# LabVIEW™ FPGA Course Manual

**Course Software Version 2009**
**August 2009 Edition**
**Part Number 372510C-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 1800 300 800, Austria 43 662 457990-0, Belgium 32 (0) 2 757 0020, Brazil 55 11 3262 3599, Canada 800 433 3488, China 86 21 5050 9800, Czech Republic 420 224 235 774, Denmark 45 45 76 26 00, Finland 358 (0) 9 725 72511, France 01 57 66 24 24, Germany 49 89 7413130, India 91 80 41190000, Israel 972 3 6393737, Italy 39 02 41309277, Japan 0120-527196, Korea 82 02 3451 3400, Lebanon 961 (0) 1 33 28 28, Malaysia 1800 887710, Mexico 01 800 010 0793, Netherlands 31 (0) 348 433 466, New Zealand 0800 553 322, Norway 47 (0) 66 90 76 60, Poland 48 22 328 90 10, Portugal 351 210 311 210, Russia 7 495 783 6851, Singapore 1800 226 5886, Slovenia 386 3 425 42 00, South Africa 27 0 11 805 8197, Spain 34 91 640 0085, Sweden 46 (0) 8 587 895 00, Switzerland 41 56 2005151, Taiwan 886 02 2377 2222, Thailand 662 278 6777, Turkey 90 212 279 3031, United Kingdom 44 (0) 1635 523545

For further support information, refer to the *Additional Information and Resources* appendix. To comment on National Instruments documentation, refer to the National Instruments Web site at ni.com/info and enter the info code feedback.

# Contents

## Lesson 5
## Timing an FPGA VI



## Lesson 6
## Data Sharing on FPGA



## Lesson 7
## Single-Cycle Timed Loops



## Lesson 8
## Basic Host Integration – PC/Real-Time



## Lesson 9
## DMA Data Transfers

## Lesson 10
## Modular Programming and Code Reuse



## Appendix A
## Pipelining



## Appendix B
## Additional Information and Resources

## Course Evaluation

# 4

# FPGA I/O

In this lesson, you will learn how to add FPGA I/O to your LabVIEW project and use it on the block diagram. You will also learn about the differences between performing I/O on an R Series device and on a CompactRIO chassis and the differences between integer and fixed-point data. Using I/O Nodes, you will learn how to access both analog and digital data.

## Topics

# A. Introduction

Inputs and outputs on FPGA targets allow you to connect the FPGA target to other devices, such as sensors and actuators. FPGA I/O resources are fixed elements of the FPGA targets that you use to transfer data among the different parts of the system. On some FPGA targets, FPGA I/O resources correspond to lines on front panel connectors, PXI backplanes, or Real-Time System Integration (RTSI) connectors. On other FPGA targets, FPGA I/O resources are nodes inside FPGAs that connect the part of the FPGA designed by National Instruments with the part of the FPGA you design. Each FPGA I/O resource has a specific type, such as digital or analog. An FPGA target might have multiple resources of the same or different types. You can create FPGA I/O items, determine the I/O resources on the FPGA target that you want to use, and then assign unique names to the I/O resources you use.

**Note** Refer to the specific FPGA target hardware documentation for information about supported features and I/O functionality on the FPGA target you use.

Several I/O example VIs are available in the NI Example Finder in **Toolkits and Modules»FPGA»CompactRIO»Basic IO** and **Toolkits and Modules»FPGA»CompactRIO»FPGA Fundamentals**.

Use the following terms when working with FPGA I/O Nodes:

- **Terminal**—Represents a hardware connection on a CompactRIO module.
- **I/O Resource**—Specifies a logical representation in LabVIEW FPGA of a hardware terminal.
- **I/O Name**—Specifies a name assigned by the developer to a particular I/O resource that normally describes the function of the resource. Assigning a different I/O resource to an I/O name updates all instances of the I/O name within the project.

The FPGA VI configures the FPGA circuit. When the FPGA circuit is activated, it performs the I/O operations in hardware. For example, if you configure an FPGA I/O node to read a digital line, the FPGA I/O node reads the line and returns the result. Consequently the FPGA can react to the input with the speed and determinism available in the FPGA target hardware.

You can put inputs and outputs, analog and digital, all in the same node on the block diagram. You can use target-specific properties and methods on the FPGA I/O items with the FPGA I/O Property Node and the FPGA I/O Method Node, respectively.

# B. Configuring FPGA I/O

In order to access I/O on the FPGA, you must first add I/O to your project. Depending on the FPGA target you are working with, there are different ways to add FPGA I/O to a LabVIEW project. If you are using a CompactRIO target, you can choose to detect modules when your add your chassis to the project. Autodetection of modules ensures that all FPGA I/O resources are added to the project. When you add the chassis to your project, the Discover C Series Modules dialog box shown in Figure 4-1 appears.

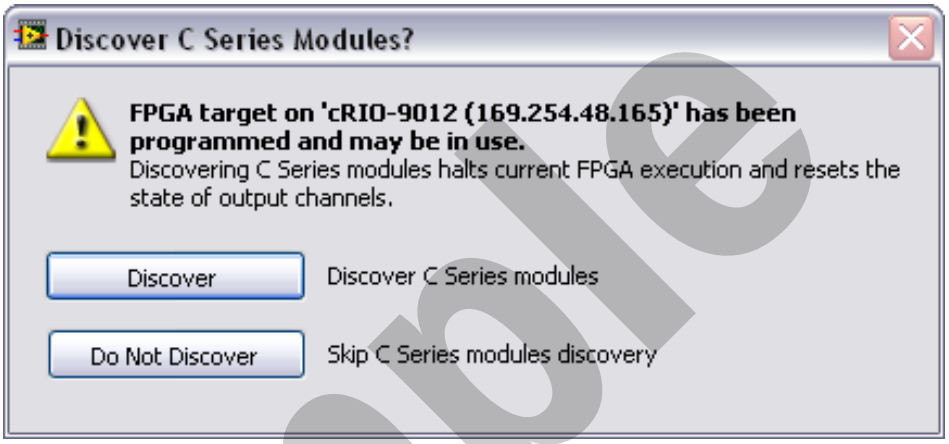


**Figure 4-1.** Discover C Series Modules Dialog Box

Click **Discover** to automatically detect the modules and add I/O resources to the project. If you choose not to automatically detect modules, add them manually by right-clicking the FPGA target and selecting **New»C Series Modules**.

When using an R Series target, you can manually add I/O resources to your project to ensure that you add only the resources that you intend to use. This significantly reduces the number of items in the Project Explorer window. Right-click the FPGA target in the Project Explorer and select **New»FPGA I/O** to launch the New FPGA I/O dialog box, show in Figure 4-2. Select **I/O** in the left pane of the New FPGA I/O dialog box and click **Add** to add the I/O resource to the project. Rename I/O resources by clicking the default name and entering a meaningful name for your application.

