO»Read** method, as shown in Figure 18.



26. Ensure that your block diagram is wired as shown in Figure 19.

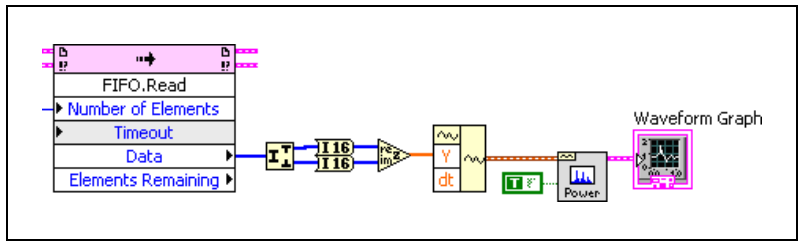


Figure 19. Analyzing the Data

## Adding Controls and Indicators

27. Add the following controls and indicators, as shown in Figure 20:

- **Decimation**—specifies how much decimation is performed on the analog input signal. This value determines the maximum bandwidth of your signal.
- **NCO frequency**—specifies the center frequency for the analog input waveform.
- **Frequency**—specifies the frequency at which to generate a sine tone. This tone can serve as a test signal for the simple spectrum analyzer.

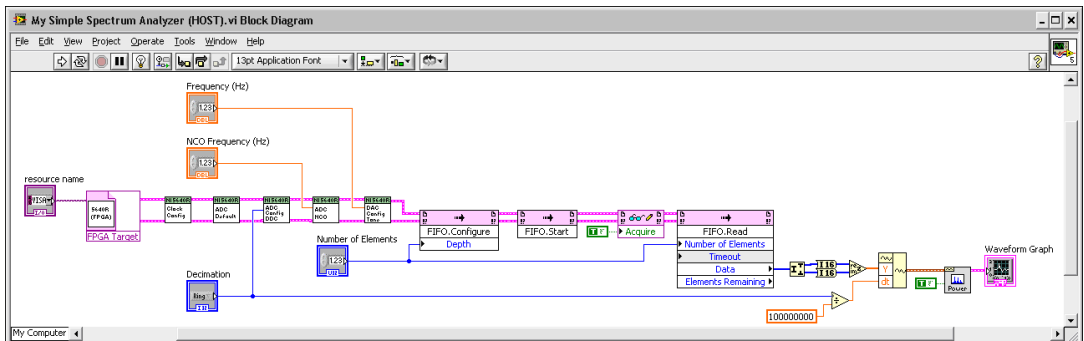


Figure 20. Host VI Block Diagram With Controls and Indicators

28. Divide the **decimation rate** value by 100M and wire the output to the **dt** input of the Build Waveform VI.

## Closing the Reference to the FPGA VI

29. Close the reference to the FPGA VI running on the FPGA target by placing the Close FPGA VI Reference VI, available on the **Functions** palette under **FPGA Interface**, on the block diagram as shown in Figure 21.
30. Place a Simple Error Handler VI, found on the **Programming»Dialog & User Interface** palette, on your block diagram.
31. Save the VI. Your host VI block diagram and front panel should now look like Figures 21 and 22, respectively.

