FlexRIO™

Adapter Module Development Kit User Manual

August 2020
372544L-01

Worldwide Technical Support and Product Information

ni.com

Worldwide Offices

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

National Instruments Corporate Headquarters

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

For further support information, refer to the *NI Services* appendix. To comment on National Instruments documentation, refer to the National Instruments website at ni.com/info and enter the Info Code feedback.

© 2008–2020 National Instruments Corporation. All rights reserved.

Legal Information

Limited Warranty

This document is provided ‘as is’ and is subject to being changed, without notice, in future editions. For the latest version, refer to `ni.com/manuals`. NI reviews this document carefully for technical accuracy; however, NI MAKES NO EXPRESS OR IMPLIED WARRANTIES AS TO THE ACCURACY OF THE INFORMATION CONTAINED HEREIN AND SHALL NOT BE LIABLE FOR ANY ERRORS.

NI warrants that its hardware products will be free of defects in materials and workmanship that cause the product to fail to substantially conform to the applicable NI published specifications for one (1) year from the date of invoice.

For a period of ninety (90) days from the date of invoice, NI warrants that (i) its software products will perform substantially in accordance with the applicable documentation provided with the software and (ii) the software media will be free from defects in materials and workmanship.

If NI receives notice of a defect or non-conformance during the applicable warranty period, NI will, in its discretion: (i) repair or replace the affected product, or (ii) refund the fees paid for the affected product. Repaired or replaced Hardware will be warranted for the remainder of the original warranty period or ninety (90) days, whichever is longer. If NI elects to repair or replace the product, NI may use new or refurbished parts or products that are equivalent to new in performance and reliability and are at least functionally equivalent to the original part or product.

You must obtain an RMA number from NI before returning any product to NI. NI reserves the right to charge a fee for examining and testing Hardware not covered by the Limited Warranty.

This Limited Warranty does not apply if the defect of the product resulted from improper or inadequate maintenance, installation, repair, or calibration (performed by a party other than NI); unauthorized modification; improper environment; use of an improper hardware or software key; improper use or operation outside of the specification for the product; improper voltages; accident, abuse, or neglect; or a hazard such as lightning, flood, or other act of nature.

THE REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND THE CUSTOMER’S SOLE REMEDIES, AND SHALL APPLY EVEN IF SUCH REMEDIES FAIL OF THEIR ESSENTIAL PURPOSE.

EXCEPT AS EXPRESSLY SET FORTH HEREIN, PRODUCTS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND AND NI DISCLAIMS ALL WARRANTIES, EXPRESSED OR IMPLIED, WITH RESPECT TO THE PRODUCTS, INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE OR NON-INFRINGEMENT, AND ANY WARRANTIES THAT MAY ARISE FROM USAGE OF TRADE OR COURSE OF DEALING. NI DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE OF OR THE RESULTS OF THE USE OF THE PRODUCTS IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE. NI DOES NOT WARRANT THAT THE OPERATION OF THE PRODUCTS WILL BE UNINTERRUPTED OR ERROR FREE.

In the event that you and NI have a separate signed written agreement with warranty terms covering the products, then the warranty terms in the separate agreement shall control.

Copyright

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

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

End-User License Agreements and Third-Party Legal Notices

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

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

U.S. Government Restricted Rights

If you are an agency, department, or other entity of the United States Government (“Government”), the use, duplication, reproduction, release, modification, disclosure or transfer of the technical data included in this manual is governed by the Restricted Rights provisions under Federal Acquisition Regulation 52.227-14 for civilian agencies and Defense Federal Acquisition Regulation Supplement Section 252.227-7014 and 252.227-7015 for military agencies.

Trademarks

Refer to the *NI Trademarks and Logo Guidelines* at `ni.com/trademarks` for more information on National Instruments trademarks.

ARM, Keil, and µVision are trademarks or registered of ARM Ltd or its subsidiaries.

LEGO, the LEGO logo, WEDO, and MINDSTORMS are trademarks of the LEGO Group.

TETRIX by Pitsco is a trademark of Pitsco, Inc.

FIELDBUS FOUNDATION™ and FOUNDATION™ are trademarks of the Fieldbus Foundation.

EtherCAT® is a registered trademark of and licensed by Beckhoff Automation GmbH.

CANopen® is a registered Community Trademark of CAN in Automation e.V.

DeviceNet™ and EtherNet/IP™ are trademarks of ODVA.

Go!, SensorDAQ, and Vernier are registered trademarks of Vernier Software & Technology. Vernier Software & Technology and vernier.com are trademarks or trade dress.

Xilinx is the registered trademark of Xilinx, Inc.

Taptite and Trilobular are registered trademarks of Research Engineering & Manufacturing Inc.

FireWire® is the registered trademark of Apple Inc.

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

Handle Graphics®, MATLAB®, Real-Time Workshop®, Simulink®, Stateflow®, and xPC TargetBox® are registered trademarks, and TargetBox™ and Target Language Compiler™ are trademarks of The MathWorks, Inc.

Tektronix®, Tek, and Tektronix, Enabling Technology are registered trademarks of Tektronix, Inc.

The Bluetooth® word mark is a registered trademark owned by the Bluetooth SIG, Inc.

The ExpressCard™ word mark and logos are owned by PCMCIA and any use of such marks by National Instruments is under license.

The mark LabWindows is used under a license from Microsoft Corporation. Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

Other product and company names mentioned herein are trademarks or trade names of their respective companies.

Members of the National Instruments Alliance Partner Program are business entities independent from National Instruments and have no agency, partnership, or joint-venture relationship with National Instruments.

Patents

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

Export Compliance Information

Refer to the *Export Compliance Information* at ni.com/legal/export-compliance for the National Instruments global trade compliance policy and how to obtain relevant HTS codes, ECCNs, and other import/export data.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

YOU ARE ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY AND RELIABILITY OF THE PRODUCTS WHENEVER THE PRODUCTS ARE INCORPORATED IN YOUR SYSTEM OR APPLICATION, INCLUDING THE APPROPRIATE DESIGN, PROCESS, AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

PRODUCTS ARE NOT DESIGNED, MANUFACTURED, OR TESTED FOR USE IN LIFE OR SAFETY CRITICAL SYSTEMS, HAZARDOUS ENVIRONMENTS OR ANY OTHER ENVIRONMENTS REQUIRING FAIL-SAFE PERFORMANCE, INCLUDING IN THE OPERATION OF NUCLEAR FACILITIES; AIRCRAFT NAVIGATION; AIR TRAFFIC CONTROL SYSTEMS; LIFE SAVING OR LIFE SUSTAINING SYSTEMS OR SUCH OTHER MEDICAL DEVICES; OR ANY OTHER APPLICATION IN WHICH THE FAILURE OF THE PRODUCT OR SERVICE COULD LEAD TO DEATH, PERSONAL INJURY, SEVERE PROPERTY DAMAGE OR ENVIRONMENTAL HARM (COLLECTIVELY, “HIGH-RISK USES”). FURTHER, PRUDENT STEPS MUST BE TAKEN TO PROTECT AGAINST FAILURES, INCLUDING PROVIDING BACK-UP AND SHUT-DOWN MECHANISMS. NI EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS OF THE PRODUCTS OR SERVICES FOR HIGH-RISK USES.

Compliance

Electromagnetic Compatibility Information

This hardware has been tested and found to comply with the applicable regulatory requirements and limits for electromagnetic compatibility (EMC) as indicated in the hardware's Declaration of Conformity (DoC)[1]. These requirements and limits are designed to provide reasonable protection against harmful interference when the hardware is operated in the intended electromagnetic environment. In special cases, for example when either highly sensitive or noisy hardware is being used in close proximity, additional mitigation measures may have to be employed to minimize the potential for electromagnetic interference.

While this hardware is compliant with the applicable regulatory EMC requirements, there is no guarantee that interference will not occur in a particular installation. To minimize the potential for the hardware to cause interference to radio and television reception or to experience unacceptable performance degradation, install and use this hardware in strict accordance with the instructions in the hardware documentation and the DoC[1].

If this hardware does cause interference with licensed radio communications services or other nearby electronics, which can be determined by turning the hardware off and on, you are encouraged to try to correct the interference by one or more of the following measures:

- Reorient the antenna of the receiver (the device suffering interference).
- Relocate the transmitter (the device generating interference) with respect to the receiver.
- Plug the transmitter into a different outlet so that the transmitter and the receiver are on different branch circuits.

Some hardware may require the use of a metal, shielded enclosure (windowless version) to meet the EMC requirements for special EMC environments such as, for marine use or in heavy industrial areas. Refer to the hardware's user documentation and the DoC[1] for product installation requirements.

When the hardware is connected to a test object or to test leads, the system may become more sensitive to disturbances or may cause interference in the local electromagnetic environment.

Operation of this hardware in a residential area is likely to cause harmful interference. Users are required to correct the interference at their own expense or cease operation of the hardware.

Changes or modifications not expressly approved by NI could void the user's right to operate the hardware under the local regulatory rules.

[1] The Declaration of Conformity (DoC) contains important EMC compliance information and instructions for the user or installer. To obtain the DoC for this product, visit `ni.com/certification`, search by model number or product line, and click the appropriate link in the Certification column.

Contents

Chapter 4
Interfacing Adapter Modules with NI-793xR and NI 797xR Devices

Chapter 5
Printed Circuit Board (PCB) Design Considerations

Chapter 6
Card Edge Connector

Chapter 7
Module Enclosure

Chapter 8
Installing the Adapter Module

Chapter 9
Configuring Your Adapter Module for Use with NI 795xR/796xR Modules and LabVIEW FPGA

Chapter 12
Creating Socketed Component-Level IP for Your Adapter Module and NI-793xR/NI 797xR Modules

Chapter 13
Designing and Debugging Component-Level IP

Appendix A
Signal Suggestions

Appendix B
Troubleshooting

Figures

Tables

About This Manual

The FlexRIO Adapter Module Development Kit enables third parties, such as system integrators, alliance members, and individual users, to create custom adapter modules for use with the following FlexRIO modules:

PXI modules:

- NI PXI-7951R (NI 7951R)
- NI PXI-7952R (NI 7952R)
- NI PXI-7953R (NI 7953R)
- NI PXI-7954R (NI 7954R)

PXI Express modules:

- NI PXIe-7961R (NI 7961R)
- NI PXIe-7962R (NI 7962R)
- NI PXIe-7965R (NI 7965R)
- NI PXIe-7966R (NI 7966R)
- NI PXIe-7971R (NI 7971R)
- NI PXIe-7972R (NI 7972R)
- NI PXIe-7975R (NI 7975R)
- NI PXIe-7976R (NI 7976R)

Controllers for FlexRIO:

- NI-7931R
- NI-7932R
- NI-7935R

The adapter module is necessary to customize the measurement I/O of the FlexRIO FPGA module or Controller for FlexRIO for a complete FlexRIO system.

The FlexRIO FPGA module and Controller for FlexRIO offer direct access to the pins of the FPGA, which allows for maximum speed and performance. This level of access to the FPGA requires careful attention to design detail when developing an adapter module to mate with the FlexRIO device. You can design custom adapter modules for the following development applications:

- Analog-to-digital converters (ADCs) and digital-to-analog converters (DACs)
- Low-voltage differential signal (LVDS) or low-voltage transistor-to-transistor logic (LVTTL) buffers to the device under test (DUT) connector
- Serial/custom protocol physical layers (PHYs), such as RS-485 and IEEE-1394

A complete adapter module consists of an enclosure containing a printed circuit board (PCB) with application-specific circuitry. This manual provides detailed information about the electrical and mechanical requirements of FlexRIO adapter module design.

Related Documentation

The following documents contain information that you may find helpful as you read this manual.

Table 1. FlexRIO Documentation Locations and Descriptions

Document	Location	Description
Getting started guide for your FPGA module	Available from the Start menu and at `ni.com/manuals`.	Contains installation instructions for your FlexRIO system.
Specifications document for your FPGA module	Available from the Start menu and at `ni.com/manuals`.	Contains specifications for your FPGA module.
Getting started guide for your Controller for FlexRIO	Available from the Start menu and at `ni.com/manuals`.	Contains installation instructions for your FlexRIO system.
Specifications document for your Controller for FlexRIO	Available from the Start menu and at `ni.com/manuals`.	Contains specifications for your Controller for FlexRIO.
NI-7931R/7932R/7935R User Manual	Available from the Start menu and at `ni.com/manuals`.	Contains instructions for creating applications for your NI-793*x*R Controller for FlexRIO.
Getting started guide for your adapter module	Available from the Start menu and at `ni.com/manuals`.	Contains installation instructions and signal information for your adapter module.
Specifications document for your adapter module	Available from the Start menu and at `ni.com/manuals`.	Contains specifications for your adapter module.

Table 1. FlexRIO Documentation Locations and Descriptions (Continued)

Document		Location	Description
FlexRIO Help		Available from the Start menu and at `ni.com/manuals`.	Contains information about the FlexRIO FPGA module, Controller for FlexRIO, FlexRIO adapter module, and CLIP configuration.
LabVIEW FPGA documentation	*FPGA Module* book	*LabVIEW Help*	Select **Help»Search the LabVIEW Help** in LabVIEW to view the *LabVIEW Help*. Browse to the **FPGA Module** book in the **Contents** tab for information about using the LabVIEW FPGA Module to create VIs that run on the FlexRIO FPGA module.
	Getting Started with the LabVIEW FPGA book	*FPGA Module* book in the *LabVIEW Help*	Provides links to the top resources that you can use to get started with LabVIEW FPGA.
	Integrating Third-Party IP (FPGA Module) book	*Integrating Third-Party IP (FPGA Module)* book in the *LabVIEW Help*	In the *LabVIEW Help*, select **FPGA Module» Integrating Third-Party IP** to access information about adding custom HDL code to your LabVIEW project.
	LabVIEW FPGA Module Release and Upgrade Notes	Available at `ni.com/manuals`. In LabVIEW you can also view this document by selecting **Start»All Programs»National Instruments» LabVIEW» LabVIEW Manuals**.	Contains information about installing the LabVIEW FPGA Module, describes new features, and provides upgrade information.

Table 1. FlexRIO Documentation Locations and Descriptions (Continued)

Document		Location	Description
LabVIEW Real-Time documentation	*Getting Started with the LabVIEW Real-Time Module*	Available at `ni.com/manuals`. You can also view this document by selecting **Start»All Programs»National Instruments» LabVIEW» LabVIEW Manuals**.	Provides exercises to teach you how to develop a real-time project and VIs, from setting up RT targets to building, debugging, and deploying real-time applications. This document provides references to the *LabVIEW Help* and other Real-Time Module as you create the real-time application.
	Real-Time Module book in the *LabVIEW Help*	Select **Help»Search the LabVIEW Help** in LabVIEW to view the *LabVIEW Help*. Browse the **Real-Time Module** book in the **Contents** tab.	Contains information about how to build deterministic applications using the LabVIEW Real-Time Module.
	LabVIEW Real-Time Module Release and Upgrade Notes	This document is available at `ni.com/manuals`. In LabVIEW, you can also view the LabVIEW Manuals directory that contains this document by selecting **Start»All Programs»National Instruments» LabVIEW» LabVIEW Manuals**.	Includes information about system requirements, installation, configuration, new features and changes, and compatibility issues for the LabVIEW Real-Time Module.

Additional Resources

The following resources contain information you might find helpful:

- *National Instruments Example Finder*—LabVIEW contains an extensive library of VIs and example programs for use with FlexRIO devices. To access the NI Example Finder, open LabVIEW and select **Help»Find Examples**, then select **Hardware Input and Output» FlexRIO**.
- `ni.com/flexrio`—Contains product information, and helpful links to the FlexRIO forum and the NI community for FlexRIO devices.
- *LabVIEW FPGA IPNet*—Offers resources for browsing, understanding, and downloading LabVIEW FPGA functions or Intellectual Property (IP). Use this resource to acquire IP that you need for your application, download examples to help learn programming techniques, and explore the depth of IP offered by the LabVIEW FPGA platform. To access the LabVIEW FPGA IPNet, visit `ni.com/ipnet`.
- Documentation available from Xilinx—Xilinx FPGA documentation provides information necessary for custom adapter module development, such as FPGA voltage limits and I/O standard specifications. Refer to Appendix C, *Xilinx Documentation References*, for specific Xilinx documentation recommendations.
- `ni.com/ask`—Allows you to create a new technical support request and access direct support from NI engineers. You may create a request to register your FlexRIO Adapter Module Development Kit, receive feedback on your custom module, or to obtain your unique Vendor ID. Refer to the Registration section of Chapter 1, *Before You Begin*, for more information about registering your adapter module.
- `ni.com/ipnet`—Contains LabVIEW FPGA functions and intellectual property to share.

FlexRIO Adapter Module Development Kit Features and Changes

Version 4.1

New Features

This release adds the following features to the FlexRIO Adapter Module Development Kit:

- Added support for the NI-7931R, NI-7932R, and NI-7935R devices.
- Added CLIP design best practices

Version 4.0

New Features

This release adds the following features to the FlexRIO Adapter Module Development Kit:

- Added support for the NI 7971R, NI 7972R, and NI 7976R devices.

Version 3.0

New Features

This release adds the following features to the FlexRIO Adapter Module Development Kit:

- Added support for the NI PXIe-7975R device.
- Added support for the `.fam` adapter module configuration file, as described in Chapter 9, *Configuring Your Adapter Module for Use with NI 795xR/796xR Modules and LabVIEW FPGA*.