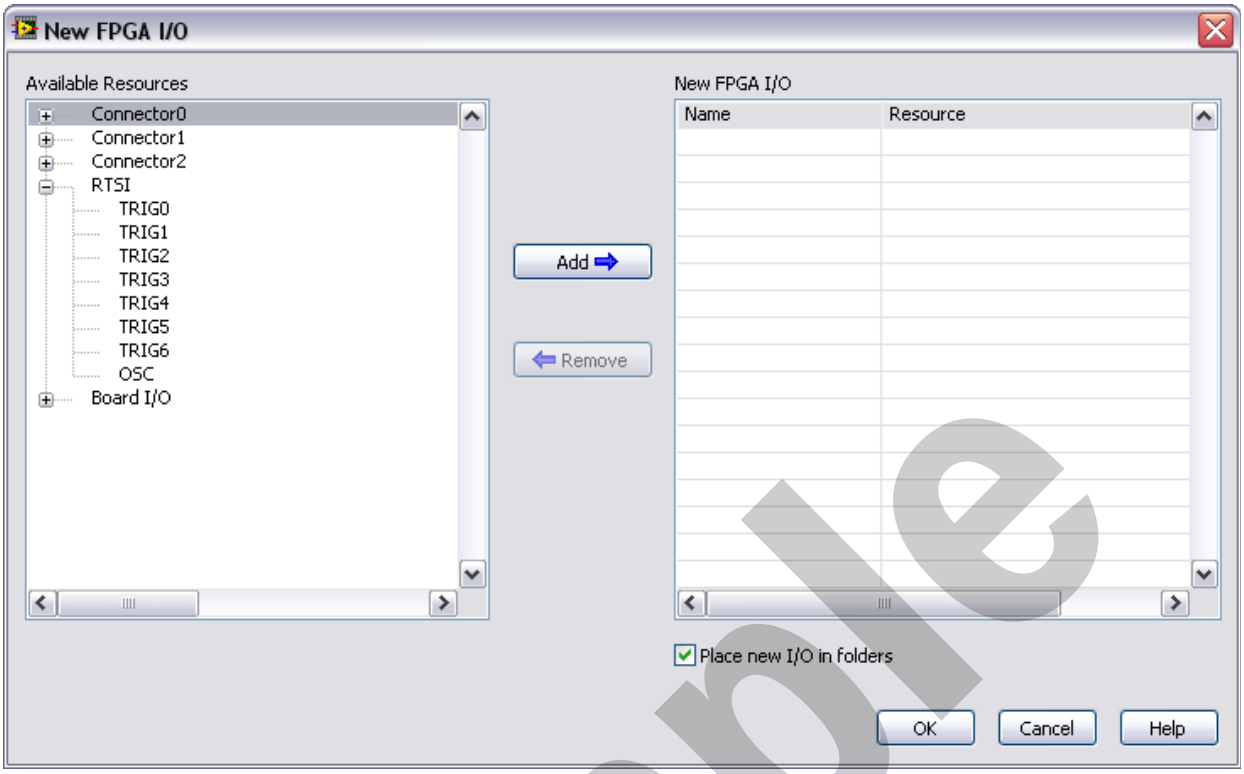


**Figure 4-2.** Discover C Series Modules Dialog Box

After you add I/O resources to your project, you can access them in VIs developed under the FPGA target. You can access FPGA I/O resources by opening the FPGA I/O palette. Table 4-1 describes the objects available in that palette.

**Table 4-1.** FPGA I/O Palette Objects

| Palette Object | Description |
|---|---|
| FPGA I/O Constant | Specifies an FPGA I/O item on the block diagram. |
| FPGA I/O Method | Invokes a method on an I/O item or hardware under and FPGA target in the Project Explorer window. |
| FPGA I/O Node | Performs specific FPGA I/O operations on FPGA targets. |
| FPGA I/O Property | Gets or sets one or more properties on an I/O item or hardware under an FPGA target in the Project Explorer window. |

You place an FPGA I/O Node on the block diagram from the palette or by dragging an FPGA I/O item from the Project Explorer.

If you add the FPGA I/O Node from the Functions palette, you must configure it by right-clicking, selecting **Select FPGA I/O**, and making the appropriate choices for configuration. The Select FPGA I/O submenu

displays the FPGA I/O items that appear in the Project Explorer tree. You can also click the FPGA I/O Node and use the shortcut menu to add new FPGA I/O items or select from FPGA I/O items you previously added to the project.

# C. I/O Types

There are two basic FPGA I/O types in LabVIEW FPGA hardware: digital I/O and analog I/O.

In addition to these basic two FPGA I/O types, there are also CompactRIO modules for motion control and CAN. Refer to the *LabVIEW Help* for more information on those modules.

## Digital

Digital lines are basic digital I/O. Digital lines are bi-directional on all R Series devices and some CompactRIO modules. Refer to the documentation for your specific device for more information about digital enable and digital data functionality.

FPGA targets might organize digital I/O resources as individual lines or as groups of lines called ports. A digital line I/O uses the Boolean data type. Ports use a data type that is dependent on the target. Ports for most I/O are a group of 8 bits, but others can be 16 or 32 bits. One bit is used for each line. Some FPGA targets provide access to digital I/O resources as only lines or ports. Other FPGA targets allow you to access the same physical lines as individual lines and as ports.

You can use digital input and output resources to configure the I/O resource and control the direction of dataflow. If you use a digital resource to write an output signal, you must disable the output before you can use the same resource to read an input signal. Use the FPGA I/O Method Node with the Set Output Enable method to disable the output line.

If you use the FPGA I/O Node to write a digital output, the FPGA I/O Node writes the data and enables the terminal for output. You also can use the FPGA I/O Method Node with the Set Output Data method to write data without enabling the output. Use the FPGA I/O Method Node with the Set Output Enable method to enable the digital terminal, which allows the data to be driven out. Use the Set Output Data method before the Set Output Enable method to specify the state of the digital resource when you enable the output. For example, you might have one portion of the block diagram continuously generating an internal signal. Use the FPGA I/O Method Node with the Set Output Enable method in another portion of the block diagram to independently control when the internal signal is actually driven out to an external device.

The FPGA devices do not have built-in counter hardware. All counters must be programmed into the FPGA itself. The count register can be 32, 16, or 8 bits, depending on the type of integer selected for the counter indicator. The loop period also determines the minimum detectable pulse width. An example counter is shown in Figure 4-3.

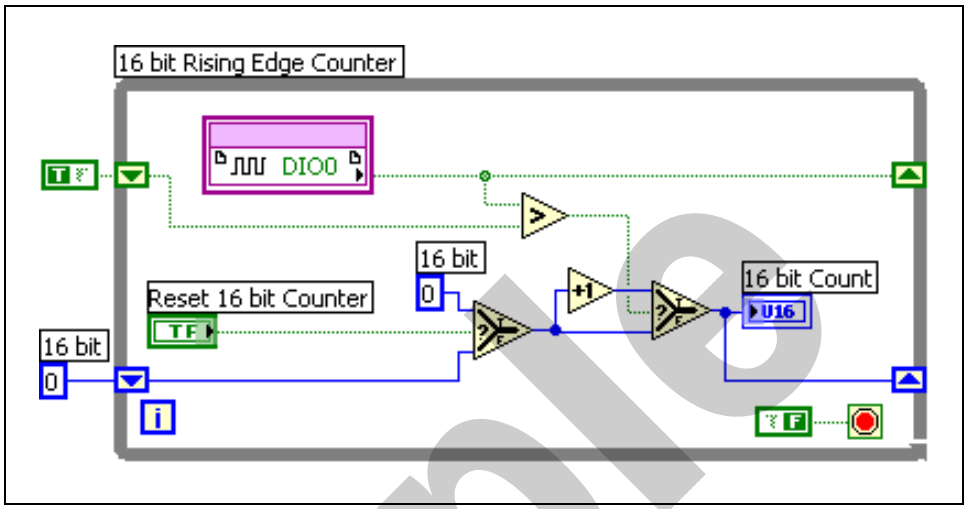


**Figure 4-3.** 16 bit Rising Edge Counter

The example shown in Figure 4-3 has the following specifications:

- About 10 ticks per iteration
- 250 ns to guarantee a high or low read
- 500 ns period = 2 MHz signals.