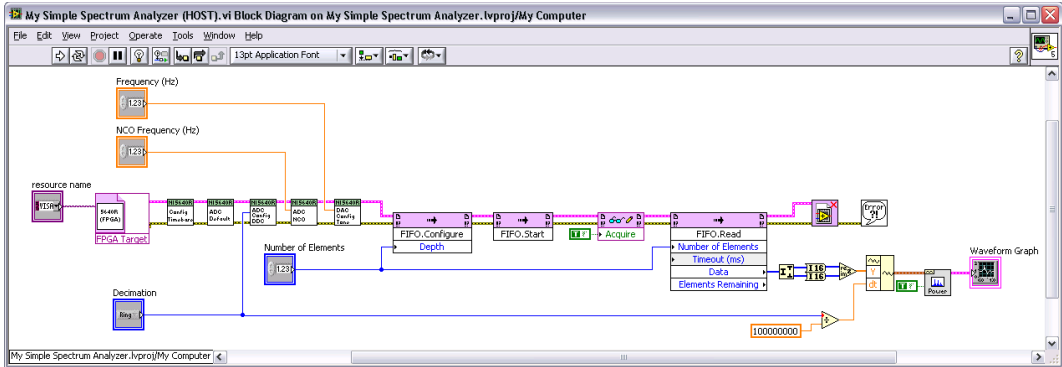
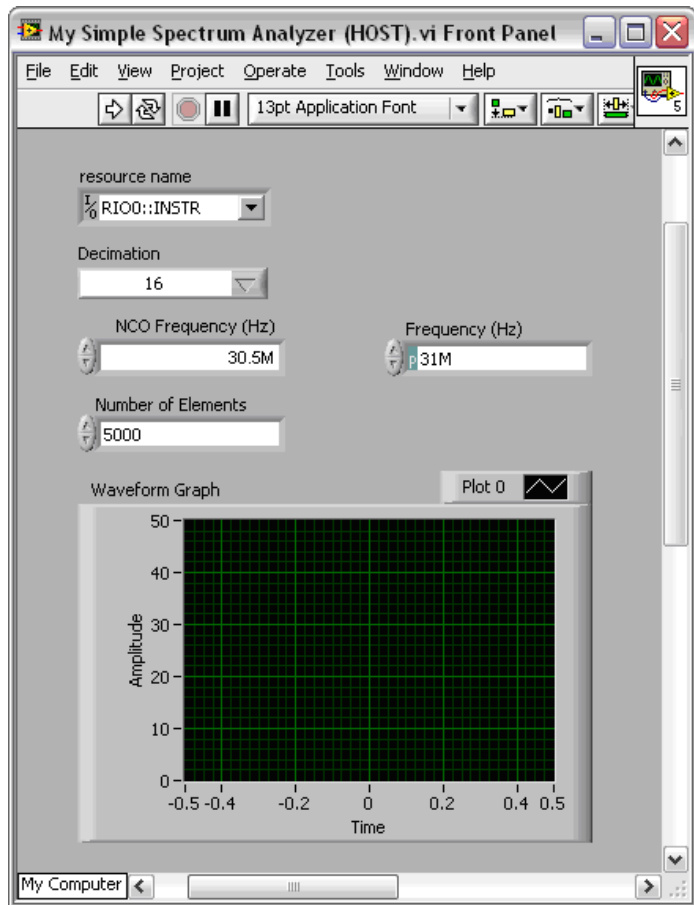


Figure 21. Host VI Block Diagram



**Figure 22.** Host VI Front Panel

32. Populate your application with the control values shown in Figure 22.
33. On the NI PCI-5640R front panel, connect the AO 0 connector to the AI 0 connector.
34. Click the **Run** button to run the host VI. The host VI programmatically downloads and runs the FPGA VI on the NI PCI-5640R.

The spectrum of the generated tone appears on the host VI front panel waveform graph. The tone appears at 500 kHz because the generated tone frequency and the analyzer center frequency are 500 kHz apart.

# What You Have Learned

---

This document demonstrates the following key points about developing a NI PCI-5640R solution in LabVIEW:

- A typical NI PCI-5640R solution consists of a LabVIEW project, at least one FPGA VI, and at least one host VI.
- The FPGA VI runs on the FPGA on the NI PCI-5640R. The host VI can run on a Windows PC.
  - Use the LabVIEW project to organize VIs, configure settings for the VIs, configure the NI PCI-5640R, and configure the FPGA I/O.
  - Use the FPGA VI to read and write to the NI PCI-5640R I/O Nodes and to implement logical operations in the FPGA.
  - Use a DMA FIFO to transfer data to the host computer.
  - Use the host VI to communicate with the FPGA VI.
  - Use Timed Loops for time-critical operations.

## Using FPGAs for Signal Processing

---

This section provides more information about using FPGAs for signal processing applications.

### Using Single-Cycle Timed Loops

Single-Cycle Timed Loops are While loops in which every cycle is guaranteed to execute in a single clock cycle. The clock that drives the Single-Cycle Timed Loop is configurable. This type of structure is critical for analog input (AI) and analog output (AO) operations. After a DAC has been configured, it expects a new sample on every rising edge of its IQ clock (`DAC_<i>_IQ_Clk`). Similarly, an ADC provides a new sample on every rising edge of its IQ clock (`ADC_<i>_Port_A_Clk`). Therefore, you must always write data to a DAC inside a Single-Cycle Timed Loop that is clocked by the DAC IQ clock. Similarly, you must always read data from the ADC inside a Single-Cycle Timed Loop that is clocked by the ADC IQ clock.

Some LabVIEW FPGA primitives cannot be placed inside a Single-Cycle Timed Loop without eliciting a code generation error. Refer to the *Executing Code in a Single FPGA Target Clock Cycle* section of the *LabVIEW Help* for more information about the Single-Cycle Timed Loop and the VIs you can use in the Single-Cycle Timed Loop.

## Understanding Clock Domains

A clock domain refers to all the components that use a particular clock to run. All the code inside a Single-Cycle Timed Loop is timed with the selected clock, making that loop a clock domain. The LabVIEW FPGA Module automatically adds the necessary infrastructure to transfer data safely between clock domains. Nevertheless, there is some latency when transferring data from one clock domain to another.