Note The NI PXIe-797xR devices are only supported in FlexRIO Support 13.1 and later.

Fixed Issues

This release of the *FlexRIO Adapter Module Development Kit User Manual* fixes the following issues.

Table 2. Fixed Issues with the FlexRIO Adapter Module Development Kit, Version 3.0

CAR ID	Summary
344908	Added a note about using wildcards in `.ucf` files, and added information about using the CLIP Wizard to create custom adapter module CLIP or to edit existing adapter module CLIP.

Version 2.0

New Features

This release adds the following features to the FlexRIO Adapter Module Development Kit:

- Updated PCB, gold finger, and enclosure design files.
- Updated information about the new FlexRIO adapter module enclosure, as described in Chapter 5, *Printed Circuit Board (PCB) Design Considerations*.
- Updated information about gold finger connections and the use of X-traces, as described in the *Card Edge Connector Finishing* section of Chapter 6, *Card Edge Connector*.
- Updated information concerning appropriate PCB thickness, as described in Chapter 7, *Module Enclosure*.

Fixed Issues

This release of the *FlexRIO Adapter Module Development Kit User Manual* fixes the following issues.

Table 3. Fixed Issues with the FlexRIO Adapter Module Development Kit, Version 2.0

CAR ID	Summary
319666	Clarified the functionality of a LabVIEW FPGA Global Reset and clocking.
302743	Added inrush current limits to the DC Power Connector Pins table.
302746	Added information and an image detailing how to create X-traces between isolated gold finger pairs.
302742	Increased PCB thickness for improved connectivity.
215287	Clarified the I^2C address for reading and writing to the EEPROM.
209674	Updated the regulations on the power rails.

Version 1.2

New Features

This release adds the following features to the FlexRIO Adapter Module Development Kit:

- Added information about FlexRIO PXIe-796xR devices.
- Added registration instructions for your FlexRIO Adapter Module Development Kit.
- Updated adapter module removal protocol.
- Added information about grounding considerations in adapter module designs.
- Updated PCB, card edge connector, and adapter module enclosure dimensions, as described in Chapter 5, *Printed Circuit Board (PCB) Design Considerations*.
- Added PXI Express-based synchronization clock (IoModSyncClk) information.

Fixed Issues

This release of the *FlexRIO Adapter Module Development Kit User Manual* fixes the following issues.

Table 4. Fixed Issues with the FlexRIO Adapter Module Development Kit, Version 1.2

CAR ID	Summary
172060	Corrected bank grouping in Figure 3-2, *NI 795xR and NI 796xR Front Panel Connector Pin Assignments and Locations*.
184265	Updated FPGA I/O pin allocation tables to confirm the correct listing of clock-capable signals.

Table 4. Fixed Issues with the FlexRIO Adapter Module Development Kit, Version 1.2 (Continued)

CAR ID	Summary
186055	Specified the voltage regulation of the FlexRIO FPGA module power supply to the adapter module.
209674	Updated power supply voltage regulation values.
214282	Updated dimensional drawings to confirm correct values. Added table listing available design files and file types.
217291	Corrected the FPGA module pinout orientation.

Version 1.1

New Features

This release adds the following features to the FlexRIO Adapter Module Development Kit:

- Updated PCB, gold finger, and enclosure design files.
- Added STEP, IGES, and Pro/E design file types.

Fixed Issues

This release of the *FlexRIO Adapter Module Development Kit User Manual* fixes the following issues.

Table 5. Fixed Issues with the FlexRIO Adapter Module Development Kit, Version 1.1

CAR ID	Summary
134060	Figure 2-4, *LabVIEW FPGA, CLIP, and Hardware Integration Diagram (Virtex-5)*, depicts the DRAM as being included in the FlexRIO FPGA module design, but the FPGA module box should be reduced to show that the DRAM is external to the FlexRIO FPGA module.
134541	Corrected documented PCB thickness. Specified primary and secondary side clearances.
134541	Added information that all adapter module CLIPs should utilize BUFGCE components to gate clock inputs to LabVIEW FPGA logic. Clock inputs may be unstable during/before IO Module power-up. By connecting IO Module Enabled to the enable pin of the BUFG, you can keep the clock disabled until it is stable.

Table 5. Fixed Issues with the FlexRIO Adapter Module Development Kit, Version 1.1 (Continued)

CAR ID	Summary
136184	Added a list showing the FPGA I/O that is used for each GPIO line. For example, a UserGpio(32) should be listed noted as G26 on Bank 13.
145838	Added section about simultaneously switching outputs (SSO) noise and another section showing trace/via clearance on the backside of the PCB near the fingers to show that the edge of the enclosure can contact the PCB.

1 Before You Begin

The following sections contain information you need before developing your FlexRIO adapter module.

Development Requirements

Successful system design with the FlexRIO Adapter Module Development Kit requires knowledge in the following areas:

- Schematic circuit design
- PCB layout and fabrication
- Circuit card assembly
- VHDL code design

Note Knowledge of LabVIEW and LabVIEW FPGA is required for FlexRIO application development.

Registration

You must register your FlexRIO Adapter Module Development Kit. During the registration process, NI sends an invite to the private MDK Community Page. This page contains information about MDK updates and can be configured to send email notifications when new versions are posted. To register, visit `ni.com/ask`, and create a new technical support request.

Include the following information in the request:

- Your company name
- Contact name and email address—FlexRIO Adapter Module Development Kit updates are distributed quickly through the MDK Community Page on `ni.com`. This email address is used to invite you to the FlexRIO Adapter Module Development Kit community.
- A brief description of the adapter module project—If you require support from NI during development, this information helps the FlexRIO support team determine faster and more specific suggestions for your success.
- IO Module Vendor ID request—The FlexRIO IO Module Vendor ID is similar to a PCI vendor ID, and is a crucial element needed to configure your adapter module. Each adapter module manufacturer should use their allotted IO Module Vendor ID for all adapter modules that they produce. NI recommends using a single IO Module Vendor ID for your company, for ease of tracking and support.

If you want to use your existing PCI vendor ID as an IO Module Vendor ID, include this information as well. If you do not have a PCI vendor ID or do not want to use it as your NI FlexRIO IO Module Vendor ID, NI can assign an ID that is unique among FlexRIO vendor ID developments.

Note For more information about the IO Module Vendor ID and its use with your adapter module and your NI 795*x*R/796*x*R module, refer to Chapter 9, *Configuring Your Adapter Module for Use with NI 795xR/796xR Modules and LabVIEW FPGA*.

Note For more information about the IO Module Vendor ID and its use with your adapter module and your NI-793*x*R/NI 797*x*R module, refer to Chapter 10, *Configuring Your Adapter Module for Use with NI-793xR/797xR Modules and LabVIEW FPGA*.

Adapter Module Review

Throughout the adapter module design process, you can contact NI for an adapter module design review. NI engineers will review your FPGA-to-adapter module interface schematic to ensure that it meets the specifications required of the interface. To request an adapter module review, visit `ni.com/ask`, and create a new technical support request with the following information:

- A searchable PDF of your adapter module's schematic, preferably using the FlexRIO GPIO signal naming conventions mentioned in the MDK Manual (e.g., S142 = GPIO_0). The signal naming conventions can be found in Appendix A, *Signal Suggestions*.
- A concise description of your adapter module and the signals in your design
- Data rates/clock speeds of the FPGA-to-adapter module signals in your design
- Current draw on power supply rails
- Logic levels within the design, including V_{cco} settings
- Block diagram of the module's functionality
- A functional description of the circuitry (1 paragraph)
- Data type (signal or clock) of the FPGA-to-adapter module signals in your design

The review process takes approximately one week from the date that you contact NI.

Prior to contacting NI for an adapter module review, refer to Appendix B, *Troubleshooting*, of this document for a list of common issues.

FlexRIO Module Development Kit Installed Files

The FlexRIO Module Development Kit includes design files, example adapter module development kit development files, and documentation to assist with the design of your adapter module. The files are located in the following directories:

Design Files

Navigate to the design files by selecting **Start»All Programs»National Instruments» NI FlexRIO»NI FlexRIO Adapter Module Development Kit»Design Files**.

Documentation

The FlexRIO Adapter Module Development Kit documentation is located at **Start»All Programs»National Instruments»NI FlexRIO»NI FlexRIO Adapter Module Development Kit»Documentation**. For the locations of other FlexRIO documentation, such as device specifications and programming help, refer to the *Related Documentation* section of this manual.

Example Files

The FlexRIO example files are located at **Start»All Programs»National Instruments» NI FlexRIO»NI FlexRIO Adapter Module Development Kit»Examples**.

2

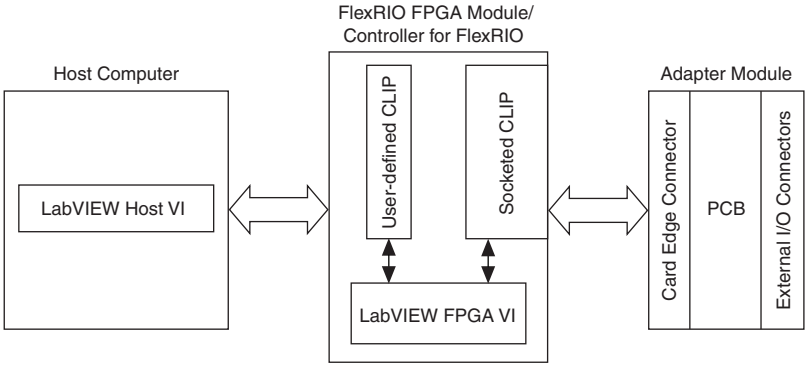
FlexRIO Solution Architecture Overview

The FlexRIO architecture allows you to fully customize your application. FlexRIO-based projects consist of the following elements:

- FlexRIO FPGA module or Controller for FlexRIO
- Adapter module PCB and module enclosure
- LabVIEW host VI
- LabVIEW FPGA VI
- Adapter module component-level intellectual property (CLIP)

Figure 2-1 illustrates the FlexRIO system architecture.

Figure 2-1. FlexRIO System Architecture Elements



The LabVIEW host VI and LabVIEW FPGA VI require you to write your own host and FPGA VIs based on the requirements of your application. The adapter module socketed CLIP and adapter module PCB are unique to adapter module development for the FlexRIO architecture. The proceeding sections describe all these elements in more detail.

Refer to the *FlexRIO FPGA Module Device Overview* (NI 795*x*R/796*x*R/797*x*R) or the *Controller for FlexRIO Device Overview* (NI-793*x*R) sections for more information about your module.

FlexRIO FPGA Module Device Overview

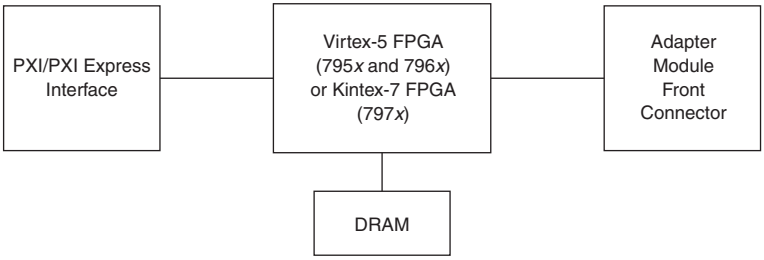
The FlexRIO FPGA module is a PXI/PXI Express device with the following features.

Table 2-1. FPGA Features

NI PXI-795*x*R/PXIe-796*x*R	NI PXIe-797*x*R
66 general-purpose I/O (GPIO) differential pairs, which can be configured as 132 single-ended signals	68 general-purpose I/O (GPIO) differential pairs, which can be configured as 136 single-ended signals
I/O data rates of up to 1 Gb/s running LVDS configuration, and 400 Mb/s single-ended	I/O data rates of up to 1 Gb/s running LVDS configuration, and 400 Mb/s single-ended
Virtex-5 FPGA • NI 7951R—Virtex-5 LX30 • NI 7952R—Virtex-5 LX50 • NI 7953R—Virtex-5 LX85 • NI 7954R—Virtex-5 LX110 • NI 7961R- Virtex-5 SX50T • NI 7962R—Virtex-5 SX50T • NI 7965R—Virtex-5 SX95T • NI 7966R—Virtex-5 SX95T	Kintex-7 FPGA • NI 7971R—Kintex-7 XC7K325T • NI 7972R—Kintex-7 XC7K325T • NI 7975R—Kintex-7 XC7K410T • NI 7976R—Kintex-7 XC7K410T
Onboard DRAM • **(NI 7952R/7953R/7954R only)** Two independently accessible 64 MB banks • **(NI 7962R/7965/7966R only)** Two independently accessible 256 MB banks	Onboard DRAM • **(7972R/7975R/7976R only)** Single 2 GB bank

Note The NI 7951R, NI 7961R, and NI 7971R devices do not have onboard DRAM.

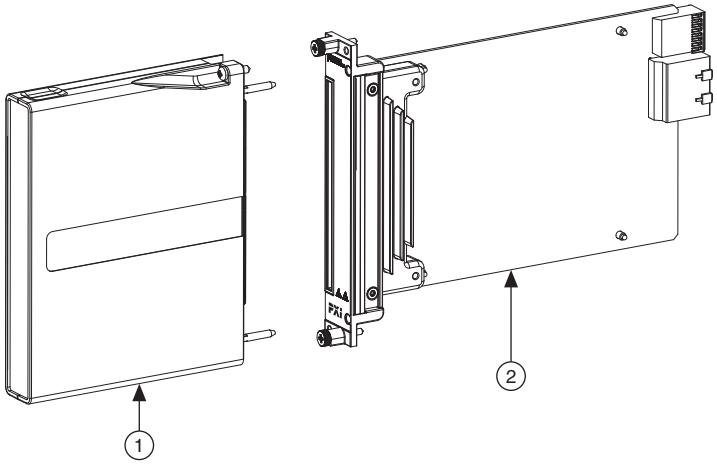
Figure 2-2. FlexRIO FPGA Module Architecture



Adapter Module Printed Circuit Board (PCB)

Mount the adapter module PCB in a standard FlexRIO enclosure and insert it into the front panel connector of your FlexRIO FPGA module. This module assembly allows you to fully customize the interface to your FlexRIO device. Chapters 2 through 7 detail the steps and guidelines necessary for developing a FlexRIO custom adapter module. Figure 2-3 depicts a FlexRIO device, which consists of the adapter module and the FlexRIO FPGA module.

Figure 2-3. FlexRIO FPGA Module and Adapter Module



1 FlexRIO Adapter Module

2 FlexRIO FPGA Module

LabVIEW Host VI and LabVIEW FPGA VI

The FlexRIO device requires both a LabVIEW host VI and a LabVIEW FPGA VI to operate. The host VI runs on the host PC, while the FPGA VI is compiled to and runs on the FPGA. For more details about the methods for communication between the host PC and LabVIEW FPGA VIs, refer to the *FPGA Module* book in the *LabVIEW Help*.

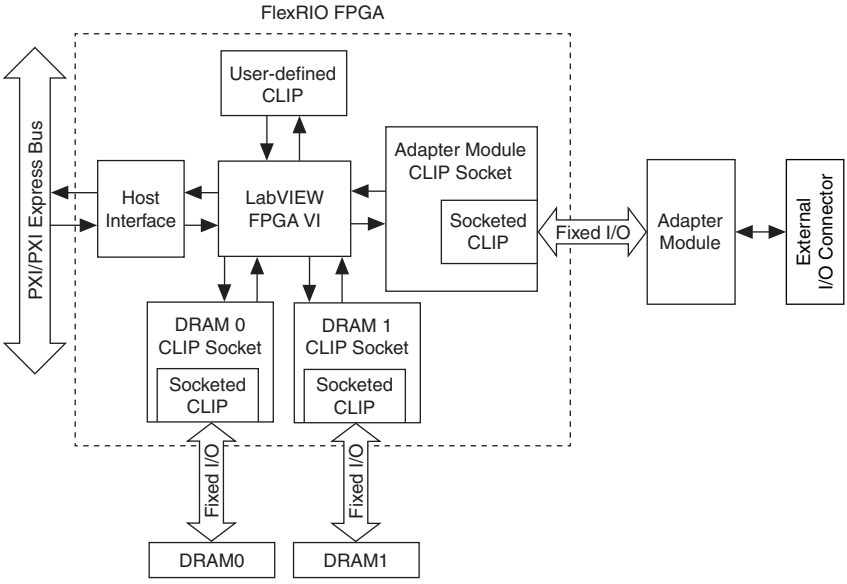
Adapter Module Component-Level IP (CLIP)

LabVIEW FPGA includes a feature for HDL IP integration called component-level IP (CLIP), which allows you to insert your own HDL IP into a LabVIEW FPGA target. FlexRIO devices support the following types of CLIP:

- *User-defined CLIP*—Allows you to insert HDL IP into an FPGA target, enabling VHDL code to communicate directly with an FPGA VI.
- *Socketed CLIP*—Provides the same IP integration functionality of the user-defined CLIP, but also allows the CLIP to communicate directly with circuitry external to the FPGA. Adapter module socketed CLIP allows your IP to communicate directly with both the FPGA VI and the external adapter module connector interface.