It can read a signal at a maximum of 2 MHz. It is important to benchmark your counter before using it in a final application. Refer to **Toolkits and Modules»FPGA»Compact RIO/R Series»FPGA Fundamentals» Counters** in the NI Example Finder for related examples.

## Analog

There are two analog I/O data types available for use, depending on your FPGA target. If you use an R Series target, the analog I/O data type is either an I16 or an I32, depending on the device. These devices return uncalibrated data. If you use a CompactRIO target, then the default data type is fixed-point and the data has been calibrated. You can, however, choose to configure the module to return raw integer data as well.

No matter the type of target, floating-point data cannot be used. This limits the types of operations you can perform on the FPGA. If you must perform floating-point analysis, you should pass the data to a host VI. More information about passing data between the FPGA and the host can be found in Lesson 8, *Basic Host Integration – PC/Real-Time*.

# D. Integer Math

If you configure an FPGA I/O Node to read an analog input, the FPGA I/O Node might initiate a conversion, wait for the result, and then return the binary representation of the voltage as a signed integer. The analog input process and the size of the resulting data type varies by FPGA target. For many FPGA targets, 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 RT host VI and convert the binary representation to a voltage or other physical quantity.

If you configure the FPGA I/O Node to write an analog output, the FPGA I/O Node might write the binary representation of the voltage to the digital-to-analog converter (DAC), which sets the analog output voltage. The size of the data type varies by FPGA target. You can generate voltage information in two sources-the RT host VI or the FPGA VI. Typically, the RT host VI converts the voltage to an appropriate 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 the appropriate binary representations. In both cases, the DAC produces a voltage that corresponds to the binary representation.

Most of the math functions on the Numeric palette support both integer and fixed-point data types. Notable exceptions include the Divide, Reciprocal and Square Root functions, which will only work on the FPGA target if you are using fixed-point data.

It is possible to achieve some of the functionality of the Divide function for integer data if you make use of the Scale by Power of 2 and Quotient & Remainder functions. You use these functions to perform simple scaling of data, as shown in Figure 4-4.

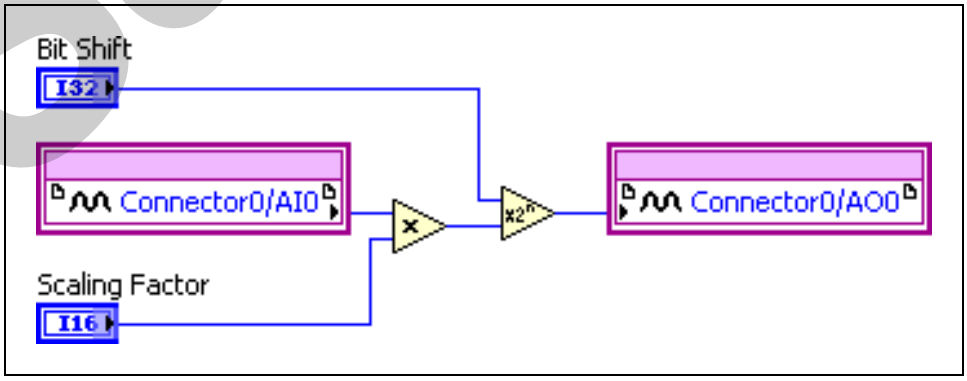


**Figure 4-4.** R Series Scaling on FPGA

In this example of scaling data in LabVIEW FPGA, we want to scale the data by .70. To get this scaling very close, we multiply the analog input by a scaling factor of 11500, then divide it by $2^{14}$ by using the Scale by the Power of 2 function with the Bit Shift (n input of the function) set to –14. This results in a the data from Connector0/AI0 being multiplied by 0.7019.

Additionally, you can divide numbers using the quotient and remainder function, however, this can use many slices when implemented.

### Converting Binary Representations

When you configure the FPGA I/O Node to read an analog input, the FPGA I/O Node initiates a conversion, waits for the result, and returns the binary, uncalibrated representation of the voltage as a signed 32-bit integer (I32). The analog input process and the size of the resultant data type varies by FPGA target.

The equation that converts the binary representation to a physical quantity depends on the FPGA target and transducer. Avoid executing this calculation in the FPGA VI because the R Series FPGA supports only integer operations, and performing the calculation consumes space on the FPGA.

It is good practice to convert and calibrate the I/O values in a host VI.

## E. Fixed-Point Math

The fixed-point data type is a numeric data type that represents a set of rational numbers using binary digits, or bits. Unlike the floating-point data type, which allows the total number of bits LabVIEW uses to represent numbers to vary, you can configure fixed-point numbers to always use a specific number of bits. Hardware and targets that only can store and process data with a limited or fixed number of bits then can store and process the numbers. You can specify the range and precision of fixed-point numbers. You can specify any size between 1 and 64 bits, inclusive, for a fixed-point number. You can configure fixed-point numbers as signed or unsigned.

**Note** To represent a rational number using the fixed-point data type, the denominator of the rational number must be a power of 2, because the binary number system is a base-2 number system.

Use the fixed-point data type when you do not need the dynamic functionality of floating-point representation, or when you want to work with a target that does not support floating-point arithmetic, such as an FPGA target.

The fixed-point data type provides some of the flexibility of the floating-point data type but also maintains the size and speed advantages of integer arithmetic. By default, each operation on the fixed-point data type generates a fixed-point result that is large enough to hold all possible output values specified by the input types.

**Note** FPGA Signal Generation VIs and some functions do not support the fixed-point data type.

**Caution** If you wire a fixed-point number to an integer, you might lose fractional bits.

The range of a fixed-point data type describes the minimum value, maximum value and delta (the smallest change in data). These three values are determined by the sign encoding, word length, and integer word length of the fixed-point data.

## Encoding, Word Length, and Integer Word Length

The sign encoding, word length, and integer word length are the three parameters that define the values that can be represented by a fixed-point number. These parameters are all user-defined and can be modified by right-clicking on a fixed-point constant, control, or function and selecting Properties.

- **Sign encoding**—The setting that specifies whether the fixed-point value is signed or unsigned. If you select signed, the sign bit is always the first bit in the bit string that represents the data.
- **Word length**—The total number of bits in the bit string that LabVIEW uses to represent all possible values of the fixed-point data. LabVIEW accepts a maximum word length of 64 bits. Certain targets might limit data to smaller word lengths. If you open a VI on a target and the VI contains fixed-point data with larger word lengths than the target can accept, the VI contains broken wires. Refer to the documentation for a target to determine the maximum word length the target accepts.
- **Integer word length**—The number of integer bits in the bit string that LabVIEW uses to represent all possible values of the fixed-point data, or, given an initial position to the left or right of the most significant bit, the number of bits to shift the binary point to reach the most significant bit. The integer word length can be larger than the word length, and can be positive or negative.

When performing calculations with fixed-point data, it is important to track the configuration of the fixed-point data as the data is manipulated. Most operations result in a different configuration of fixed-point data at the output than the input. The easiest way to view the configuration of the fixed-point data, as well as the minimum, maximum and delta values, is to turn on the Context Help and click on the a wire, control, or indicator. Figure 4-5 displays the data type of a particular wire.