**Note** Different signals that cross clock domains are not guaranteed to arrive at the same time. Bundle your signals in a cluster to ensure that they arrive simultaneously.

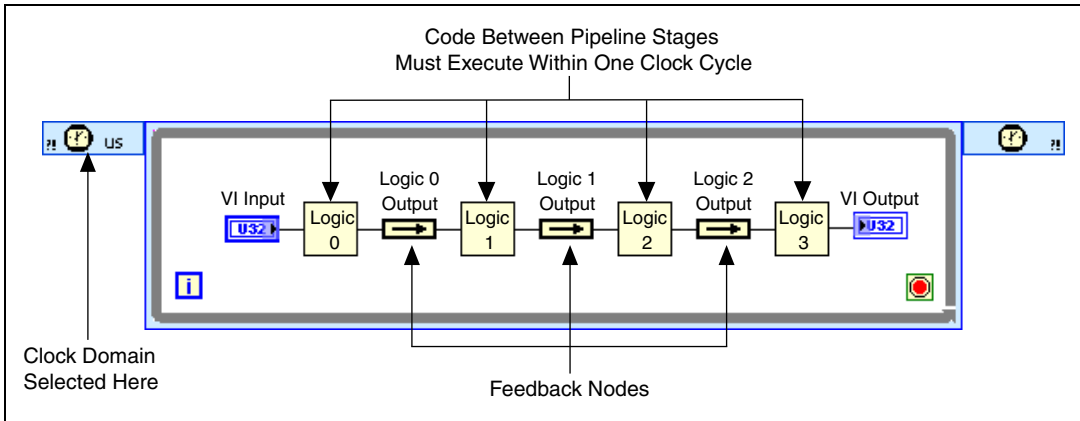
## Pipelining

Pipelining is an implementation technique that allows using Single-Cycle Timed Loops to accomplish operations requiring multiple clock cycles.

Most applications require some signal processing before writing to the AO or after reading from the AI. If the signal processing is inside a Single-Cycle Timed Loop, a single clock cycle may not provide enough time for the FPGA to perform all necessary computations.

To resolve this problem you can pipeline your signal processing. Pipelining divides the Single-Cycle Timed Loop into segments separated by registers, which can be implemented using shift registers, feedback nodes, or local variables. Each segment executes a portion of the computation that can be completed within a single clock cycle. At each clock cycle, every segment executes and writes its results to a feedback node. On the next iteration of the loop, the stored result is fed to the next segment. The multiple segments overlap in execution, so each segment processes new data at every clock cycle.

Refer to Figure 23 for an example of pipelining. Additionally, the LabVIEW FPGA examples illustrate this technique.



**Figure 23.** Four-Stage Pipeline

Each output value is processed using four substeps. The Logic 0, Logic 1, Logic 2, and Logic 3 blocks run concurrently.



**Note** Refer to the *Executing Code in a Single FPGA Target Clock Cycle* topic of the *LabVIEW Help* for more information about the Single-Cycle Timed Loop and the VIs you can use in the Single-Cycle Timed Loop.

You can also refer to the *Using the Timed Loop to Write Multirate Applications in LabVIEW* application note on the NI Developer Zone Web site. This application note describes the features of the Timed loop and how to use the Timed loop to develop multirate applications.

## FPGA Signal Processing References

The following texts are recommended for in-depth information on FPGA signal processing.

For detailed discussions of signal processing theory, discrete-time artifacts, and the Nyquist Theorem:

- Oppenheim, Alan V., and Schafer, Ronald W., and Buck, John R. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1999.

For high-level introductory material presented in intuitive ways with real-world examples:

- Smith, Steven W. *Digital Signal Processing: A Practical Guide for Engineers and Scientists*. Oxford: Newnes, 2002.

For very detailed, analytical discussions of various communication signal processing operations:

- Gitlin, Richard D., and Hayes, Jeremiah F., and Weinstein, Stephen B. Data Communications Principles. London: Plenum Publishers, 1992.
- Proakis, John G., and Salehi, Masoud. Communications Systems Engineering. Englewood Cliffs, NJ: Prentice Hall, 1994.

## Tips and Troubleshooting

---

This section reviews some tips and troubleshooting options that may be helpful when you use the NI PCI-5640R module and NI-5640R software.

### It takes a long time to access controls and indicators in the Read/Write Control

When you click on the Read/Write Control, LabVIEW must locate the corresponding controls and indicators in your FPGA VI. If your FPGA VI is not loaded into memory when you click the Read/Write Control, then LabVIEW loads it at this point and unloads it when the operation is complete. Avoid LabVIEW loading and unloading the FPGA VI by keeping the FPGA VI open while you work on your Host VI.