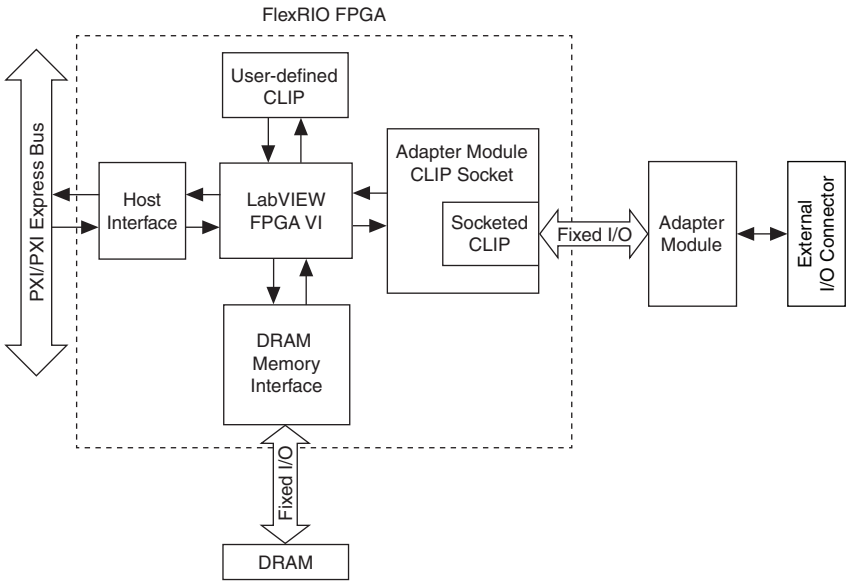
The following figure shows the relationship between an FPGA VI configured for use with an NI 795*x*R or NI 796*x*R, the two types of CLIP, and a FlexRIO adapter module.

Figure 2-4. LabVIEW FPGA, CLIP, and Hardware Integration Diagram (Virtex-5)



The following figure shows the relationship between an FPGA VI configured for use with n NI 797*x*R, the two types of CLIP, and a FlexRIO adapter module.

Figure 2-5. LabVIEW FPGA, CLIP, and Hardware Integration Diagram (Kintex-7)



For information about creating CLIP for your adapter module and NI 795*x*R/796*x*R module, refer to Chapter 11, *Creating Socketed Component-Level IP for Your Adapter Module and NI 795xR/796xR Modules*.

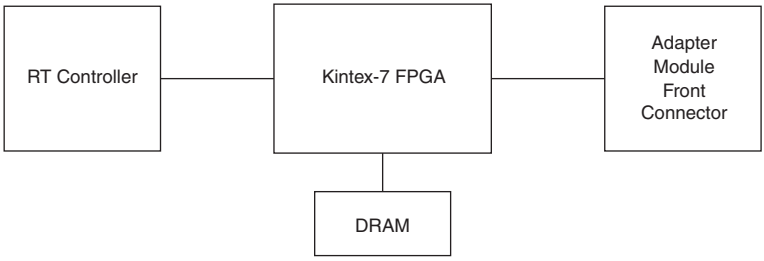
For information about creating CLIP for your adapter module and NI 797*x*R module, refer to Chapter 12, *Creating Socketed Component-Level IP for Your Adapter Module and NI-793xR/NI 797xR Modules*.

Controller for FlexRIO Device Overview

The Controllers for FlexRIO have the following features.

- 68 general-purpose I/O (GPIO) differential pairs, which can be configured as 136 single-ended signals
- I/O data rates of up to 1 Gb/s running LVDS configuration, and 400 Mb/s single-ended
- Kintex-7 FPGA
  - NI-7931R—Kintex-7 XC7K325T
  - NI-7932R—Kintex-7 XC7K325T
  - NI-7935R—Kintex-7 XC7K410T
- Single 2 GB bank onboard DRAM

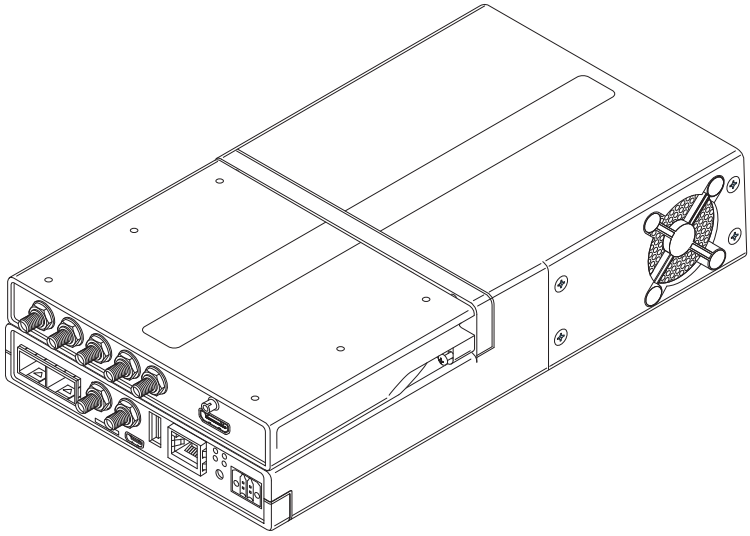
Figure 2-6. Controller for FlexRIO Architecture



Adapter Module Printed Circuit Board (PCB)

Mount the adapter module PCB in a standard FlexRIO enclosure and insert it into the front panel connector of your Controller for FlexRIO. This module assembly allows you to fully customize the interface to your FlexRIO device. Chapters 2 through 7 detail the steps and guidelines necessary for developing a FlexRIO custom adapter module. Figure 2-7 depicts an adapter module connected to a Controller for FlexRIO.

Figure 2-7. Controller for FlexRIO and Adapter Module



When designing a custom PCB, consider the following elements:

- PCB design
- Mechanical design
- Electrical design
  - Adapter module interface and protocol
  - Circuitry design used to create the interface to your application by utilizing user signals and the Controller for FlexRIO I/O

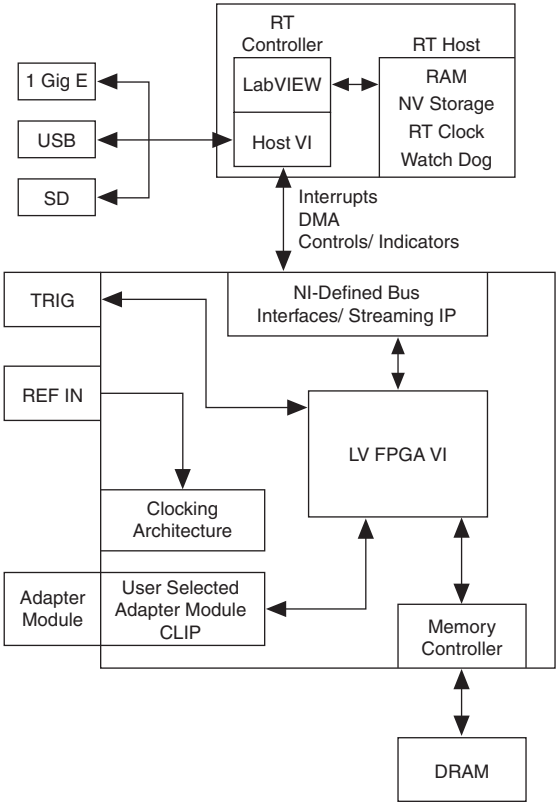
The following chapters go into greater detail about the requirements for adapter module development.

LabVIEW Host VI and LabVIEW FPGA VI

The FlexRIO device requires both a LabVIEW host VI and a LabVIEW FPGA VI to operate. The host VI runs on the host PC, while the FPGA VI is compiled to and runs on the FPGA. For more details about the methods for communication between the host PC and LabVIEW FPGA VIs, refer to the *FPGA Module* book in the *LabVIEW Help*.

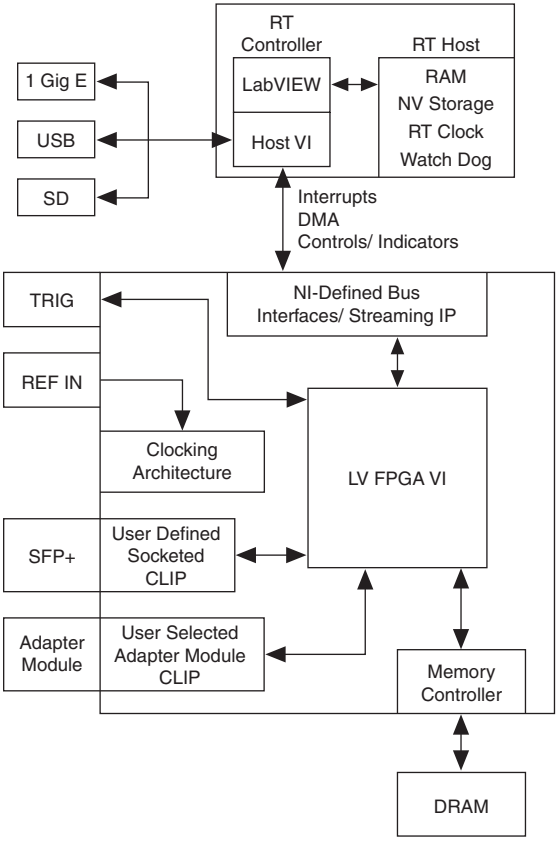
The following figure shows the relationship between an FPGA VI configured for use with an NI-7931R, user-defined CLIP, and a FlexRIO adapter module.

Figure 2-8. LabVIEW FPGA, CLIP, and Hardware Integration Diagram (NI-7931R)



The following figure shows the relationship between an FPGA VI configured for use with an NI-7932R or NI-7935R, user-defined CLIP, and a FlexRIO adapter module.

Figure 2-9. LabVIEW FPGA, CLIP, and Hardware Integration Diagram (NI-7932R and NI-7935R)



For information about creating CLIP for your adapter module and NI 793*x*R module, refer to Chapter 12, *Creating Socketed Component-Level IP for Your Adapter Module and NI-793xR/NI 797xR Modules*.

3

Interfacing Adapter Modules with NI 795xR and NI 796xR Modules

This chapter explains how to interface adapter modules with NI 795*x*R/796*x*R modules, including electrical design considerations. For information about interfacing adapter modules with NI-793*x*R/NI 797*x*R modules, refer to Chapter 4, *Interfacing Adapter Modules with NI-793xR and NI 797xR Devices*. For additional specifications information, refer to the specifications document for your FPGA module, available from the Start menu and at `ni.com/manuals`.

Caution To ensure proper and reliable operation of the adapter module, do not exceed the specifications in this chapter.

Adapter Module Electrical Interface

The FlexRIO FPGA module front panel connector provides power, clocking, and I/O connections between the FlexRIO FPGA module and the adapter module. This interface also includes dedicated pins for adapter module detection, identification, and power status. The proceeding sections describe front panel details and pin locations.

Power Guidelines

Five DC power rails are available for the adapter module connector. Table 3-1 lists these rails, their purpose, and the power available to each rail at the adapter module connector.
NI recommends that the maximum power dissipated in the adapter module does not exceed 6 W total for all components included in your design. If power for adapter module operation is provided externally, due to cooling requirements, the total dissipation within the adapter module should still remain below 6 W. Power dissipated outside of the adapter module—for example, power in external terminations—can be excluded from the 6 W maximum power number.

Caution NI recommends powering the adapter module from the power rails provided from the FlexRIO FPGA module. If you use external power rails instead or in addition to the provided rails, the adapter module must not drive the GPIO lines at a voltage different than V_{ccoA} or V_{ccoB}. The adapter module should never power any pin on the FlexRIO FPGA module connector if the PXI/PXI Express chassis is powered off.

Table 3-1. DC Power Rails

Power Rail Name	Value	Maximum Inrush Current	Maximum Steady State Current	Description
+3.3V	3.3 V +5%/-7.5%	1.3 A	1.0 A	General-purpose rail that can power 3.3 V logic or act as a source to generate other required power rails.
+12V	12 V +5%/-6.5%	350 mA*	250 mA*	General-purpose rail that can act as an input to DC-DC converters or to power analog circuitry.
V_{ccoA}/V_{ccoB}	Selectable (1.2 V, 1.5 V, 1.8 V, 2.5 V, and 3.3 V)‡	500 mA each*	250 mA each* †	I/O rails that are shared with the FPGA for digital interface circuitry.
V_{eeprom}	+3.3 V	N/A	10 mA	Powers the I^2C EEPROM on the adapter module that stores adapter module configuration information.

* The aggregate power level of V_{ccoA}, V_{ccoB}, and +12V must never exceed 4.13 W, including inrush periods. Violating this rule may cause the PXI chassis to reset.

† The FlexRIO FPGA Module can provide up to 500 mA of steady state current on V_{ccoA}/V_{ccoB} as long as you observe the power limit in the previous note. NI does not recommend exceeding 250 mA of steady state current.

‡ Although V_{ccoA}/V_{ccoB} supports 3.3 V, NI does not recommend using that voltage level for V_{ccoA}/V_{ccoB}. Future FPGA generations may not support 3.3 V power rails.

Like any electronic circuit, the adapter module power rails require bulk and bypass capacitance. The effect of the total capacitance on the inrush current is represented by the following equation.

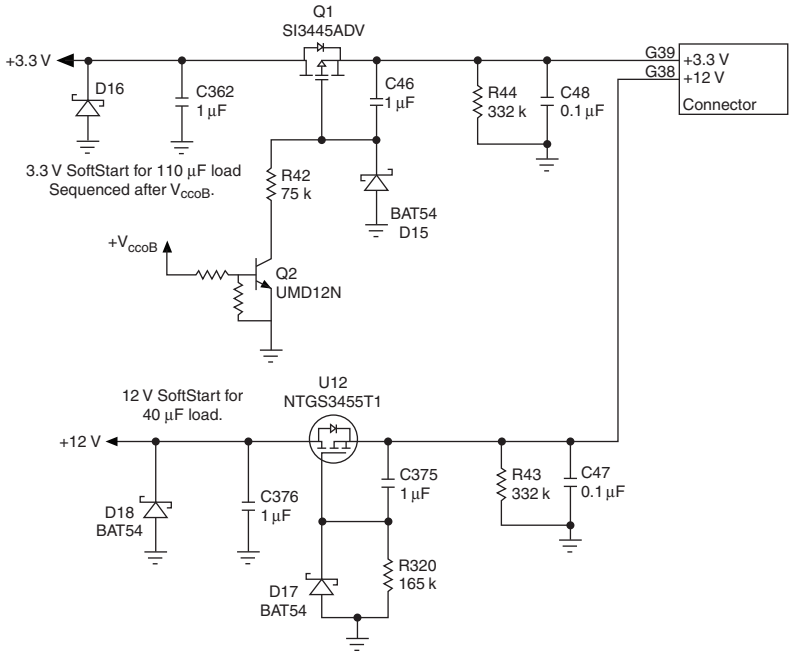
$$I = C * (\Delta V/\Delta t)$$

where C is the total capacitance on the rail charging at a rate of $\Delta V/\Delta t$.

Although some circuitry on the FlexRIO FPGA module limits the turn time (Δt) of each power rail, you should measure the actual inrush currents throughout the development of your adapter module. If any inrush current exceeds the limits listed in Table 3-1, you should make design changes, such as reducing the total capacitance or adding soft start circuits on the adapter module, to reduce the inrush current. Failure to observe these guidelines cause the device to pull excessive current from the backplane, which may cause a system reset. For more information about power rails, refer to the specifications document for your FPGA module.

Examples of adapter module soft start circuits for +3.3 V and +12 V are shown in the following figure 3-1.

Figure 3-1. Adapter Module Soft Start Circuits



The soft start circuits use positive channel field effect transistors (Q1 and U12) to allow conduction between the FlexRIO FPGA module and the adapter module. The +3.3 V circuit is also sequenced through Q2 so that it powers on after V_{ccoB} is powered. Diodes D16 and D18 add reverse voltage protection to the internal adapter module power rail. Diodes D15 and D17 reset the soft start function when power is removed at the connector.