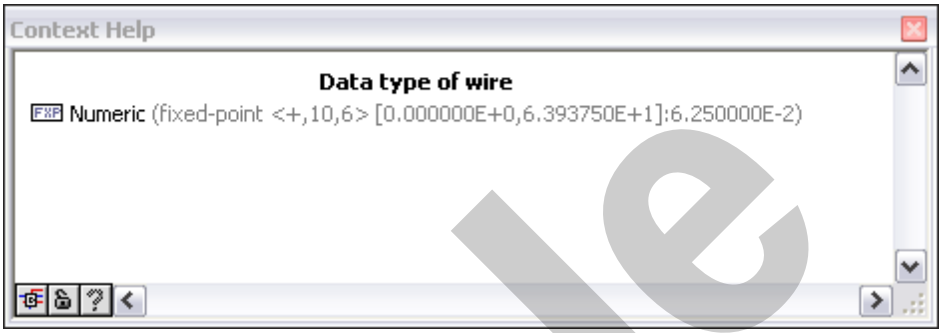


**Figure 4-5.** Fixed Point Context Help

## Numeric Representation of Fixed-Point Numbers

The maximum and minimum values of a fixed-point number are dependent upon the sign encoding, word length, and integer word length of the number. The delta of the number is based on the difference between the word length and the integer word length.

**Table 4-2.** Numeric Representation Examples

| Representation | Delta | Minimum Value | Maximum Value |
|---|---|---|---|
| U8 | 1 | 0 | 255 |
| I8 | 1 | –128 | 127 |
| FXP <+,8,7> | 0.5 | 0 | 127.5 |
| FXP <+,8,6> | 0.25 | 0 | 63.75 |
| FXP <±,8,7> | 0.5 | –64 | 63.5 |
| FXP <±,8,6> | 0.25 | –32 | 31.75 |
| FXP <+,8,0> | 0.0039 | 0 | 0.9961 |

When the fixed-point number is unsigned, it is represented as <+,X,Y>. For an unsigned fixed-point number, the maximum value of the fixed-point data is $2^Y - 1/(2^{(X-Y)})$ and the delta is $1/(2^{(X-Y)})$. The minimum value that can be represented is zero. For example:

<+,8,6> is an unsigned 8-bit number with six integer bits and two decimal bits. The maximum value that can be represented is $2^6 - 1/(2^{(8-6)}) = 64 - ¼ = 63.75$ and the delta is $1/(2^{(8-6)}) = 0.25$.

When the fixed-point number is signed, it is represented as <±,X,Y>. In this case, the maximum value of the fixed-point data is $2^{Y-1} - 1/(2^{(X-Y)})$. The delta is still represented as $1/(2^{(X-Y)})$. The minimum value that can be represented is $-2^{Y-1}$. For example:

<±,8,6> is a signed 8-bit number with six integer bits. The maximum value that can be represented is $2^{6-1} - 1/(2^{(8-6)}) = 31.75$ and the delta is $1/(2^{(8-6)}) = 0.25$. The minimum value that can be represented is $-2^{6-1} = -32$.

**Note** The difference between the maximum and minimum values for a signed fixed-point number is the same as the difference between the maximum and minimum values for an unsigned fixed-point number with the same word and integer word lengths. The delta also does not change.

If the integer word length is larger than the word length, LabVIEW does not store the integer bits that exceed the word length. For <+,8,10>, the two least significant bits would not be stored. The maximum value that can be represented by this number is $2^{10} - 1/(2^{(8-10)}) = 1020$ and the delta is $1/(2^{(8-10)}) = 4$.

If the integer word length is negative, LabVIEW does not store any integer bits and also does not store the number of fractional bits equal to the negative number, starting from the binary point. Thus, for <+,8,–2>, the maximum value that can be represented is $2^{-2} - 1/(2^{(8+2)}) = 0.2490234375$ and the delta is $1/(2^{(8+2)}) = 9.765625E\text{-}4$.

## Fixed-Point Configuration

You can configure a fixed-point number in a numeric control, constant, or indicator.

After you configure a fixed-point control, constant, or indicator, that object cannot display a number that does not conform to the settings you specify. If the object you configure is an indicator, LabVIEW coerces any input value to the indicator to conform to the fixed-point configuration settings of the indicator.

## Configuring Controls

Complete the following steps to configure a fixed-point number in a numeric control.

1. Right-click the control and select **Properties** from the shortcut menu to display the Numeric Properties dialog box.

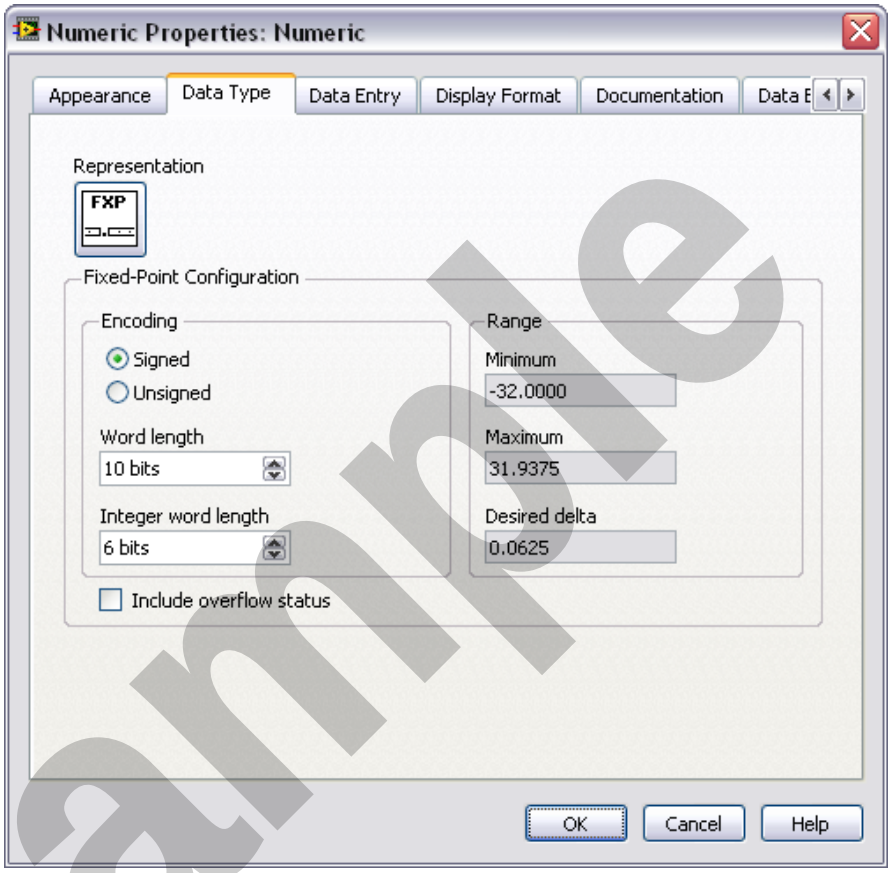


**Figure 4-6.** Fixed-Point Control Configuration

2. On the Data Type page, click the data type icon in the Representation section and select **FXP (Fixed-point)** from the shortcut menu. The Fixed-Point Configuration section displays default values for the Encoding and Range options.
3. Complete the following steps to configure the Range of the fixed-point number.
   a. Select **Signed** or **Unsigned** to specify whether you want to represent a signed or unsigned number.
   b. In the **Word length** field, specify the total number of bits you want to use to represent the value of the fixed-point number.
   c. In the **Integer word length** field, specify the number of integer bits you want to use to represent the value of the fixed-point number.

4. (Optional) Place a checkmark in the **Include overflow status** checkbox to include an overflow status in the fixed-point number.
5. Click **OK** to close the dialog box and apply the configuration settings.

## Configuring Constants

Complete the following steps to configure a fixed-point number in a numeric constant.

1. Right-click the constant and select **Properties** from the shortcut menu to display the Numeric Constant Properties dialog box.
2. On the Data Type page, remove the checkmark from the **Adapt to entered data** checkbox.