### I get a pop-up indicating that an error was detected in the communication between the host computer and the FPGA target

Error -61046 may occur when your application tries to read or write controls or indicators that are in a slow clock domain. For example, you may see this error if you have a control/indicator inside a single-cycle timed loop that is clocked by the ADC\_0\_Port\_A\_Clk and have configured the ADC 0 to decimate by a large value (larger than 128). Such a setup can cause this problem because the control or indicator is being updated very slowly and the communication times out while trying to access it. This configuration is shown in Figure 24.

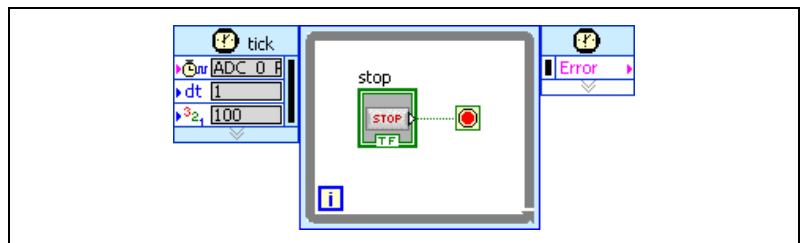


Figure 24. Controls in a Slow Clock Domain

To solve this problem, place the control or indicator in a clock domain that is faster (like the default Top-Level Clock). You can then use local variables to transfer the value to a slower clock domain, as shown in Figure 25.

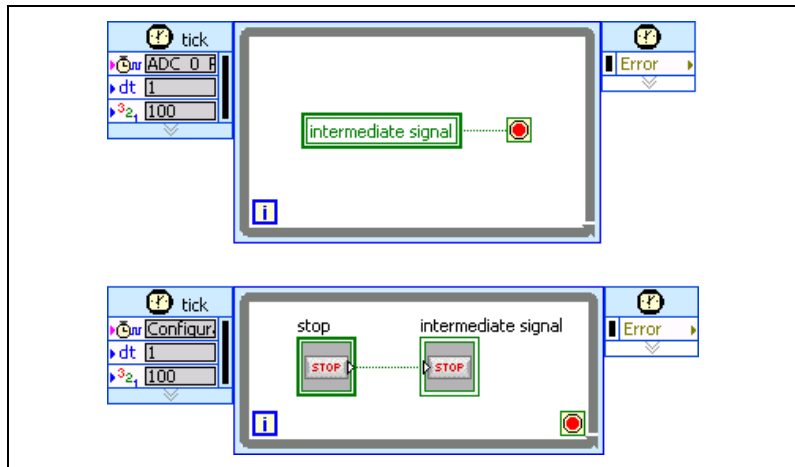


Figure 25. Using Local Variables to Transfer the Value

## When I open a Host VI, LabVIEW cannot find a file named filename.lvbit

In this situation, click **Cancel** and the VI should load properly. This situation can occur if some of your subVIs have been saved with the typedef control pointing to another FPGA VI. In this case, when your subVI loads, it attempts to locate the original FPGA VI. However, when the VI is finished loading, if the “bind to typedef” option is selected on the “Open FPGA Reference” VI, the typedef is updated to point to the right FPGA VI.

## When I Run an FPGA VI, a compilation error instantly occurs

The following suggestions may help to address this problem:

- The problem could be related to timing constraints in the FPGA. Check the compilation report and make sure your FPGA VI is not violating any timing constraints.
- The problem could be that the compilation process aborted abnormally at some point without being detected by LabVIEW. In this case, LabVIEW caches the error report and, because LabVIEW detects that your FPGA VI has not changed, it simply returns the cached result for every compilation request. You can force a compilation by right-clicking on the FPGA VI on the Project Explorer window and selecting **Compile**.

# Where to Go From Here

---

Refer to the *FPGA Module* book in the *LabVIEW Help* for more information about the LabVIEW FPGA Module, including conceptual topics about the features available with the LabVIEW FPGA Module, procedures to complete common tasks with the LabVIEW FPGA Module, and reference information about VIs and functions available with the LabVIEW FPGA Module.

Refer to the *FPGA Interface* book in the *LabVIEW Help* for more information about programming host VIs to communicate with FPGA VIs.

Refer to the *NI 5640R Help*, available from the **Start** menu at **All Programs»National Instruments»NI-5640R»Documentation**, for more information about the NI PCI-5640R and NI-5640R programming.



**Note** These help files are available within LabVIEW from the **Help** pull-down menu.

National Instruments, NI, ni.com, and LabVIEW are trademarks of National Instruments Corporation. Refer to the *Terms of Use* section on [ni.com/legal](http://ni.com/legal) for more information about National Instruments trademarks. Virtex-II Pro is a trademark of Xilinx, Inc. Other product and company names mentioned herein are trademarks or trade names of their respective companies. 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](http://ni.com/patents).