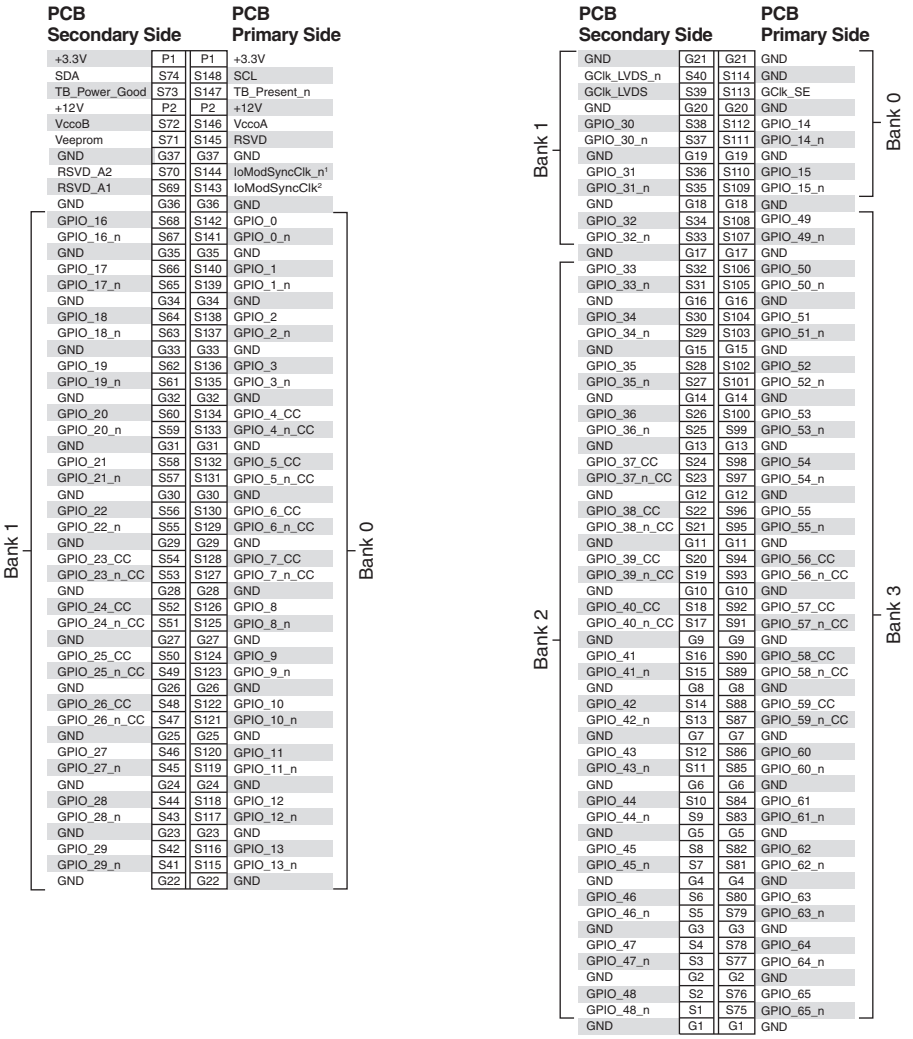
Adapter Module Connector Signals

The adapter module PCB outline includes a card edge connector. This card edge connector inserts into the front panel connector of the FlexRIO FPGA module and provides access to the available signals.

Signal Descriptions

The following figure shows the available signals on the NI 795*x*R and NI 796*x*R FPGA modules. With the exception of the two footnotes, the NI 795*x*R and NI 796*x*R modules have identical pinouts.

Figure 3-2. NI 795*x*R and NI 796*x*R Front Panel Connector Pin Assignments and Locations



1 RSVD_B2 on the NI PXI-795*x*R

2 RSVD_B1 on the NI PXI-795*x*R

Tables 3-2, 3-3, and 3-4 describe the control signals and other signals available from the NI 795*x*R and NI 796*x*R. These signals include the adapter module control signals, global clock, and power rails. These areas, along with suggested circuits and component selection, are covered in the proceeding chapters.

Table 3-2. Control Pin Assignments and Signal Descriptions

Pin	Signal	Direction	Description
S71	V_{eeprom}	To adapter module	Use the 3.3V power supply only to power the adapter module identification EEPROM.
S73	TB_Power_Good	From adapter module	3.3V TTL input to the FPGA. The timeout for this signal is 815 ms. The adapter module asserts this signal to the FlexRIO FPGA module at startup to indicate that the adapter module power rails are powered and stable. The adapter module holds this line asserted while the device is in operation.
S147	TB_Present_n	From adapter module	The adapter module pulls this signal low at all times to indicate to the FlexRIO FPGA module that the adapter module is present. Connect this line to GND.
S148	SCL	To adapter module	I^2C bus clock from the FlexRIO FPGA module bus master to the adapter module. This signal is powered by V_{eeprom}.
S74	SDA	Bidirectional	I^2C serial data line. This signal is powered by V_{eeprom}.
S146	V_{ccoA}	To adapter module	Selectable voltage rail for powering the interface circuitry between the adapter module and the FlexRIO FPGA module. Refer to the *Electrical Design Considerations* section for more information about I/O bank voltages.
S72	V_{ccoB}	To adapter module	Selectable voltage rail for powering the interface circuitry between the adapter module and the FlexRIO FPGA module. Refer to the *Electrical Design Considerations* section for more information about I/O bank voltages.

Table 3-2. Control Pin Assignments and Signal Descriptions (Continued)

Pin	Signal	Direction	Description
S114	GND	Ground	For correct operation, the adapter module design should connect all GND signals directly to a ground plane on the adapter module PCB.
S69	RSVD_A1	—	Reserved for future use. Do not connect.
S70	RSVD_A2	—	Reserved for future use. Do not connect.
S143	**(NI 795*x*R)** RSVD_B1	—	Reserved for future use. Do not connect.
	(NI 796*x*R) IoModSyncClk	—	Positive terminal for differential clock to the adapter module from either PXIe_DStarA or PXI_Clk10. For more information about clocking, refer to the *IoModSyncClk (NI 796xR Only)* section.
S144	**(NI 795*x*R)** RSVD_B1	—	Reserved for future use. Do not connect.
	(NI 796*x*R) IoModSyncClk_n	—	Negative terminal for differential clock to the adapter module from either PXIe_DStarA or PXI_Clk10. For more information about clocking, refer to the *IoModSyncClk (NI 796xR Only)* section.
S145	RSVD_CNTL	—	Reserved for future use. Do not connect.

Table 3-3. Global Clock Input Connections and Pin Assignments

Pin	Signal	Direction	Description
S39	GClk_LVDS	From adapter module	Connected directly to LVDS FPGA global clock pins (2.5 V). **Note**: LVDS requires 100 Ω parallel termination at the connection destination. Enable GClk_LVDS/GClk_LVDS_n termination in the `.ucf` constraints. For more information about constraints in the `.ucf` file, refer to the *ExampleIOModuleCLIPV5.ucf* section of Chapter 11, *Creating Socketed Component-Level IP for Your Adapter Module and NI 795xR/796xR Modules*.
S40	GClk_LVDS_n	From adapter module	
S113	GClk_SE	From adapter module	Connected directly to +3.3V LVTTL FPGA global clock pin.

Table 3-4. Power Connections Pin Assignments

Pin	Signal	Direction	Description
P1	+3.3V	To adapter module	General-purpose power rail.
P2	+12V	To adapter module	General-purpose power rail.
G<1..37>	GND	Ground	For correct operation, the adapter module design should connect all GND signals directly to a ground plane on the adapter module PCB.

General-Purpose Input/Output (GPIO)

Refer to Knowledge Base article 6NND5NOA for a list of the general-purpose I/O (GPIO) signals available from the NI 795*x*R/NI 796*x*R FlexRIO FPGA modules.

The NI 795*x*R and NI 796*x*R FPGA modules contain 66 GPIO differential pairs, which can be configured as 132 single-ended signals or a combination of each. You can use these signals to digitally interface between the FlexRIO FPGA module and the adapter module.

Each GPIO line is individually ground referenced when used as a single-ended (SE) signal. For instance, in single-ended mode, GPIO_0 and GPIO_0_n are independent signals and can be thought of as GPIO_*x* and GPIO_*y*, respectively.

When configured as a differential signal, two lines make a differential pair that comprises one signal. The pair consists of a positive side and a negative side. The negative side is designated with the *_n*. For example, GPIO_0 is the positive half of the differential signal and GPIO_0_n is the negative half.

In addition to providing digital interfacing capabilities, signal lines designated with *_CC* are clock capable, meaning that they have access to regional clock resources within that bank. For more information about clock-capable signals, refer to the *Clocks and Timing* section.

GPIO Bank Details (NI 795*x*R and NI 796*x*R)

Four FPGA banks of GPIO lines are available at the adapter module card edge connector. These four banks correspond to four physical I/O banks on the FPGA. FPGA I/O is divided into banks to allow for greater flexibility in their configuration. The four FPGA I/O banks on the FlexRIO FPGA module are connected to two separate V_{cco} supplies, which can be independently configured and can determine the compatible I/O standards available. You can also share regional clocking resources across an I/O bank. For more information about I/O banking and design implications, refer to the *Electrical Design Considerations* section.

The following table lists the association between the I/O banks of the FlexRIO FPGA module and the Xilinx FPGA banks from which they are derived. Each bank has an I/O reference voltage. There are two independent I/O voltage rails—V_{ccoA} and V_{ccoB}.

Table 3-5. Bank Reference (NI 795*x*R and NI 796*x*R)

Adapter Module Bank	V_{cco} Rail Reference	Xilinx FPGA Bank (For Reference)
0	V_{ccoA}	17
1	V_{ccoA}	13
2	V_{ccoB}	11
3	V_{ccoB}	15

Note: If you are using LabVIEW FPGA 2012 or older, V_{ccoA}/V_{ccoB} values are set in the `.tbc` file. If you are using LabVIEW FPGA 2013 or later, V_{ccoA}/V_{ccoB} values are set in the `.fam` file. For more information about configuring your `.tbc` or `.fam` file, refer to Chapter 9, *Configuring Your Adapter Module for Use with NI 795xR/796xR Modules and LabVIEW FPGA*.

The adapter module schematic also has physical references on the connector to denote bank references. For more information about signals and their banks, refer to the *Signal Descriptions* section.

Adapter Module Interface Protocol

FlexRIO FPGA modules interface to a wide variety of adapter modules. These modules may have different V_{cco} or V_{ccoA}/V_{ccoB} power requirements, as well as a variety of GPIO I/O standards and signal directions. FlexRIO devices also allow users to change I/O characteristics with each different adapter module. NI recommends that each adapter module design contain an EEPROM for identification and protection.

The identification EEPROM is connected to dedicated pins and designated at a specific I^2C address, and allows the FlexRIO driver to identify an inserted adapter module. Including an EEPROM in your design allows for improved electrical protection for the adapter module and the FlexRIO FPGA module. Using an identification EEPROM prevents the use of incompatible adapter module settings such as incorrect power rails and double-driving of digital interface pins. Refer to the *EEPROM Overview* section for additional information about adding an ID EEPROM to your adapter module design.

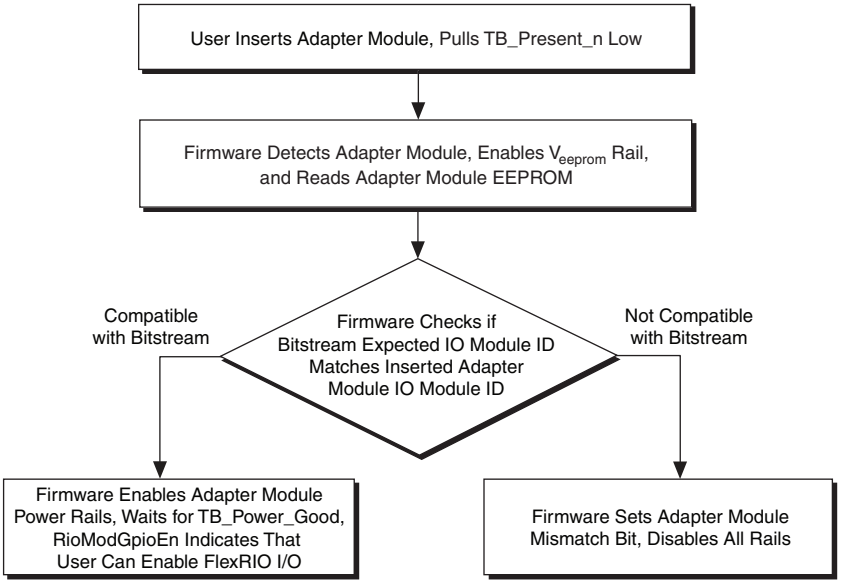
The FlexRIO device employs the following adapter module identification, power up, and removal processes to prevent applying improper voltages or double-driving signals.

Adapter Module Insertion Protocol

Figure 3-3 depicts the process that the FlexRIO system performs when you insert a new adapter module into the FlexRIO FPGA module.

Note The system also performs the following steps when you download a new LabVIEW FPGA bitstream to the FlexRIO device or when you assert a Reset command.

Figure 3-3. Adapter Module Insertion Protocol



Note The steps for detecting an adapter module are performed during device configuration. For more information about adapter module configuration, refer to Chapter 9, *Configuring Your Adapter Module for Use with NI 795xR/796xR Modules and LabVIEW FPGA*.

Note You can confirm proper adapter module connection or adapter module mismatch in the IO Module Properties Status dialog in LabVIEW. In your LabVIEW **Project Explorer** window, right-click the **IO Module** item under the FPGA Target and select **Properties** to display the **IO Module Properties** dialog box. Click the **Status** category to view the adapter module **Status** dialog.

Adapter Module Removal Protocol

To properly remove an adapter module from the FlexRIO FPGA module, you must disable the adapter module within the LabVIEW FPGA user interface. To disable the adapter module within LabVIEW, complete the following steps:

1. In your LabVIEW **Project Explorer** window, right-click the **IO Module** item under the FPGA Target and select **Properties** to display the **IO Module Properties** dialog box.
2. Click the **Status** category to view the adapter module **Status** dialog.
3. Deselect the checkbox for **Enable IO Module Power**. When this option is deselected, firmware tristates the FlexRIO I/O and disables all adapter module power rails.
4. Click **OK**.

FlexRIO driver software allows you to enable and disable adapter modules, as well as control EEPROM access, within the LabVIEW interface. Refer to Chapter 9, *Configuring Your Adapter Module for Use with NI 795xR/796xR Modules and LabVIEW FPGA*, for information about programming your adapter module EEPROM.

EEPROM Overview

NI strongly recommends adding an EEPROM to your adapter module for identification purposes. Adding EEPROM provides better electrical protection for both the adapter module and the FlexRIO FPGA module. It also improves the software configuration experience within LabVIEW FPGA.

The adapter module identification EEPROM included in your design must be an I^2C-capable 2 Kbit device (256 × 8). These parts are widely available from electronics manufacturers and are often identified as 24C02. Manufacturers typically place a prefix to identify their version, for example, ST Microelectronics EEPROM part number is M24C02.

Figure 3-4 depicts a complete FlexRIO EEPROM connection circuit. This circuit shows the connections to the EEPROM (U_1) and two additional connections required for proper adapter module configuration. The supporting parts required for the EEPROM are a bypass capacitor (C_1) on the line from V_{eeprom}, a pull-up resistor (R_1) on address line A1 to V_{eeprom}, and a pull-down resistor (R_2) on the WP line. C_1 is a 0.1 µF ceramic capacitor with a voltage rating of 6.3 V or higher. For best performance, use a capacitor with an X7R dielectric. Use 4.7 kΩ resistors, which can be any resistor type rated for at least 25 V and 10% tolerance or better. Other details regarding this interface can be found in the EEPROM datasheet. For more information about EEPROM use and programming in your application, refer to Chapter 9, *Configuring Your Adapter Module for Use with NI 795xR/796xR Modules and LabVIEW FPGA*.

Note The I²C address for the EEPROM is 0x52. Use 0xA4 for write operations and 0xA5 for read operations. Use the following bit fields to ensure the correct I²C address.

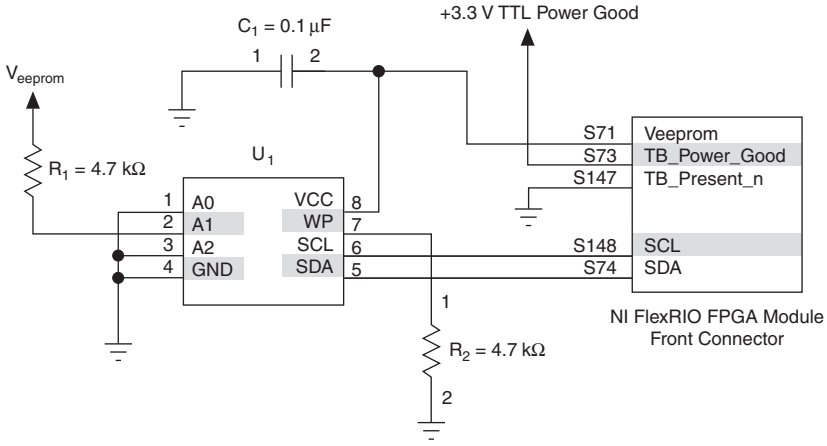
1	0	1	0	0	1	0	R/$\overline{W}$
EEPROM Device Type Identifier				Chip Enable			Operation
				A2	A1	A0	1 = R 0 = W

EEPROM Recommendations

Use an ST Microelectronics serial I²C bus EEPROM (part number M24C02-WMN) or equivalent for the FlexRIO adapter module. Refer to `www.st.com` for datasheets and additional information.

EEPROM Schematic and Wiring

Figure 3-4. EEPROM Wiring



Note Multiple components on the adapter module can share the I²C bus. Depending on the adapter module design, some of these components that share the I²C bus with the EEPROM may not power on when the initial adapter module insertion protocol executes. If you use components that pull down the I²C lines to a low level when powered off, NI recommends using a switch that separates the I²C bus from the EEPROM I²C lines.

Electrical Design Considerations

This section provides electrical considerations, clocking and timing information, and additional information about the design of your adapter module.

Power up and Sequencing with External Hardware

When your FlexRIO device is connected to external equipment, you should consider the powering sequence of the system. When you insert a new adapter module, download a new bitstream to the LabVIEW FPGA target, or assert the Reset command, the adapter module power is disabled until the FPGA verifies device compatibility. Refer to the *Adapter Module Insertion Protocol* section for more information about adapter module insertion protocol.

If your external circuitry drives signals into the FlexRIO device, NI recommends disabling this circuitry when the adapter module is not powered. You can monitor adapter module power status in the LabVIEW FPGA VI and on the host VI.