**Note** When you select **Adapt to entered data**, LabVIEW displays any value you enter with the shortest possible word length and integer word length. You must remove the checkmark from this checkbox if you want to make changes to the Fixed-Point Configuration settings.

3. Click the data type icon in the Representation section and select **FXP (Fixed-point)** from the shortcut menu. The Fixed-Point Configuration section displays default values for the Range and Encoding options.
4. Complete the following steps to configure the Range of the fixed-point number.
   a. Select **Signed** or **Unsigned** to specify whether you want to represent a signed or unsigned number.
   b. In the **Word length** field, specify the total number of bits you want to use to represent the value of the fixed-point number.
   c. In the **Integer word length** field, specify the number of integer bits you want to use to represent the value of the fixed-point number.
5. (Optional) Place a checkmark in the **Include overflow status** checkbox to include an overflow status in the fixed-point number.
6. Click **OK** to close the dialog box and apply the configuration settings.

## Configuring Indicators

Complete the following steps to configure a fixed-point number in a numeric indicator.

1. Right-click the indicator and select **Properties** from the shortcut menu to display the Numeric Properties dialog box.
2. On the Data Type page, click the data type icon in the Representation section and select **FXP (Fixed-point)** from the shortcut menu. The Fixed-Point Configuration section displays default values for the Range and Encoding options.

3. Place a checkmark in the **Adapt to source** checkbox if you want the value to inherit the fixed-point configuration settings of an input fixed-point value. If you select this option, skip to step 6.
4. (Optional) Complete the following steps to configure the Range of the fixed-point number.
   a. In the **Minimum** field, enter the minimum value to which you want the fixed-point number to conform.
   b. In the **Maximum** field, enter the maximum value to which you want the fixed-point number to conform.
   c. In the **Desired delta** field, enter the increment between numbers within the Range.
5. (Optional) Place a checkmark in the **Include overflow status** checkbox to include an overflow status in the fixed-point number.
6. Click **OK** to close the dialog box and apply the configuration settings.

## Fixed-Point Arithmetic

When performing arithmetic on integer data, the output data type matches the input data types. However, when you add two values that are the maximum values allowed for that data type, the result will not be accurate. For example, adding together two U16 integers with a value of 65535 should result in a value of 131070. However, that value is outside of the range of values allowed by a U16, so the result is an inaccurate output.

For fixed-point arithmetic, the output of the function is configured to accommodate the largest possible result based on the inputs. The size of the output is limited to 64 bits, which is a limitation of the data type. LabVIEW propagates the range of values along each wire in the diagram, not fixed-point representation. This allows LabVIEW to reduce resources when possible. For terminals and other points when representation is necessary, the representation is then calculated to be as small as possible without losing data.

In the following sections, we examine four arithmetic functions: addition, subtraction, multiplication, and division. Because LabVIEW propagates ranges, the formulas in the following sections are helpful guidelines for calculating the representation of an output value, but are not the formulas that LabVIEW uses.

### Addition

When using the Add function, the two inputs should have the same fixed-point configuration. If they have different configurations, then one or both of the inputs will be coerced to a configuration that can accommodate the full range of both inputs. For the output, the word length and integer

word length are each increased by one and the sign encoding matches the setting of the inputs. Thus, if the inputs are each configured as <±,A,B>, then the output will be configured as <±,A+1,B+1>. Figure 4-7 shows an example of addition of fixed point data.

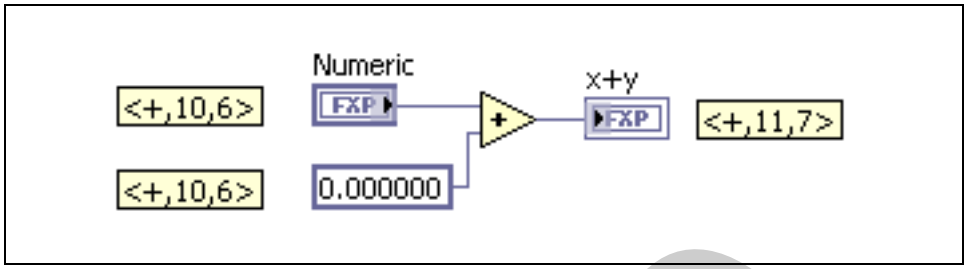


**Figure 4-7.** Fixed-Point Addition

For the output, integer word length increases by one to accommodate the maximum value. Since the integer word length increases, the word length must also increase so that no accuracy is lost. Thus, <±,10,6> + <±,10,6> = <±,11,7>

To add two fixed-point values with different configurations, you must use the longer integer word length from the two inputs for the output, and you must add together the longer word length and the longer decimal word length (word length - integer word length) to determine the word length of the output. This is done to ensure that no accuracy is lost.

For example, adding fixed-point values with configurations of <±,10,7> and <±,10,6> requires that the output have 7 integer bits (from <±,10,7>) and 4 decimal bits (from <±,10,6>), resulting in both inputs being coerced to <±,11,7>. For the output, <±,11,7> + <±,11,7> = <±,12,8>

If one input is signed and the other is unsigned, then the word length and integer word length will increase by one more bit.

## Subtraction

When using the Subtract function, the two inputs should have the same fixed-point configuration, as described for addition. Also, as with addition, the input word length and integer word length each be increase by one for the output. Unlike addition, the output is always signed, regardless of whether or not the inputs were signed. Thus, if the inputs are each configured as <+,A,B>, then the output is configured as <±,A+1,B+1>. Figure 4-8 shows an example of subtraction of fixed point data.

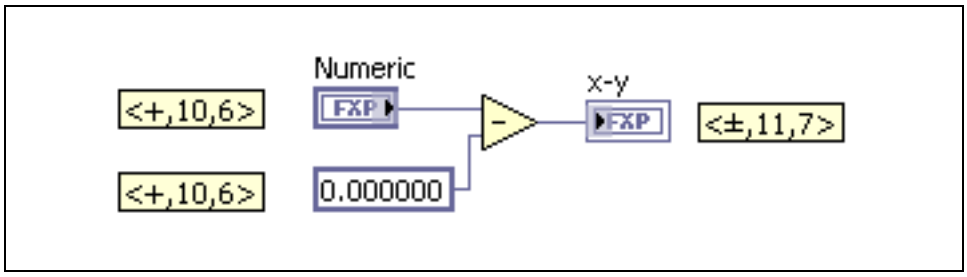


**Figure 4-8.** Fixed-Point Subtraction

As with addition, if the inputs have different configurations, they are coerced to a common configuration that accommodates both data types.

## Multiplication

When using the Multiply function, the two inputs are not required to have the same fixed-point configuration. For the integer value of the output to accommodate the largest possible value of each operand, the integer word length must increase to match the sum of the integer word lengths of the inputs. For the resolution of the output to accommodate the highest resolution of both inputs, the decimal word length must also increase. Thus, the word length of the output must increase to match the sum of the word lengths of the inputs. If either input is signed, then the output is signed. Otherwise, the output is unsigned. Thus, if the inputs are each configured as <±,A,B> and <+,C,D>, then the output is configured as <±,A+C,B+D>. Figure 4-9 shows an example of multiplication of fixed point data.