FPGA I/O and Protection

Use caution when connecting adapter module circuitry to the FPGA. Refer to Xilinx documentation, available at `www.xilinx.com`, for specific FPGA I/O voltage limits. To avoid adapter module circuit damage, tristate all FPGA outputs using the adapter module socketed CLIP until the adapter module power is enabled. Refer to Chapter 11, *Creating Socketed Component-Level IP for Your Adapter Module and NI 795xR/796xR Modules*, for more information about adapter module socketed CLIP. Refer to Appendix C, *Xilinx Documentation References*, for links to applicable Xilinx documentation about your FPGA.

NI recommends that you *never* directly expose signals from the front panel connector of the FlexRIO FPGA module to the external connectivity of the adapter module. Some form of buffering is required when routing signals from the FlexRIO FPGA module to the circuits external to the adapter module. Most of these parts are high speed and offer electrostatic discharge (ESD) protection as well as voltage tolerance protection. It is not necessary to consider this buffer requirement when using the FlexRIO FPGA module in applications such as interfacing to an ADC, as the digital signals are not directly exposed.

Grounding

FlexRIO adapter module designs may include several ground planes. The interface to the FlexRIO FPGA module provides the following two ground connections:

- FlexRIO digital ground—The FlexRIO FPGA module front connector includes several pins routed to a ground plane (GND), as shown in the *Signal Descriptions* section. All FlexRIO GPIO, control, and power pins are referenced to this ground plane. To maintain proper trace impedance, reference all FlexRIO signals to this ground plane on the adapter module.
- PXI/PXI Express chassis ground—The required FlexRIO adapter module enclosure connects to the FlexRIO FPGA module front panel, and thereby to PXI/PXI Express chassis ground by way of the guide pins and mounting screws. This chassis ground is

available on the adapter module PCB by way of the four enclosure mounting screws. NI recommends including adapter module designs with provisions for a strong connection between any cable or connector shields and this chassis ground. You can make this connection through the adapter module PCB and/or the adapter module front panel. This continuous shield connection is especially important for adapter modules that must pass emissions testing.

NI 795*x*R/796*x*R FPGA I/O Bank Voltages

The FlexRIO FPGA I/O are divided into four I/O banks. Each bank is powered by one of two V_{cco} power supply rails, either V_{ccoA} or V_{ccoB}. When determining how to divide your I/O among these banks, you must consider the V_{cco} voltage and I/O standard requirements of your application. The V_{ccoA}/V_{ccoB} setting determines which I/O standards are available in the I/O bank.

For example, if V_{ccoA} is set to 3.3 V, then you may configure the GPIO in banks <0..1> to use the LVTTL, LVCMOS33, or LVDCI33 I/O standards. If V_{ccoB} is set to 2.5 V, then you may configure the GPIO in banks <2..3> to use the LVCMOS25, LVDCI25, or LVDS_25 I/O standards. You may choose different I/O standards within a single I/O bank if all I/O standards are compatible with the selected V_{ccoA}/V_{ccoB} voltage setting.

FlexRIO V_{ccoA}/V_{ccoB} voltage settings and GPIO standards are set in the adapter module configuration (`.tbc` or `.fam`) file during device configuration. The section *FlexRIO Supported Xilinx I/O Standards* in Chapter 9,*Configuring Your Adapter Module for Use with NI 795xR/796xR Modules and LabVIEW FPGA*, contains a list of the most common I/O standards and their required V_{cco} voltages. For more information about setting I/O standards for your FlexRIO device, refer to Chapter 9, *Configuring Your Adapter Module for Use with NI 795xR/796xR Modules and LabVIEW FPGA*. For I/O standard specifications, refer to Xilinx documentation, available at `www.xilinx.com`. Refer to Appendix C, *Xilinx Documentation References*, for a list of Xilinx documents applicable to FlexRIO applications.

Simultaneous Switching Output (SSO) Noise

When multiple FPGA output drivers change state at the same time, the changing current causes a power supply disturbance. These disturbances can cause undesired behavior in output drivers, input receivers, or in internal logic, which is referred to as *simultaneous switching output (SSO) noise*. Published SSO limits determine the number and type of I/O output drivers that can change state simultaneously without introducing excessive levels of SSO noise to your application.

You should consider SSO limitations when using many GPIO as outputs. NI recommends spreading outputs across all of the GPIO banks to reduce the quantity of SSO in each FPGA bank. FlexRIO FPGA modules meet the nominal PCB requirements documented for the Xilinx SSO limits. To estimate the effect of SSO noise in your application, Xilinx provides the *Virtex-5 FPGA SSO Calculator*, which contains all SSO limit data for all I/O standards. Refer to Xilinx documentation, available at `www.xilinx.com`, for additional information about SSO noise and to access the *Virtex-5 FPGA SSO Calculator*. Refer to Appendix C, *Xilinx Documentation References*, for links to applicable Xilinx documentation.

Clocks and Timing

The NI 795*x*R and NI 796*x*R module front panel connectors expose two FPGA global clock inputs, as well as 16 pairs of clock-capable I/O lines. These inputs provide an external clock to the FlexRIO FPGA for use in synchronous interfaces.

It is important to consider timing requirements for any synchronous interface to the FlexRIO FPGA. I/O timing is defined using the socketed CLIP VHDL and user Xilinx User Constraint File (UCF), which are two elements that comprise the socketed CLIP interface.

Note Refer to the *ExampleIOModuleCLIPV5.ucf* section of Chapter 11, *Creating Socketed Component-Level IP for Your Adapter Module and NI 795xR/796xR Modules*, for more information about using the `.ucf` file.

Use the adapter module CLIP `.ucf` file to provide necessary period and offset constraints, or to provide setup/hold requirements to the Xilinx compiler. You can also add digital clock managers (DCMs) or phase-locked loops (PLLs) in your CLIP for phase shifting or frequency multiplication. For example, an adapter module could include an ADC which drives 16-bit samples along with a synchronous clock to the FPGA. The ADC datasheet specifies the relationship between the clock and data output of the part. In this case, the CLIP UCF should contain a period constraint for the external clock, as well as an offset constraint for the incoming data relative to the external clock.

For information about adding timing constraints to your CLIP, refer to the *ExampleIOModuleCLIPV5.ucf* section of Chapter 11 10, *Creating Socketed Component-Level IP for Your Adapter Module and NI 795xR/796xR Modules*. For more information about timing constraints, FPGA clock input frequency limitations, and DCM/PLL details, refer to Xilinx documentation, available at `www.xilinx.com`. Refer to the *Xilinx Documentation References* for a list of Xilinx documents applicable to FlexRIO applications.

Clock-Capable I/O Signals

NI recommends using the two FPGA global clock inputs whenever possible, as these clocks have the fewest restrictions. If your application requires additional clocking signals, each NI 795*x*R/796*x*R FPGA I/O target exposes four pairs of regional clock-capable (_CC) lines. You can configure these clock-capable lines as differential pairs or as single-ended lines. If you configure the lines as single-ended lines, you must route the clocking signal to the positive side. The negative side can be used as a GPIO line.

Refer to Appendix A, *Signal Suggestions*, for your device's pinout chart.

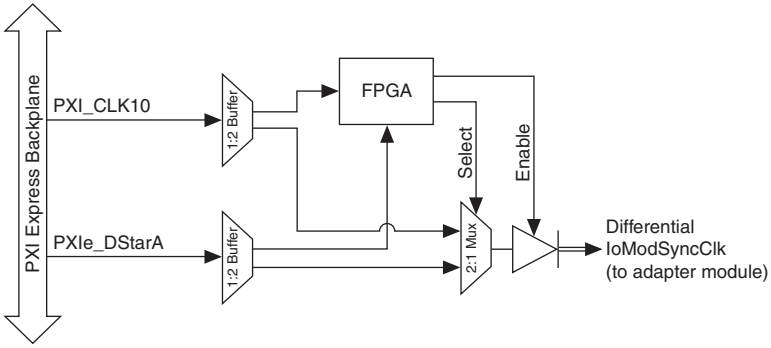
As GPIO signals, these lines can be either inputs or outputs. You can also use these signals to pass a clock into the FPGA. These clock-capable signals have access to regional clocking resources in the FPGA and may be useful for high-speed synchronous interfaces where the I/O is contained in a single clock region. Refer to Xilinx documentation, available at `www.xilinx.com`, for more information about clock-capable signals and regional clocking.

Refer to the *Xilinx Documentation References* for a list of Xilinx documents applicable to FlexRIO applications.

IoModSyncClk (NI 796*x*R Only)

The FlexRIO PXI Express FPGA module provides the IoModSyncClk signal for synchronization between adapter modules. IoModSyncClk is a low-voltage, positive-emitter-coupled logic (LVPECL) clock that can either be sourced by PXI_CLK10 or by PXIe_DStarA, as shown in Figure 3-5 3-5.

Figure 3-5. IoModSyncClk Source Block Diagram

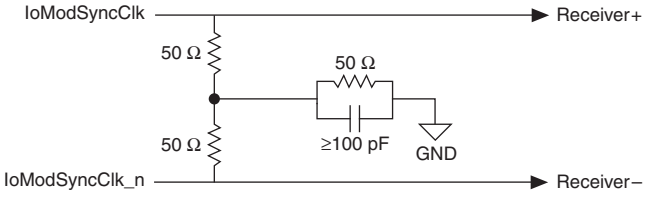


For more information about the IoModSyncClk and using it in your adapter module design, refer to the *IoModSyncClk.fam Values (NI 796xR Only)* section of Chapter 9, *Configuring Your Adapter Module for Use with NI 795xR/796xR Modules and LabVIEW FPGA*. Refer to the *FlexRIO FPGA Module Base Clock Properties* topic in the *FlexRIO Help* for more information about clock resources.

Termination

If an adapter module uses IoModSyncClk, terminate both lines with a 50 Ω load to 1.3 V (or Thevenin equivalent) as close to the receiver as possible. Figure 3-6 shows the recommended termination for IoModSyncClk.

Figure 3-6. IoModSyncClk Termination



GPIO Termination and Impedance

All GPIO traces from the FlexRIO FPGA to the front panel connector are routed as 100 Ω differential/50 Ω single-ended traces. As listed in the previous section, you can configure FPGA I/O to use Xilinx digitally controlled impedance (DCI) technology. DCI technology allows the driver output impedance to match the trace impedance. For best performance, NI recommends designing the trace impedance in the adapter module PCB to match the driver output impedances.

Note DCI technology is available on the NI 795*x*R and NI 796*x*R devices only.

For example, if your adapter module uses GPIO as single-ended outputs to IC inputs, route these signals as 50 Ω single-ended traces. If your adapter module drives an LVDS signal into the FlexRIO FPGA, route this signal as a 100 Ω differential signal. In the case of FPGA LVDS inputs, 100 Ω parallel termination at the receiver may be enabled using the socketed CLIP UCF file. For more information, refer to the *ExampleIOModuleCLIPV5.ucf* section of Chapter 11, *Creating Socketed Component-Level IP for Your Adapter Module and NI 795xR/796xR Modules*.

Minimizing Crosstalk

All FlexRIO GPIO signals are routed to maximize flexibility, allowing them to be used as 132 single-ended signals or 66 differential pairs. GPIO*x* and GPIO*x*_n are differentially coupled, and can induce crosstalk on each other when used as independent, single-ended signals.

For maximum performance in these single-ended applications, NI recommends using only one side of a differential pair and connecting the opposite side of the pair to ground. Use this method only for the highest performance applications as doing so reduces the number of available GPIO signals.

Sharing the I²C Bus

The FlexRIO FPGA module front panel connector includes a dedicated 3.3V I²C bus for the ID EEPROM. This bus is powered from the V_{eeprom} rail and accessible by way of the FlexRIO driver and the adapter module socketed CLIP. You can attach other I²C devices to this bus only if the I²C bus address is unique.

Note The power on protocol for the FlexRIO FPGA module uses the I²C bus in a manner that may affect your application. Specifically, during adapter module identification, only the V_{eeprom} rail is powered, which means any user circuitry is unpowered. The I²C bus interface signals (SCL and SDA), however, are toggling. If another I²C device *not* powered from V_{eeprom} shares the I²C bus to the ID EEPROM, the SCL and SDA signals may be pulled down on the unpowered device, which prevents the identification and powering of your FlexRIO FPGA module. Consult your I²C datasheet to determine if additional circuitry is necessary to ensure proper adapter module identification while sharing the I²C bus among multiple devices.

Choosing Circuitry Components

When designing the interface circuity between the adapter module and the FlexRIO FPGA module, NI recommends choosing 2.5V or lower components whenever possible. As FPGA process geometrics decrease, the supported interface levels also decrease. Using 2.5V or lower components makes present adapter module designs more likely to remain compatible with future FPGA families.

Unused Pin Recommendations

The following table lists the NI recommendations for the unassigned/unused pins on the FlexRIO FPGA module.

Table 3-6. Unassigned Pin Recommendations

Pin	Recommended Use
GPIO	Leave unconnected or tie to ground using a 1 kΩ or greater pull-down resistor to minimize single-ended I/O crosstalk.
GClk Inputs	Leave unconnected.
V_{ccoA}/V_{ccoB}	Leave unconnected.
+12V, +3.3V	Leave unconnected.
IoModSyncClk	If an adapter module does not use IoModSyncClk, leave the lines unconnected and unterminated.

4

Interfacing Adapter Modules with NI-793xR and NI 797xR Devices

This chapter explains how to interface adapter modules with NI-793*x*R and NI 797*x*R modules, including electrical design considerations. For information about interfacing adapter modules with NI 795*x*R and NI 796*x*R modules, refer to Chapter 3, *Interfacing Adapter Modules with NI 795xR and NI 796xR Modules*. For additional specifications information, refer to the specifications document for your NI-793*x*R or NI 797*x*R, available from the Start menu and at `ni.com/manuals`.

Caution To ensure proper and reliable operation of the adapter module, do not exceed the specifications in this chapter.

Adapter Module Electrical Interface

The NI-793*x*R and NI 797*x*R module front panel connectors provide power, clocking, and I/O connections between the NI-793*x*R/NI 797*x*R and the adapter module. This interface also includes dedicated pins for adapter module detection, identification, and power status. The proceeding sections describe front panel details and pin locations.

Power Guidelines

Four DC power rails are available for the adapter module connector. Table 4-1 lists these rails, their purpose, and the power available to each rail at the adapter module connector. You can adjust the sequence in which the rails come online and the delay between each rail from 0 ms to 500 ms. NI recommends that the maximum power dissipated in the adapter module does not exceed 6 W total for all components included in your design. If you provide external power for adapter module operation, due to cooling requirements, the total dissipation within the adapter module should still remain below 6 W. Power dissipated outside of the adapter module—for example, power in external terminations—can be excluded from the 6 W maximum power number.

Caution NI recommends powering the adapter module from the power rails provided from the NI-793*x*R/NI 797*x*R. If you use external power rails instead or in addition to the provided rails, the adapter module must not drive the GPIO lines at a voltage different than V_{cco}. The adapter module should never power any pin on the NI-793*x*R/NI 797*x*R module connector if the NI-793*x*R/NI 797*x*R is powered off.

Table 4-1. DC Power Rails

Power Rail Name	DC Value (Volts)	Maximum Current (Amps)	Trip Current (Amps)	Control Time (msec)	Description
+3.3V	3.3 +5%/-5%	2	2.25-0/+10%	13.5 ± 30%	General-purpose power rails.
+12V	12 +5%/-5.6%	0.5	0.56-0/+10%	6.3 ± 20%	
V_{cco}	Selectable (1.2, 1.5, 1.8, 2.5, and 3.3)*†	1	2.25-0/+10%	13.5 ± 30%	I/O rail that is shared with the FPGA for digital interface circuitry.
V_{eeprom}	+3.3	.05	0.15 ±35%	12.5 ± 60%	Powers the I^2C EEPROM on the adapter module that stores adapter module configuration information.

* Although V_{cco} supports 3.3V, NI does not recommend using that voltage level for V_{cco}. Future FPGA generations may not support 3.3V power rails.

† To keep the power rails backwards compatible with the NI 795*x*R and NI 796*x*R modules, you must keep the previous V_{ccoA} and V_{ccoB} pins separate (not wired to the same net) and configure a maximum of 0.5 A of pull on each of those pins.

The NI-793*x*R and NI 797*x*R power rails are significantly different than the NI 795*x*R and NI 796*x*R power rails. The NI-793*x*R and NI 797*x*R modules feature hot swap controllers on all rails. These hot swap controllers actively control power-on inrush currents and provide hard over-current shutdown points (trip points). These controllers remove the need to design soft start circuits on the adapter module, as this function is now built into the NI-793*x*R and NI 797*x*R.

Note Soft start circuits are still required on the adapter module for backwards compatibility with the NI 795*x*R and NI 796*x*R devices.

The V_{eeprom} rail uses the TPS22945 controller. The other rails use the TPS24700 controller. These controllers actively limit the inrush current for the control time shown in Table 4-1. The current during the inrush period is about 10% higher than the trip current limit. If the inrush condition continues for longer than the control time, the controller disconnects the rail and the FPGA module turns off all the adapter module power rails. To reset the inrush condition, either remove and re-insert the adapter module or cycle **Control IO Module Power** from FALSE to TRUE. During development testing, NI recommends measuring the inrush time and ensuring that the design is below the minimum time listed in Table 4-1. Generally, these inrush conditions are not a problem, since even loads of 200 µF meet these conditions.

After the inrush period is over, each controller monitors its current and disconnects if the current exceeds the trip current for the control time. If any rail disconnects, the FPGA module turns off all the adapter module power rails. If the adapter power rails turn off, either remove and re-insert the adapter module or cycle **Control IO Module Power** from FALSE to TRUE. NI recommends testing the adapter modules for steady state and transient loads to be at or below the values listed in the Maximum Current (Amps) column of Table 4-1 to prevent unwanted shutdowns due to excessive current draw.

NI recommends that you set the V_{cco} current to 1 A maximum since this rail is also connected to the FPGA and is not intended for large transient currents. The V_{cco} voltage is designed to be within a few mV of the target voltage at 25 °C for a 1 A load.

While the total power of the adapter module is still limited to 6 W, this power may be pulled from any combination of rails without any other conditions. The total power being used by the adapter module is monitored by the FPGA module.

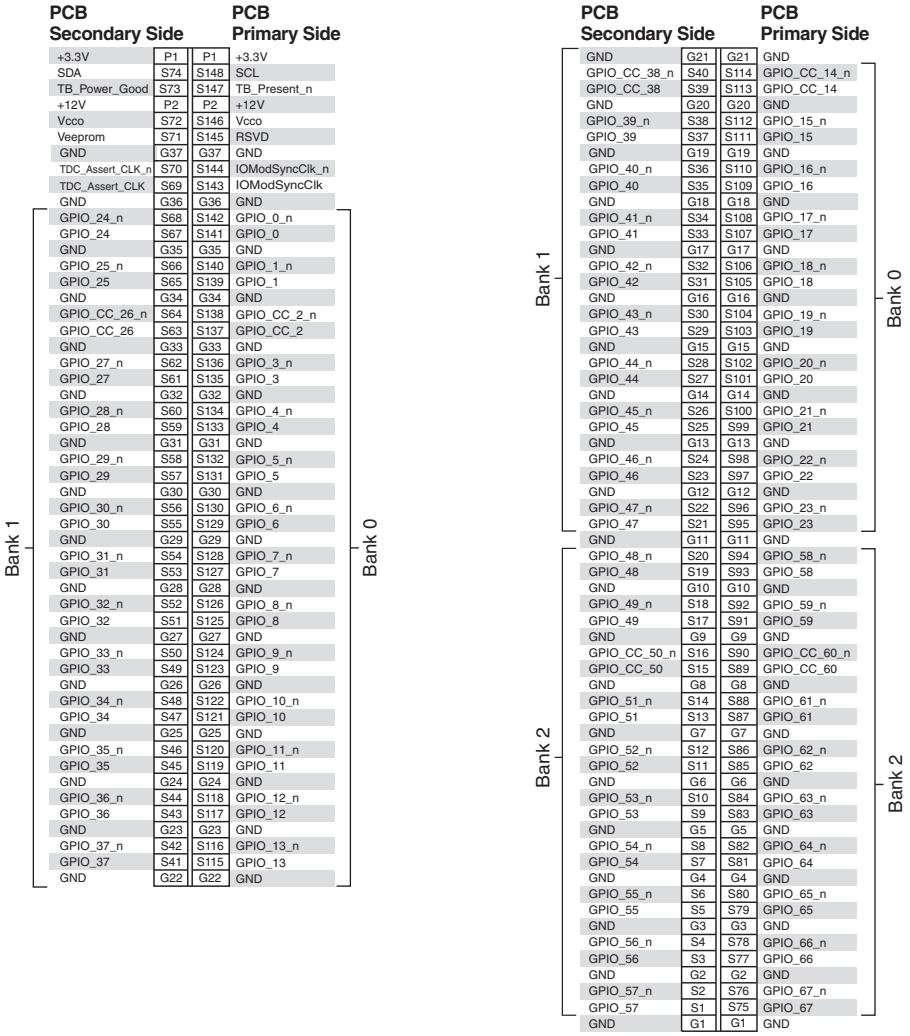
Adapter Module Connector Signals

The adapter module PCB outline includes a card edge connector. This card edge connector inserts into the front panel connectors of the NI-793*x*R/NI 797*x*R modules and provides access to the available signals.

NI-793*x*R and NI 797*x*R Signal Descriptions

The following figure shows the available signals on the NI-793*x*R and NI 797*x*R modules.

Figure 4-1. NI-793*x*R and NI 797*x*R Front Panel Connector Pin Assignments and Locations



Caution Connections that exceed any of the maximum ratings of input or output signals on the FlexRIO device can damage the NI-793*x*R/NI 797*x*R, the adapter module, and the computer or chassis. NI is *not* liable for any damage resulting from such signal connections. For the maximum input and output ratings for each signal,

refer to the specifications document for your device, available from the Start menu and at `ni.com/manuals`.

Tables 4-2 and 4-3 describe the control signals and other signals available from the NI-793*x*R and NI 797*x*R modules. These signals include the adapter module control signals and power rails. These areas, along with suggested circuits and component selection, are covered in the proceeding chapters.

Note To enable an adapter module connector on NI-793*x*R and NI 797*x*R devices to be compatible with NI 795*x*R and NI 796*x*R devices, several general-purpose I/O lines with multi-region clock capabilities on NI-793*x*R and NI 797*x*R devices are provided at locations that match the fixed global clock input signals on NI 795*x*R and NI 796*x*R devices. The dedicated global clock signals on NI 795*x*R and NI 796*x*R devices are provided at a fixed voltage level; the multi-region clock capable lines on the NI-793*x*R and NI 797*x*R devices must use a logic level that matches the remaining general-purpose I/O lines.

Table 4-2. Control Pin Assignments and Signal Descriptions

Pin	Signal	Direction	Description
S71	V_{eeprom}	To adapter module	Use this 3.3V power supply only to power the adapter module identification EEPROM.
S73	TB_Power_Good	From adapter module	3.3V TTL input to the FPGA. The timeout for this signal is 815 ms. The adapter module asserts this signal to the NI-793*x*R/NI 797*x*R at startup to indicate that the adapter module power rails are powered and stable. The adapter module holds this line asserted while the device is in operation.
S147	TB_Present_n	From adapter module	The adapter module pulls this signal low at all times to indicate to the NI-793*x*R/NI 797*x*R that the adapter module is present. Connect this line to GND.
S148	SCL	To adapter module	I^2C bus clock from the NI-793*x*R/NI 797*x*R module bus master to the adapter module. This signal is powered by V_{eeprom}.
S74	SDA	Bidirectional	I^2C serial data line. This signal is powered by V_{eeprom}.

Table 4-2. Control Pin Assignments and Signal Descriptions (Continued)

Pin	Signal	Direction	Description
S146 and S72	V_{cco}	To adapter module	Selectable voltage rail for powering the interface circuitry between the adapter module and the NI-793*x*R/NI 797*x*R. Refer to the *NI-793xR and NI 797xR FPGA I/O Bank Voltages* section for more information about I/O bank voltages. To make this signal backwards compatible with the NI 795*x*R and NI 796*x*R modules, you must not connect pins S146 and S72 together. Treat these pins like separate V_{ccoA} and V_{ccoB} rails operating at the same V_{cco} voltage.
S69	TDC_Assert_CLK	—	Reserved for future use. Do not connect.
S70	TDC_Assert_CLK_n	—	Reserved for future use. Do not connect.
S143	IoModSyncClk*	—	Positive terminal for differential clock to the adapter module from either PXIe_DStarA or PXI_Clk10. For more information about clocking, refer to the *IoModSyncClk* section.
S144	IoModSyncClk_n†	—	Negative terminal for differential clock to the adapter module from either PXIe_DStarA or PXI_Clk10. For more information about clocking, refer to the *IoModSyncClk* section.
S145	RSVD_CNTL	—	Reserved for future use. Do not connect.

* Leave IoModSyncClk unconnected on NI-793*x*R modules.

† Leave IoModSyncClk_n unconnected on NI-793*x*R modules.

Table 4-3. Power Connections Pin Assignments

Pin	Signal	Direction	Description
P1	+3.3V	To adapter module	General-purpose power rail.
P2	+12V	To adapter module	General-purpose power rail.
G<1..37>	GND	Ground	For correct operation, the adapter module design should connect all GND signals directly to a ground plane on the adapter module PCB.

General-Purpose Input/Output (GPIO)

Refer to Knowledge Base article 6NND5NOA for a list of general-purpose I/O (GPIO) signals available from the NI-793*x*R and NI 797*x*R modules.

The NI-793*x*R and NI 797*x*R modules contain 68 GPIO differential pairs, which can be configured as 136 single-ended signals or a combination of each. You can use these signals to digitally interface between the adapter module and the NI-793*x*R/NI 797*x*R.

Each GPIO line is individually ground referenced when used as a single-ended (SE) signal. For instance, in single-ended mode, GPIO_0 and GPIO_n_0 are independent signals and can be thought of as GPIO_*x* and GPIO_*y*, respectively.

When configured as a differential signal, two lines make a differential pair that comprises one signal. The pair consists of a positive side and a negative side. The negative side is designated with the _*n*. For example, GPIO_0 is the positive half of the differential signal and GPIO_n_0 is the negative half.

In addition to providing digital interfacing capabilities, signal lines designated with _*CC* are clock-capable, meaning that they have access to clock resources. For more information about clock-capable signals, refer to the *Clocks and Timing* section.

GPIO Bank *x* Details

Three FPGA banks of GPIO lines are available at the adapter module card edge connector. These three banks correspond to three physical I/O banks on the FPGA. The three FPGA I/O banks on the NI-793*x*R/NI 797*x*R are connected to the V_{cco} supply for the desired I/O standard. You can also share regional clocking resources across an I/O bank. For more information about I/O banking and design implications, refer to the *Electrical Design Considerations* section.

The following table lists the association between the I/O banks of the NI-793*x*R/NI 797*x*R and the Xilinx FPGA banks from which they are derived.

Table 4-4. Bank Reference

Adapter Module Bank	V_{cco} Rail Reference	Xilinx FPGA Bank (For Reference)
0	V_{cco}	16
1	V_{cco}	17
2	V_{cco}	18
Note: The V_{cco} value is set in the adapter module configuration file (`.fam`) file during device configuration. For more information about configuring your `.fam` file, refer to Chapter 10, *Configuring Your Adapter Module for Use with NI-793xR/797xR Modules and LabVIEW FPGA*.		

The adapter module schematic also has physical references on the connector to denote bank references. For more information about signals and their banks, refer to Appendix A, *Signal Suggestions*.

Adapter Module Interface Protocol

NI-793*x*R/NI 797*x*R modules interface to a wide variety of adapter modules. These modules may have different V_{cco} power requirements, as well as a variety of GPIO I/O standards and signal directions. FlexRIO devices also allow users to change I/O characteristics with each different adapter module. NI recommends that each adapter module design contain an EEPROM for identification and protection.

The identification EEPROM is connected to dedicated pins and designated at a specific I2C address, and allows the FlexRIO driver to identify an inserted adapter module. Including an EEPROM in your design allows for improved electrical protection for the adapter module and the NI-793*x*R/NI 797*x*R. Using an identification EEPROM prevents the use of incompatible adapter module settings such as incorrect power rails and double-driving of digital interface pins. Refer to the *EEPROM Overview* section for additional information about adding an ID EEPROM to your adapter module design.

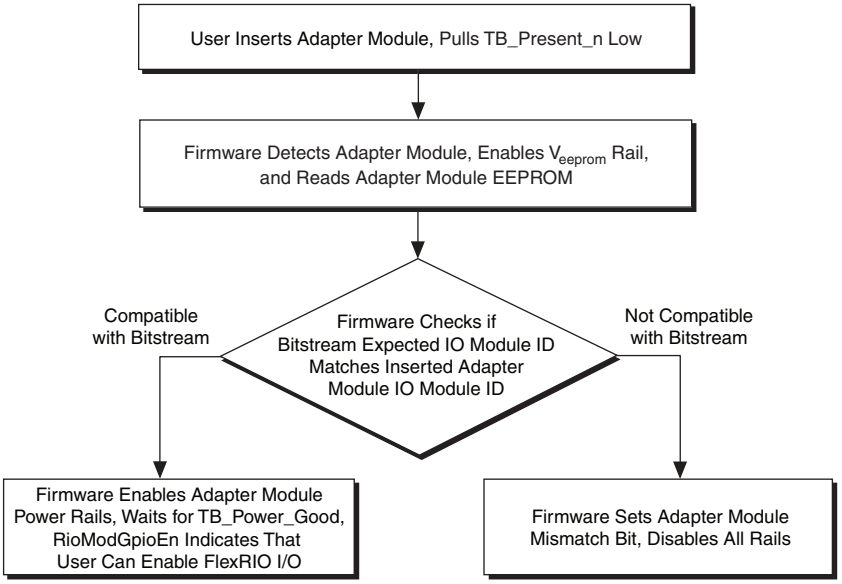
The FlexRIO device employs the following adapter module identification, power up, and removal processes to prevent applying improper voltages or double-driving signals.

Adapter Module Insertion Protocol

The following figure that depicts the process that the FlexRIO system performs when you insert a new adapter module into the NI-793*x*R/NI 797*x*R.

Note The system also performs the following steps when you download a new LabVIEW FPGA bitstream to the FlexRIO device or when you assert a Reset command.

Figure 4-2. Adapter Module Insertion Protocol



Note The steps for detecting an adapter module are performed during device configuration. For more information about adapter module configuration, refer to Chapter 10, *Configuring Your Adapter Module for Use with NI-793xR/797xR Modules and LabVIEW FPGA*.

Note You can confirm proper adapter module connection or adapter module mismatch in the IO Module Properties Status dialog in LabVIEW. In your LabVIEW **Project Explorer** window, right-click the **IO Module** item under the FPGA Target and select **Properties** to display the **IO Module Properties** dialog box. Click the **Status** category to view the adapter module **Status** dialog.

Adapter Module Removal Protocol

To properly remove an adapter module from the NI-793*x*R/NI 797*x*R, you must disable the adapter module within the LabVIEW FPGA user interface. To disable the adapter module within LabVIEW, complete the following steps:

1. In your LabVIEW **Project Explorer** window, right-click the **IO Module** item under the FPGA Target and select **Properties** to display the **IO Module Properties** dialog box.
2. Click the **Status** category to view the adapter module **Status** dialog.
3. Deselect the checkbox for **Enable IO Module Power**. When this option is deselected, firmware tristates the FlexRIO I/O and disables all adapter module power rails.
4. Click **OK**.

FlexRIO driver software allows you to enable and disable adapter modules, as well as control EEPROM access, within the LabVIEW interface. Refer to Chapter 10, *Configuring Your Adapter Module for Use with NI-793xR/797xR Modules and LabVIEW FPGA*, for information about programming your adapter module EEPROM.

EEPROM Overview

NI strongly recommends adding an EEPROM to your adapter module for identification purposes. Adding EEPROM provides better electrical protection for both the adapter module and the NI-793*x*R/NI 797*x*R. It also improves the software configuration experience within LabVIEW FPGA.

The adapter module identification EEPROM included in your design must be an I^2C-capable 2 Kbit device (256 × 8). These parts are widely available from electronics manufacturers and are often identified as 24C02. Manufacturers typically place a prefix to identify their version, for example, ST Microelectronics' EEPROM part number is M24C02.

Figure 4-3 shows a complete FlexRIO EEPROM connection circuit. This circuit shows the connections to the EEPROM (U_1) and two additional connections required for proper adapter module configuration. The supporting parts required for the EEPROM are a bypass capacitor (C_1) on the line from V_{eeprom}, a pull-up resistor (R_1) on address line A1 to V_{eeprom}, and a pull-down resistor (R_2) on the WP line. C_1 is a 0.1 µF ceramic capacitor with a voltage rating of 6.3 V or higher. For best performance, use a capacitor with an X7R dielectric. Use 4.7 kΩ resistors, which can be any resistor type rated for at least 25 V and 10% tolerance or better. Other details regarding this interface can be found in the EEPROM datasheet. For more information about EEPROM use and programming in your application, refer to Chapter 10, *Configuring Your Adapter Module for Use with NI-793xR/797xR Modules and LabVIEW FPGA*.

Note The I²C address for the EEPROM is 0x52. Use 0xA4 for write operations and 0xA5 for read operations. Use the following bit fields to ensure the correct I²C address.

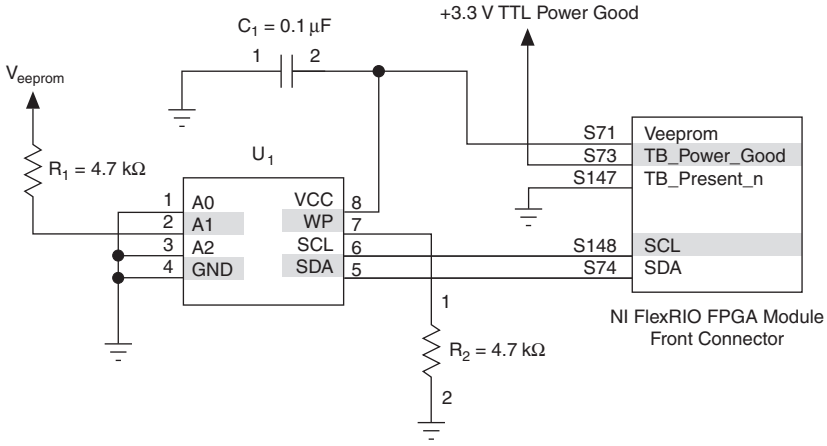
1	0	1	0	0	1	0	R/$\overline{W}$
EEPROM Device Type Identifier				Chip Enable			Operation
				A2	A1	A0	1 = R 0 = W

EEPROM Recommendations

Use an ST Microelectronics serial I²C bus EEPROM (part number M24C02-WMN) or equivalent for the FlexRIO adapter module. Refer to `www.st.com` for datasheets and additional information.