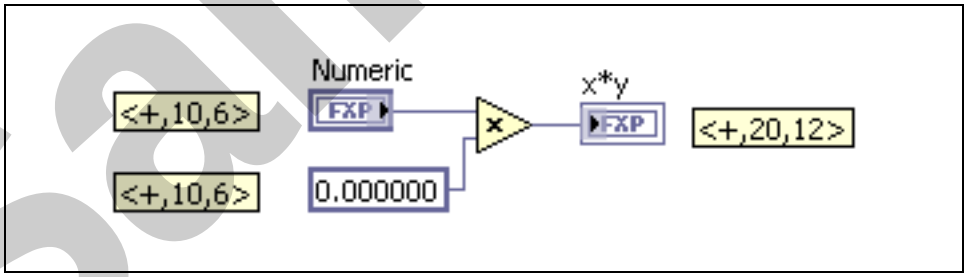


**Figure 4-9.** Fixed-Point Multiplication

## Division

When using the Divide function, the two inputs are not required to have the same fixed-point configuration. For the output, the word length and integer word length are calculated differently depending on whether or not the inputs are signed.

- Signed: <±,A,B> + <±,C,D> = <±,A+C+1,B+C–D+1>
- Unsigned: <+,A,B> + <+,C,D> = <+,A+C, B+C–D>

An example of both signed and unsigned fixed-point division is shown in Figure 4-10.

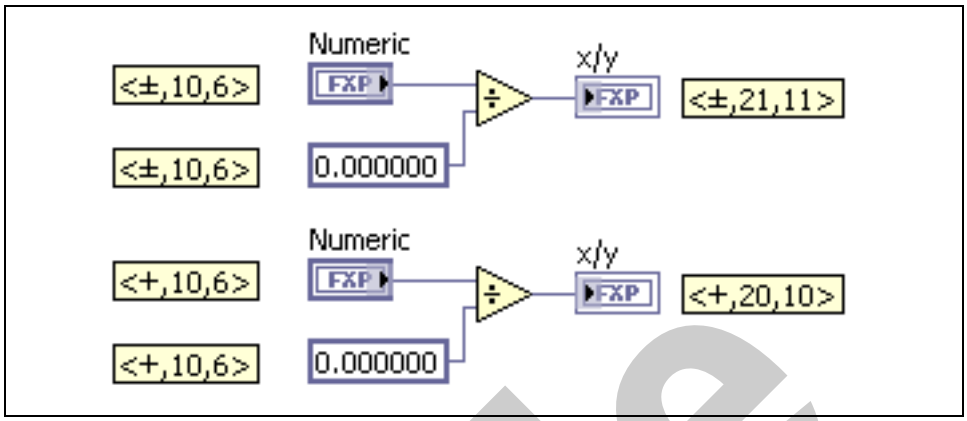


**Figure 4-10.** Fixed-Point Division

## Configuring Fixed-Point Functions

Functions that support the fixed-point data type (add, subtract, multiply, and so on) include modes to handle the overflow and rounding. In the Properties dialog box for the function, you select the overflow and rounding modes. Right-click a function and select Properties from the shortcut menu to display the Properties dialog box.

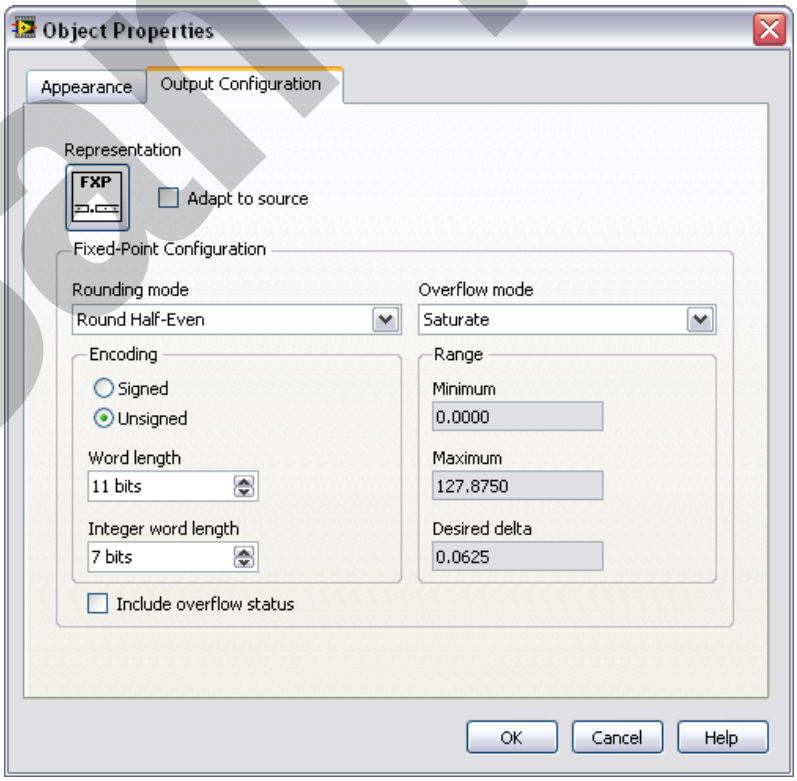


**Figure 4-11.** Fixed-Point Function Configuration

The Properties dialog box contains the following three options that are specific to configuring a fixed-point function:

- **Adapt to Source**—Sets the configuration settings for the output value to adapt to the input values you wire to the function. For fixed-point data, LabVIEW automatically sets the Fixed-Point Configuration settings to avoid data loss, if possible.
- **Rounding Mode**—Determines how the function handles quantization conditions.
- **Overflow Mode**—Determines how the function handles overflow conditions.

Rounding occurs when the precision of the input value or the result of an operation is greater than the precision of the output type. The rounding modes affect the logic generated within the FPGA as follows:

- **Truncate**—Removes fractional bits and therefore does not require any additional hardware resources. However, this mode produces the largest mean error for most data streams. This mode is the default for integer operations.
- **Round-Half-Up (Asymmetric)**—Adds to the least significant bit and therefore requires an adder that is the width of the output type.
- **Round-Half-Even**—Requires the most hardware resources and has the slowest timing of the three rounding modes. However, this mode returns the most statistically correct results for most data streams and is therefore the default rounding mode for the fixed-point data type. Although this mode requires only a modest amount of additional hardware compared to the Round-Half-Up (Asymmetric) mode, the length of the longest combinatorial path is longer, which reduces the maximum clock frequency for the path.

Overflow occurs when the result of an operation is outside the range of values that the output type can represent. The overflow modes affect the logic generated within the FPGA as follows:

- **Saturate**—Requires additional hardware resources to determine if the input value is within the desired range of the output type. Saturate is the default overflow mode. If you specify a desired range other than the default minimum and maximum, the Saturate mode performs a full comparison between the desired range and input value, which might require additional clock cycles to complete. If you do not specify a desired range, the Saturate mode checks all bits above the most significant bit of the output type to ensure the output did not overflow.
- **Wrap**—Requires fewer hardware resources than the Saturate mode.

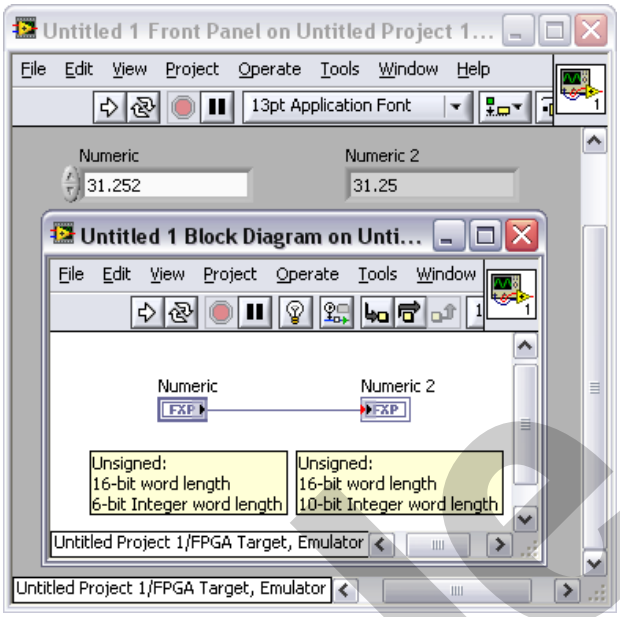


**Figure 4-12.** Mismatched Fixed-Point Data

If you select the Saturate, Round-Half-Up (Asymmetric), or Round-Half-Even modes and the output can handle the overflow or rounding, the operation does not require additional hardware resources. If a coercion dot does not appear on the block diagram, the output can handle the overflow or rounding without additional hardware resources. If a coercion dot does appear, then there is a risk of data loss.