EEPROM Schematic and Wiring

Figure 4-3. EEPROM Wiring



Note Multiple components on the adapter module can share the I²C bus. Depending on the adapter module design, some of these components that share the I²C bus with the EEPROM may not power on when the initial adapter module insertion protocol executes. If you use components that pull down the I²C lines to a low level when powered off, NI recommends using a switch that separates the I²C bus from the EEPROM I²C lines.

Electrical Design Considerations

This section provides electrical considerations, clock and timing information, and additional information about the design of your adapter module.

Power up and Sequencing with External Hardware

When your FlexRIO device is connected to external equipment, you should consider the powering sequence of the system. When you insert a new adapter module, download a new bitstream to the LabVIEW FPGA target, or assert the Reset command, the adapter module power is disabled until the FPGA verifies device compatibility. Refer to the *Adapter Module Insertion Protocol* section for more information about adapter module insertion protocol.

If your external circuitry drives signals into the FlexRIO device, NI recommends disabling this circuitry when the adapter module is not powered. You can monitor adapter module power status in the LabVIEW FPGA VI and on the host VI.

FPGA I/O and Protection

Use caution when connecting adapter module circuitry to the FPGA. Refer to Xilinx documentation, available at `www.xilinx.com`, for specific FPGA I/O voltage limits. To avoid adapter module circuit damage, tristate all FPGA outputs using the adapter module socketed CLIP until the adapter module power is enabled. Refer to Chapter 12, *Creating Socketed Component-Level IP for Your Adapter Module and NI-793xR/NI 797xR Modules*, for more information about adapter module socketed CLIP. Refer to Appendix C, *Xilinx Documentation References*, for links to applicable Xilinx documentation about your FPGA.

NI recommends that you *never* directly expose signals from the front panel connector of the NI-793*x*R/NI 797*x*R to the external connectivity of the adapter module. Some form of buffering is required when routing signals from the NI-793*x*R/NI 797*x*R to the circuits external to the adapter module. Most of these parts are high speed and offer electrostatic discharge (ESD) protection as well as voltage tolerance protection. It is not necessary to consider this buffer requirement when using the NI-793*x*R/NI 797*x*R in applications such as interfacing to an ADC, as the digital signals are not directly exposed.

Grounding

FlexRIO adapter module designs may include several ground planes. The interface to the NI-793*x*R/NI 797*x*R provides the following two ground connections:

- FlexRIO digital ground—The NI-793*x*R/NI 797*x*R adapter module connector includes several pins routed to a ground plane (GND), as shown in the *NI-793xR and NI 797xR Signal Descriptions* section. All FlexRIO GPIO, control, and power pins are referenced to this ground plane. To maintain proper trace impedance, reference all FlexRIO signals to this ground plane on the adapter module.
- PXI/PXI Express chassis ground—The required FlexRIO adapter module enclosure connects to the NI-793*x*R or NI 797*x*R module front panel, and thereby to PXI/PXI Express

chassis ground by way of the guide pins and mounting screws. This chassis ground is available on the adapter module PCB by way of the four enclosure mounting screws. NI recommends including adapter module designs with provisions for a strong connection between any cable or connector shields and this chassis ground. You can make this connection through the adapter module PCB and/or the adapter module front panel. This continuous shield connection is especially important for adapter modules that must pass emissions testing.

NI-793*x*R and NI 797*x*R FPGA I/O Bank Voltages

The FlexRIO FPGA I/O are divided into three I/O banks. Each bank is powered by the same V_{cco} power supply rail. The V_{cco} setting determines which I/O standards are available in the I/O bank. You can set the V_{cco} to any of the following voltages: 3.3 V, 2.5 V, 1.8 V, 1.5 V, or 1.2 V.

FlexRIO V_{cco} voltage settings and GPIO standards are set in the adapter module configuration (`.fam`) file during device configuration.Refer to the *FlexRIO Supported Xilinx I/O Standards* section in Chapter 9, *Configuring Your Adapter Module for Use with NI 795xR/796xR Modules and LabVIEW FPGA*, which contains a list of the most common I/O standards and their required V_{cco} voltages. For more information about setting I/O standards for your FlexRIO device, refer to Chapter 9, *Configuring Your Adapter Module for Use with NI-793xR/797xR Modules and LabVIEW FPGA*. For I/O standard specifications, refer to Xilinx documentation, available at `www.xilinx.com`. Refer to Appendix C, *Xilinx Documentation References* for a list of Xilinx documents applicable to FlexRIO applications.

Note The FPGA I/O used on Virtex-5 and Kintex-7 FPGA modules support the same range of V_{cco} levels; however, the devices have different drive strength currents and slew rates. When designing a module that is compatible with both NI 795*x*R/796*x*R and NI-793*x*R/NI 797*x*R modules, consult Xilinx documentation to evaluate these I/O differences.

Simultaneous Switching Output (SSO) Noise

When multiple FPGA output drivers change state at the same time, the changing current causes a power supply disturbance. These disturbances can cause undesired behavior in output drivers, input receivers, or in internal logic, which is referred to as *simultaneous switching output (SSO) noise*. Published SSO limits determine the number and type of I/O output drivers that can change state simultaneously without introducing excessive levels of SSO noise to your application.

You should consider SSO limitations when using many GPIO as outputs. NI recommends spreading outputs across all of the GPIO banks to reduce the quantity of SSO in each FPGA bank. NI-793*x*R and NI 797*x*R modules meet the nominal PCB requirements documented for the Xilinx SSO limits. To estimate the effect of SSO noise in your application, Xilinx provides the *Virtex-5 FPGA SSO Calculator*, which contains all SSO limit data for all I/O standards. Refer to Xilinx documentation, available at `www.xilinx.com`, for additional information about SSO noise and to access the *Virtex-5 FPGA SSO Calculator*. Refer to Appendix C, *Xilinx Documentation References* for links to applicable Xilinx documentation.

Clocks and Timing

The NI-793*x*R and NI 797*x*R module front panel connectors expose six multi-region clock-capable I/O lines, as well as five pairs of single-region clock-capable I/O lines. These inputs provide an external clock to the FlexRIO FPGA for use in synchronous interfaces.

It is important to consider timing requirements for any synchronous interface to the FlexRIO FPGA. I/O timing is defined using the socketed CLIP VHDL and the constraints file, which are two elements that comprise the socketed CLIP interface.

Note If you are using LabVIEW 2013 or earlier, define your constraints in the `.ucf` file. Refer to the *ExampleIOModuleCLIPK7.ucf* section of Chapter 12, *Creating Socketed Component-Level IP for Your Adapter Module and NI-793xR/NI 797xR Modules*, for more information about using the `.ucf` file. If you are using LabVIEW 2014 or later, define your constraints in the `.xdc` file. Refer to the *ExampleIOModuleCLIPK7.xdc* section of Chapter 12, *Creating Socketed Component-Level IP for Your Adapter Module and NI-793xR/NI 797xR Modules*, for more information about using the `.xdc` file.

Use the adapter module CLIP constraints file to provide necessary period and offset constraints, or to provide setup/hold requirements to the Xilinx compiler. You can also add digital clock managers (DCMs) or phase-locked loops (PLLs) in your CLIP for phase shifting or frequency multiplication. For example, an adapter module could include an ADC which drives 16-bit samples along with a synchronous clock to the FPGA. The ADC datasheet specifies the relationship between the clock and data output of the part. In this case, the CLIP constraints should contain a period constraint for the external clock, as well as an offset constraint for the incoming data relative to the external clock.

For information about adding timing constraints to your CLIP, refer to Chapter 12, *Creating Socketed Component-Level IP for Your Adapter Module and NI-793xR/NI 797xR Modules*. For more information about timing constraints, FPGA clock input frequency limitations, and DCM/PLL details, refer to Xilinx documentation, available at `www.xilinx.com`. Refer to Appendix C, *Xilinx Documentation References*, for a list of Xilinx documents applicable to FlexRIO applications.

Clock-Capable I/O Signals

Each NI-793*x*R and NI 797*x*R FPGA I/O target exposes six multi-region clock-capable (_CC) pairs as well as five single-region clock-capable pairs. NI recommends using multi-region clock-capable pairs in your applications when possible. You can configure the clock-capable signals as differential pairs, a single-ended positive pin signal, or as a generic GPIO line. If you configure the lines as single-ended lines, you must route the clocking signal to the positive side. The negative side can be used as a GPIO line.

Refer to the *NI-793xR and NI 797xR Signal Descriptions* section for your device's pinout chart.

As GPIO signals, these signals can be either inputs or outputs. You can also use these signals to pass a clock into the FPGA. These clock-capable signals have access to clocking resources in

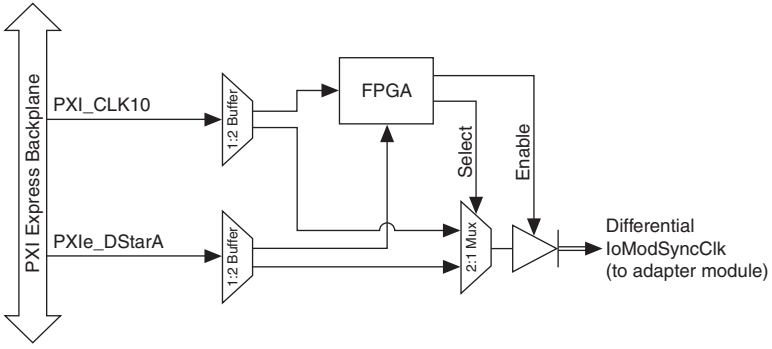
the FPGA that correspond to multi-region clock capable (MRCC) inputs in the FPGA. Refer to Xilinx documentation, available at `www.xilinx.com`, for more information about clock-capable signals and clocking. Refer to Appendix C, *Xilinx Documentation References*, for a list of Xilinx documents applicable to FlexRIO applications.

IoModSyncClk

Note NI-793*x*R modules do not use the IoModSyncClk line. Leave IoModSyncClk unconnected for NI-793*x*R devices.

The NI 797*x*R provides the IoModSyncClk signal for synchronization between adapter modules. IoModSyncClk is a low-voltage, positive-emitter-coupled logic (LVPECL) clock that can either be sourced by PXI_CLK10 or by PXIe_DStarA, as shown in the following figure.

Figure 4-4. IoModSyncClk Source Block Diagram

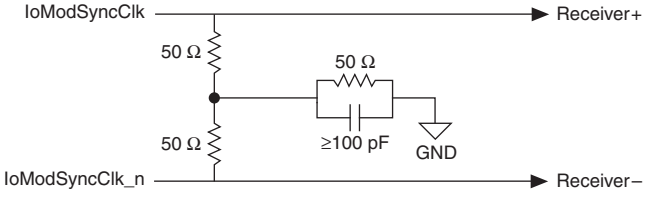


For more information about the IoModSyncClk and using it in your adapter module design, refer to the *IoModSyncClk.fam Values* section of Chapter 10, *Configuring Your Adapter Module for Use with NI-793xR/797xR Modules and LabVIEW FPGA*. Refer to the *FlexRIO FPGA Module Base Clock Properties* topic in the *FlexRIO Help*.

Termination

If an adapter module uses IoModSyncClk, terminate both lines with a 50 Ω load to 1.3V (or Thevenin equivalent) as close to the receiver as possible. The following figure shows the recommended termination for IoModSyncClk.

Figure 4-5. IoModSyncClk Termination



GPIO Termination and Impedance

All GPIO traces from the FPGA to the front panel connector are routed as 100 Ω differential/ 50 Ω single-ended traces. For best performance, NI recommends designing the trace impedance in the adapter module PCB to match the driver output impedances.

For example, if your adapter module uses GPIO as single-ended outputs to IC inputs, route these signals as 50 Ω single-ended traces. If your adapter module drives an LVDS signal into the FlexRIO FPGA, route this signal as a 100 Ω differential signal. In the case of FPGA LVDS inputs, 100 Ω parallel termination at the receiver may be enabled using the socketed CLIP constraints file. For more information, refer to Chapter 12, *Creating Socketed Component-Level IP for Your Adapter Module and NI-793xR/NI 797xR Modules*.

Minimizing Crosstalk

All FlexRIO GPIO signals are routed to maximize flexibility, allowing them to be used as 132 single-ended signals or 66 differential pairs (NI 795*x*R and NI 796*x*R), and 136 single-ended signals or 68 differential pairs (NI-793*x*R and NI 797*x*R). GPIO*x* and GPIO*x*_n are differentially coupled, and can induce crosstalk on each other when used as independent, single-ended signals.

For maximum performance in these single-ended applications, NI recommends using only one side of a differential pair and connecting the opposite side of the pair to ground. Use this method only for the highest performance applications as doing so reduces the number of available GPIO signals.

Sharing the I^2C Bus

The NI-793*x*R and NI 797*x*R module front panel connectors include a dedicated 3.3V I^2C bus for the ID EEPROM. This bus is powered from the V_{eeprom} rail and accessible by way of the FlexRIO driver and the adapter module socketed CLIP. You can attach other I^2C devices to this bus only if the I^2C bus address is unique.

Note The power on protocol for the NI-793*x*R and NI 797*x*R modules uses the I^2C bus in a manner that may affect your application. Specifically, during adapter module identification, only the V_{eeprom} rail is powered, which means any user circuitry is unpowered. The I^2C bus interface signals (SCL and SDA), however, are toggling. If another I^2C device *not* powered from V_{eeprom} shares the I^2C bus to the ID EEPROM, the SCL and SDA signals may be pulled down on the unpowered device, which prevents the identification and powering of your NI-793*x*R or NI 797*x*R module. Consult your I^2C datasheet to determine if additional circuitry is necessary to ensure proper adapter module identification while sharing the I^2C bus among multiple devices.

Choosing Circuitry Components

When designing the interface circuitry between the adapter module and the NI-793*x*R or NI 797*x*R module, NI recommends choosing 2.5V or lower components whenever possible. As FPGA process geometrics decrease, the supported interface levels also decrease. Using 2.5V or lower components makes present adapter module designs more likely to remain compatible with future FPGA families.

Unused Pin Recommendations

The following table lists the NI recommendations for the unassigned/unused pins on the NI-793*x*R and NI 797*x*R modules.

Table 4-5. NI-793*x*R/NI 797*x*R Unassigned Pin Recommendations

Pin	Recommended Use
GPIO	Leave unconnected.
V_{cco}	Leave unconnected.
+12V, +3.3V	Leave unconnected.
IoModSyncClk	If an adapter module does not use IoModSyncClk, leave the lines unconnected and unterminated.

5

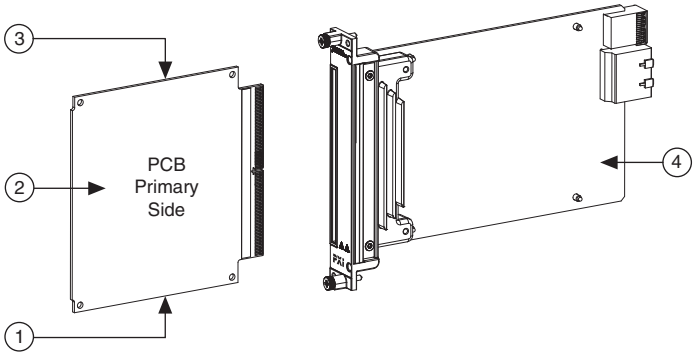
Printed Circuit Board (PCB) Design Considerations

This chapter provides mechanical and physical information about designing your custom printed circuit board (PCB). This information includes information about the board outline, placement keepout areas, height restrictions, and the card edge connector.

PCB Orientation

Figure 5-1 shows the orientation of the adapter module PCB in reference to the FlexRIO FPGA module.

Figure 5-1. FlexRIO FPGA Device Assembled Front Panel and Example Adapter Module



1	Bottom Adapter Module Edge	3	Top Adapter Module Edge
2	Example Adapter Module	4	FlexRIO FPGA Module

PCB Design Concepts

When you create an adapter module, you must use computer aided drafting (CAD) tools to create the PCB design. To begin your PCB design, you must first create a schematic that contains all symbolic electrical connections between parts and connectors. Next, perform a layout using the schematic to physically connect electrical components using copper traces on the PCB.

The schematic tool uses symbols to represent each component being used. You can create custom symbols to facilitate the development of your schematic diagram. The method for creating these custom symbols varies depending on the tool being used. Although custom symbols are useful in some instances, NI recommends using the pin names and numbers

provided in this document in order to maintain consistency with the documentation and to assist in communication with National Instruments if you require support. When creating a custom symbol for the adapter module card edge connector, refer to Appendix A, *Signal Suggestions*, for graphical representations of some suggested adapter module symbol signal assignments.