For more information about fixed-point data and fixed-point arithmetic, refer to the following *LabVIEW Help* topics:

- *Numeric Data*
- *Configuring Fixed-Point Numbers*
- *Numeric Data Types Table*

# F. CompactRIO

One of the differentiating characteristics of CompactRIO is the high degree of customization that can be achieved using modules that fit your specific needs. Each CompactRIO module has different functional features and specifications. Similarly, the input/output for each module has its own features.

Different modules return data in different fixed-point data types. Differences in configuration are due to differences in the hardware for each module. Hardware differences result in variations in accuracy and range for the data that is generated or obtained. Some modules enable you to configure their I/O nodes to return data that is both calibrated and scaled.

All modules share certain common properties, like Module ID, Serial Number, and Vendor ID. Some modules, however, have their own unique properties. For example, the NI 9233 has a Data Rate property that allows the user to programmatically configure the sample rate of the device. Some modules even have their own methods that can be called. For example, the NI 9263 analog output module has three methods: Update, Wait For Update, and Write Data. The NI 9233, however, does not have any module-specific methods associated with it.

Different modules have different timing and sampling capabilities. For example, the NI 9211 acquires a single sample per cycle at its set data rate. The NI 9233 is able to acquire a single sample from each of its channels at a user-defined sample rate.

It is important to understand the features and specifications of the CompactRIO modules you use. Refer to the *LabVIEW Help* for module-specific information.

## G. Error Handling

To add standard LabVIEW error in and error out parameters to a function, you can right-click the FPGA I/O Node on the block diagram and select **Show Error Terminals**. If an error occurs, you might receive incorrect data. Add error parameters to be sure the data you receive is valid. FPGA targets might report errors differently. Refer to the specific FPGA target hardware documentation for information about how specific FPGA targets report errors.

Adding error in and error out parameters increases the amount of space the function uses on the FPGA target. The error in and error out parameters also slows execution on the FPGA target. So, use them as a general rule, but if you have difficulty with either the size or the speed of the FPGA application, remove them after carefully testing your code.

If FPGA resources are limited, there are number of optimizations related to the use of error clusters that you can perform.

- If you are using the error wire for dataflow, remove the error wires and use a Sequence Structure instead. The Sequence Structure has a minimal cost when used in FPGA.
- Show error terminals only for modules whose functions are critical to system operation.
- Show terminals only once per module if multiple calls are made to the module. The exception to this rule is modules that have specific errors. Checking once allows you to check for general errors.

- Do not pass error clusters through the program or display the error cluster on the front panel.
- Unbundle source and/or code items and handle the errors immediately.

Some types of FPGA I/O errors must be handled in all cases.

- All safety-critical I/O operations should be monitored for errors.
- Some modules have specific error codes associated with them. For example, the NI CAN modules generate an error if a timeout occurs. The NI 9802 generates an error if it attempts to open a file that cannot be found.
- If the module being used has been removed or is not secure, an error is generated.

# Self Review Quiz

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

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

2. What is the default data type of the result of adding two signed fixed-point numbers with a word length of 20 and an integer word length of 10?
   a. <±, 40, 20>
   b. <+, 20, 10>
   c. <±, 21, 11>
   d. <+, 30, 30>

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

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

| | |
|---|---|
| NI PCI-7831R R Series board | Integer |
| NI 9233 CompactRIO module | Fixed-point |

# Self Review Quiz Answers

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

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

2. What is the default data type of the result of adding two signed fixed-point numbers with a word length of 20 and an integer word length of 10?
   a. <±, 40, 20>—This answer would have been correct if the values were being multiplied.
   b. <+, 20, 10>
   **c. <±, 21, 11>**—Since the inputs have the same data type, the integer word length and word length are each increased by one.
   d. <+, 30, 30>

3. Which of the following are parameters used to define the range of values that are represented by a fixed-point number?
   a. Terminal
   **b. Sign Encoding**—Determines whether the value is signed or unsigned.
   **c. Word Length**—Determines the number of bits used to represent the number.
   **d. Integer Word Length**—Determines the number of bits used to represent the integer portion of the number.

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

| | |
|---|---|
| NI PCI-7831R R Series board | **Integer** |
| NI 9233 CompactRIO module | **Fixed-point** |

# 4

# FPGA I/O

## Exercise 4-1 R Series I/O

### Goal

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

### Scenario

You must design an application to access the analog and digital I/O of an R Series device using FPGA I/O nodes. Since you do not have an R Series device on-hand, you must simulate the PCI-7831R that will be used in the final application.

### Design

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

### Implementation

1. Create a new project with a PCI-7831R board as the FPGA Target. You created a similar project in Exercise 2-3.
2. Select **File»Save Project** and save the project as `R-Series IO.lvproj` in the `<Exercises>\LabVIEW FPGA\R-Series IO\` directory.
3. Set the project to execute the VI on the development computer with simulated I/O.
   - ❑ Right-click the FPGA target and select **Execute VI on» Development Computer with Simulated I/O**.
   - ❑ Click **OK**.

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

4. Add AI0 and DIOPORT0 from Connector0 to the FPGA Target.

   ❑ Right-click on the FPGA Target and select **New»FPGA I/O**.

   ❑ In the New FPGA I/O dialog box, expand **Connector0**.

   ❑ Add **AI0** and **DIOPORT0**.

   ❑ Click **OK**. You should now see a Connector0 virtual folder containing two FPGA I/O items in your Project Explorer.

5. Assign unique names to the FPGA I/O resources. The use of unique names for FPGA I/O resources will result in more readable block diagrams.

   ❑ Right-click Connector0/AI0 and select **Rename**. Name the item `Simulated Analog In`.

   ❑ Rename Connector0/DIOPORT0 as `Simulated Digital Port In`.

6. Create a new VI on the FPGA Target.

   ❑ Right-click the target and select **New»VI**.

   ❑ Save the VI as `R-Series IO.vi` in the `<Exercises>\LabVIEW FPGA\R-Series IO\` directory.

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

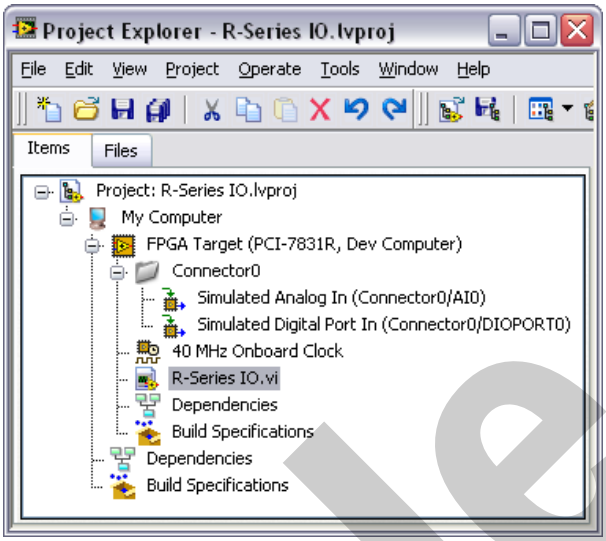


**Figure 4-1.** R Series Project with FPGA I/O

8. Build the Block Diagram shown in Figure 4-2.

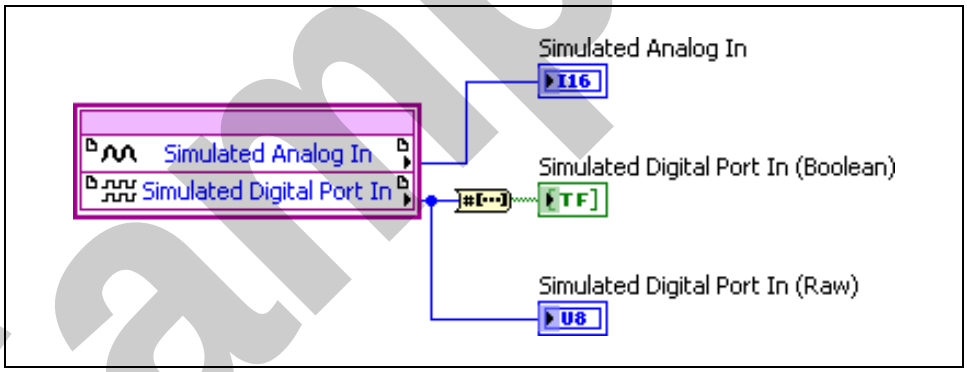


**Figure 4-2.** R-Series I/O.vi Block Diagram

❑ Add the FPGA I/O node to the block diagram

– Drag **Simulated Analog In** from the Project Explorer window to the block diagram.

– Expand the node to include **Simulated Digital Port In**.

❑ Right-click the I/O Item outputs and create indicators for each.

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

9. Arrange the Front Panel shown in Figure 4-3.

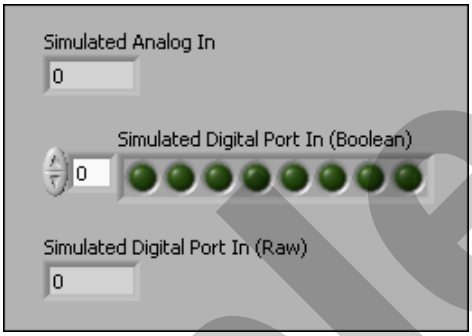


**Figure 4-3.** R-Series I/O.vi Front Panel

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

10. Save the VI.

## Testing

1. Run the VI.

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

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

2. Save and close the VI and project.

## End of Exercise 4-1

# Exercise 4-2 CompactRIO I/O

## Goal

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

## Scenario

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

## Design

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

## Implementation

1. Create a new project with a cRIO-9074 as the FPGA Target. You created a similar project in Exercise 2-4.
2. Select **File»Save Project** and save the project as `CompactRIO IO.lvproj` in the `<Exercises>\LabVIEW FPGA\CompactRIO IO\` directory.
3. Create a new VI on the FPGA Target.
   - ❑ Save the VI as `CompactRIO IO.vi` in the `<Exercises>\LabVIEW FPGA\CompactRIO IO\` directory.
4. Rename Mod1/TC0 and Mod1/TC1 to Thermocouple 0 and Thermocouple 1, respectively.
   - ❑ Expand **Mod 1** and right-click on **Mod1/TC0**. Select **Rename** and enter `Thermocouple 0`.
   - ❑ Right-click on **Mod1/TC1**. Select **Rename** and enter `Thermocouple 1`.

The Project Window should now resemble Figure 4-4.

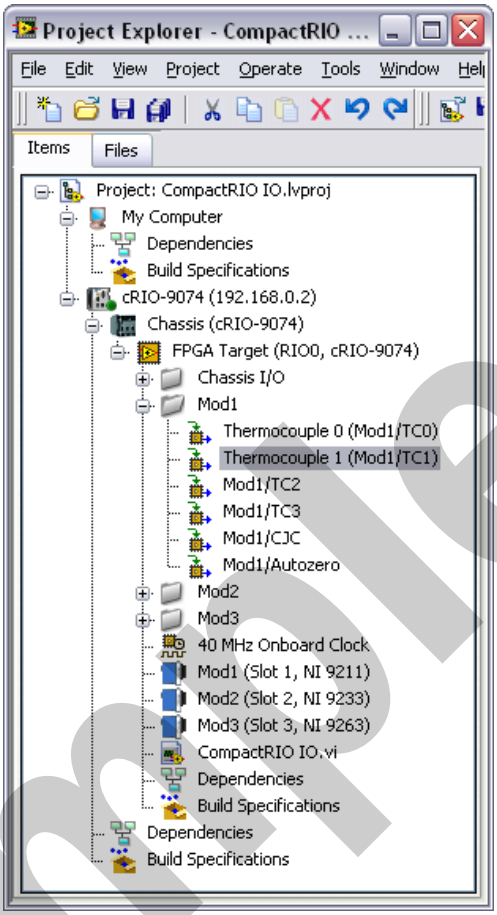


**Figure 4-4.** CompactRIO Project with FPGA I/O

5. Build the Block Diagram shown in Figure 4-5.

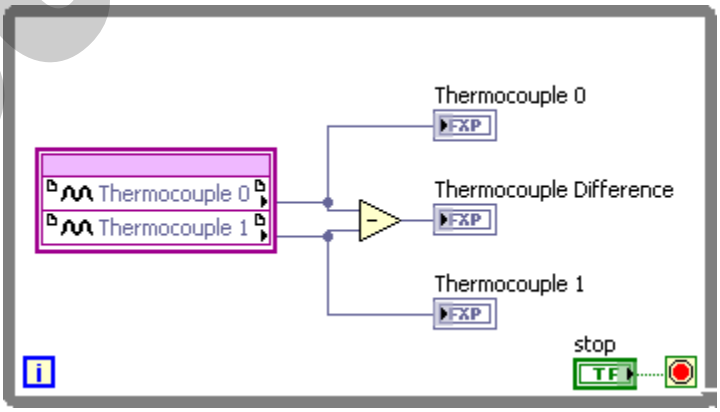


**Figure 4-5.** CompactRIO IO Block Diagram

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

6. Observe the effect of the Subtract function on the fixed-point data type.

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

  Thermocouple 0 _____

  Thermocouple 1 _____

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

  Thermocouple Difference_____

7. Arrange the Front Panel shown in Figure 4-6.

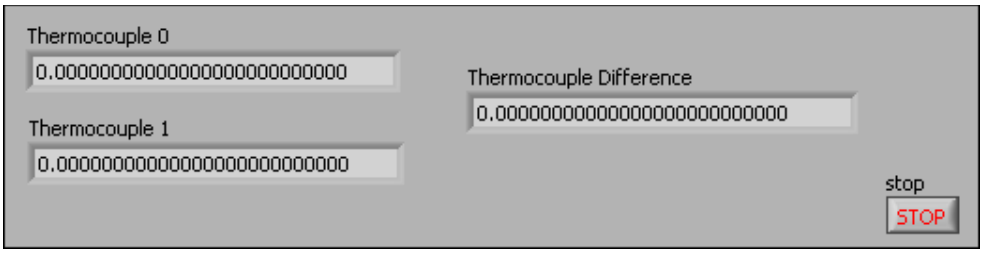


**Figure 4-6.** CompactRIO IO Front Panel

8. Save the VI.

## Testing

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

**Note** This data is calibrated, but it has not been scaled to Fahrenheit or Centigrade scales. You will perform this conversion later in Exercise 8-2.

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

3. Save and Close the VI and project.

## End of Exercise 4-2