To ensure a correct design, PCB design tools require a different set of inputs. Mechanical parameters such as board outline, hole and part locations, and board thickness, must be accurately described to ensure a successful board fabrication. To assist in designing the adapter module PCB, the FlexRIO Adapter Module Development Kit ships with media that contains Standard for the Exchange of Product model data (STEP), Initial Graphics Exchange Specification (IGES), Pro/ENGINEER, PDF, and AutoCAD® DXF files that call out the precise dimensions used to create a PCB.

Note NI strongly recommends using the electronic design files when designing the PCB. NI also recommends using metric measurements whenever possible; English measurements may suffer from round-off error. The dimensional diagrams in this manual and in the PDF files are provided for reference. NI does not recommend using the manual reference diagrams or the PDF files as the primary resource when designing the PCB.

Note NI develops its electronic design files with the latest version of DXF software. Ensure that all of your design software is compatible with this software version. If you have trouble viewing the electronic design files, contact `ni.com/support`.

Table 5-1 lists the design files and the elements they contain.

Table 5-1. FlexRIO Custom Adapter Module Design Files

File Name*	Description
`assembly.xxx`	Complete adapter module enclosure with PCB.
`bottom-housing.xxx`	Bottom half of enclosure.
`gold-fingers.xxx`	Gold finger shapes and outlines.
`panel-front-dim.xxx`	Dimensioned front panel.
`panel-front-outline.xxx`	Front panel shape and outline.
`pcb-dim.xxx`	Dimensioned PCB drawing.
`pcb-dim_1.dxf`	Page one of the dimensioned PCB drawings in the `.dxf` format.
`pcb-dim_2.dxf`	Page two of the dimensioned PCB drawings in the `.dxf` format

Table 5-1. FlexRIO Custom Adapter Module Design Files (Continued)

File Name*	Description
pcb-dim_3.dxf	Page three of the dimensioned PCB drawings in the .dxf format
pcb-outline.*xxx*	PCB outline and shape
top-housing.*xxx*	Top half of enclosure
* .*xxx* represents the file type. Available file types are .dxf, .igs, .stp, .pdf, and the Pro/ENGINEER file types, .prt.*x* and .asm.	

The design files install in the following location:

- **Windows XP**

  `C:\Documents and Settings\All Users\Shared Documents\National Instruments\FlexRIO\Module Development Kit\Design Files`

- **Windows 7/Vista**

  `C:\Users\Public\Documents\National Instruments\FlexRIO\Module Development Kit\Design Files`

- **Windows 8**

  `C:\Users\Public\Documents\National Instruments\FlexRIO\Module Development Kit\Design Files`

The FlexRIO adapter module provides approximately 355.6 sq. mm (14 sq. in.) of usable space for circuitry on the PCB primary side. The multiple keepout areas must be observed to prevent part damage and mechanical interference with the enclosure. Keepout areas are located along the periphery of the PCB and around each of the four mounting holes. There is also a large keepout area near the card edge connector where the two halves of the enclosure fit together. Refer to Figures 5-6 and 5-7 for exact locations and dimensions of the PCB keepout areas.

Note The electronic design files provided by NI denote all keepout areas. NI strongly recommends using these design files when designing the PCB.

The PCB primary side allows for a component height of 13.97 mm (0.55 in.) to prevent contact of the electrical components with the top of the enclosure. The PCB secondary side is also available for component placement, however, the clearance between the back of the PCB secondary side and the surface of the enclosure is 1.19 mm (0.047 in.). A small step in the metal housing near the I/O connector window and the card edge connection provide wide areas 19.43 mm (0.765 in.) and 21.49 mm (0.846 in.) where the component clearance is 1.40 mm (0.055 in.) on the secondary side. Therefore, place only very low-profile components such as small passives on the PCB secondary side to avoid mechanical interference.

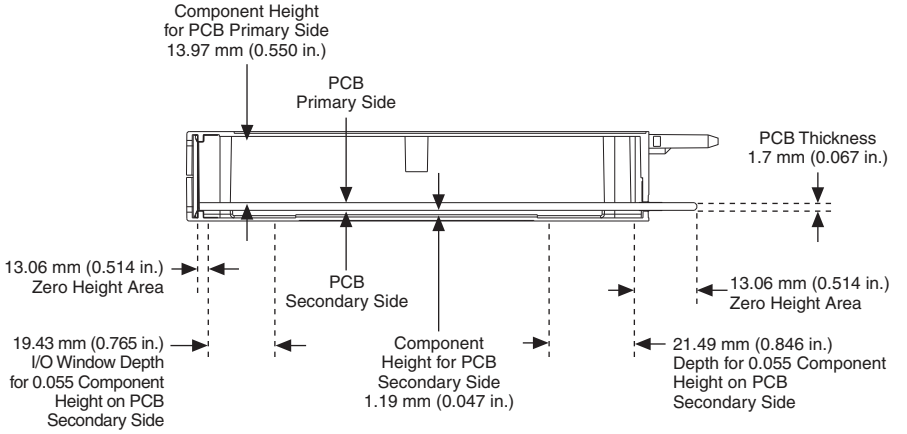
Caution Adapter module designs that include vias or traces in the card edge connector area on either side of the PCB make contact with bottom of the card edge connector EMI gaskets on the adapter module enclosure. To prevent electrical shorts or damage to the these traces, you *must* use the insulators shipped with your adapter module enclosure to protect the vias/traces from contact with these EMI gaskets.

The finished PCB thickness should be 1.70 mm (0.067 in.) ± 0.127 mm (0.005 in.) as measured metal-to-metal over the gold card edge contact area, excluding the solder mask and silkscreen.

Note The specified PCB thickness tolerance is expressed as an absolute number, ± 0.127 mm (0.005 in.), not as a percentage.

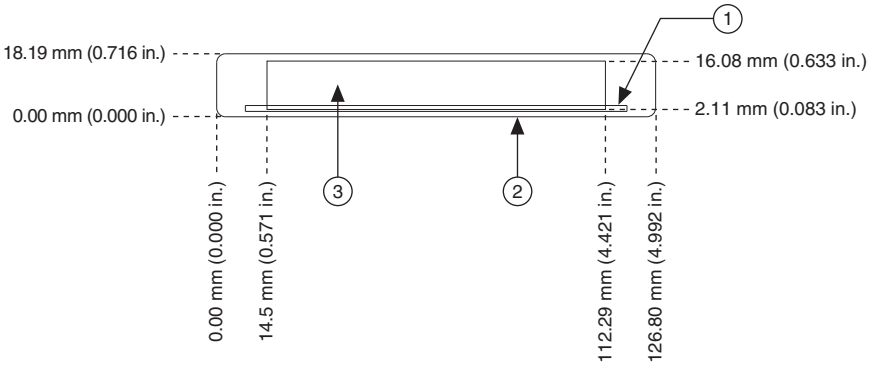
Figure 5-2 shows the recommended component clearance dimensions.

Figure 5-2. Adapter Module Cross Section Showing Component Clearance Dimensions



The input/output window at the front of the housing provides an opening that is 98.79 mm (3.85 in.) by 13.97 mm (0.55 in.), where you can include connectors to signals external to the adapter module. This I/O connector window accommodates most signal connectors, such as SMA and BNC. When designing with through-hole parts, you must maintain the secondary side component clearance. Many connectors that are traditionally a through-hole design are now offered in surface mount packages. To keep your PCB design within the clearance parameters, NI recommends using surface mount or limited protrusion through-hole parts whenever possible. Figure 5-3 shows the I/O connector window in relation to the enclosure front panel and the PCB. For more information about module enclosure front panel dimensions, refer to Chapter 7, *Module Enclosure*.

Figure 5-3. I/O Connector Area Clearance Dimensions

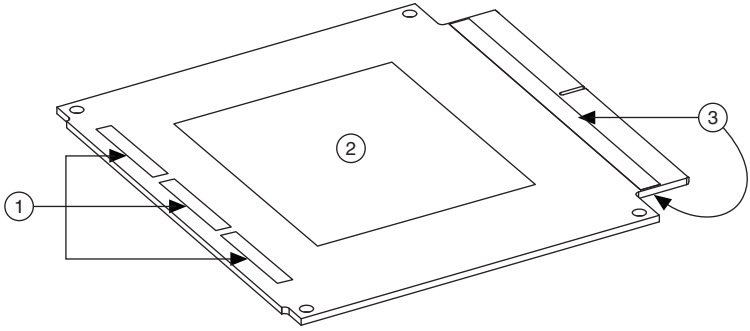


1	PCB	2	Front Panel	3	I/O Connector Area

Note The enclosure has a backside insulator to help prevent shorting secondary side components. Ensure that your PCB design does not allow any secondary side surface mount or through-hole part leads to make contact with this insulator.

Figure 5-4 shows possible locations on the PCB for the design elements of your adapter module.

Figure 5-4. Example Adapter Module PCB and Design Element Locations



1	User-Defined Connectors	3	Mylar Insulators (Both Sides)
2	Custom Measurement Circuitry		

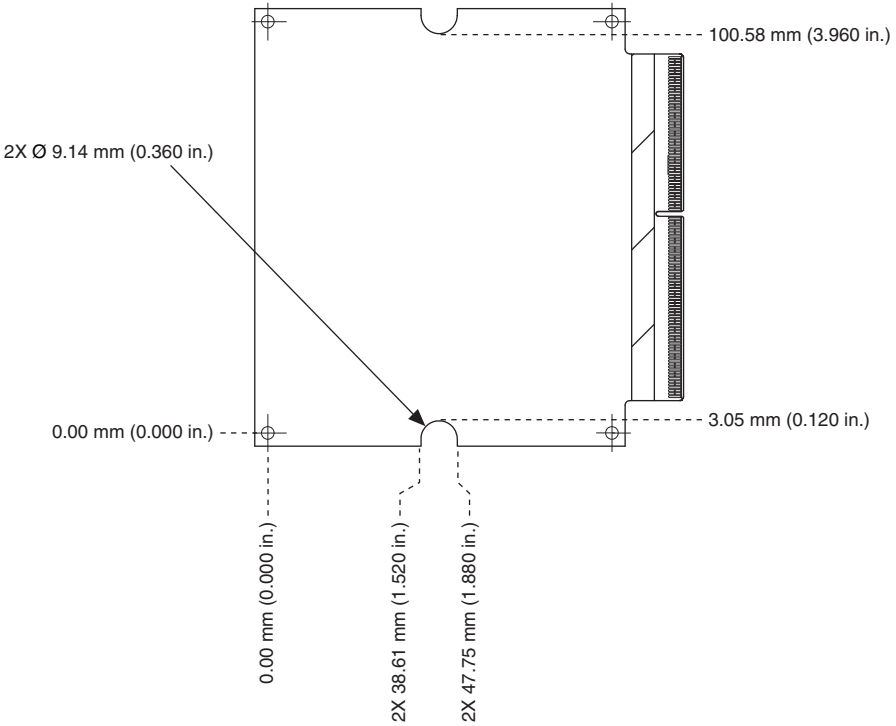
PCB Dimensions

The following figures show the dimensional requirements of the PCB. When creating the PCB in a CAD tool, place the origin of the design in the lower-left mounting hole. This placement facilitates recommended placement of components, as no other coordinate translation is required.

Side Notches

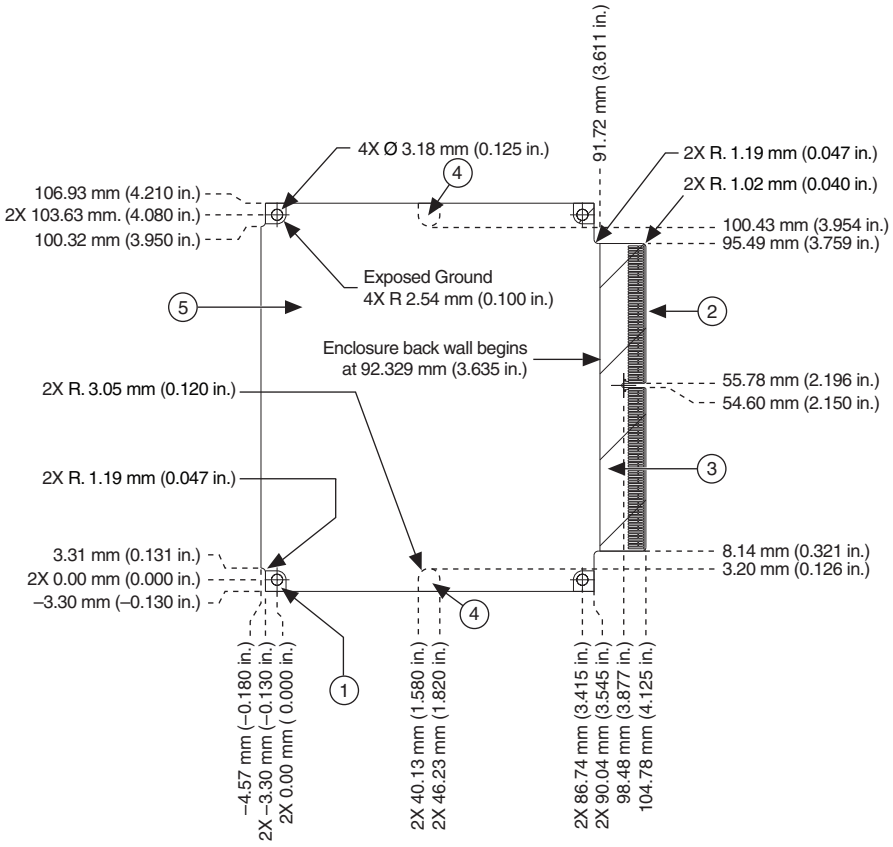
The FlexRIO adapter module 2.0 enclosures do not require the notches at the midpoints along the top and bottom edges of the PCB because the four corner screws fasten the entire enclosure together. If you want to ensure backwards compatibility with the original 1.0 adapter module enclosures, you must include the notches on the PCB. The top enclosure housing has a threaded hardware mounting boss in this area, 5.588 mm (0.220 in.) above the top side of the PCB. If you use this boss, the retained side notches provide access to the screw heads when the PCB is installed in the enclosure. Figure 5-5 shows the dimensions of the PCB notches used with the FlexRIO adapter module 1.0 enclosures.

Figure 5-5. PCB Notches Required for 1.0 Enclosures



Note Use Figure 5-6 for quick reference only. For detailed information on PCB dimensions, refer to the electronic design files provided by NI.

Figure 5-6. Adapter Module PCB Primary Side Dimensions

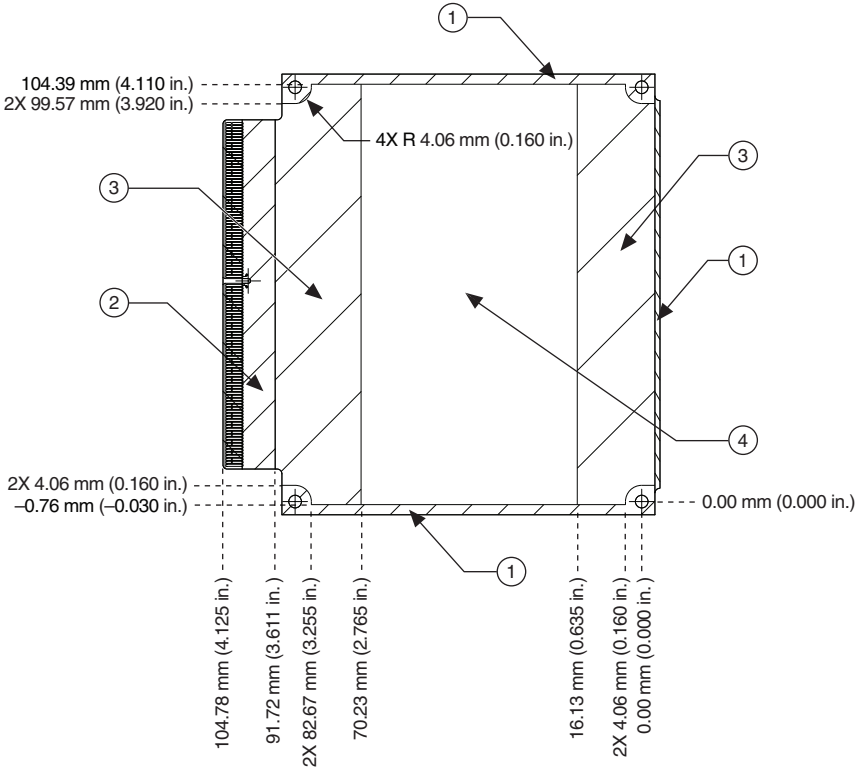


1 Exposed Ground Around Mounting Hole—4X ∅ 5.08 mm (0.200 in.)
2 Card Edge Connector
3 Component Keepout Area and location of Mylar Insulator
4 5.59 mm (0.220 in.) Height Keepout Area
5 13.97 mm (0.550 in.) Height Keepout Area

Note Use Figure 5-7 for quick reference only. For detailed information on PCB dimensions, refer to the electronic design files provided by NI.

For more detailed card edge connector dimensions, refer to Chapter 6, *Card Edge Connector*.

Figure 5-7. Adapter Module PCB Secondary Side Dimensions



1 Component/Via and Non-Ground Outer Route Keepout Area
2 Component Keepout Area and Location of Mylar Insulator
3 1.40 mm (0.055 in.) Height Keepout Area
4 1.19 mm (0.047 in.) Height Keepout Area

GPIO Trace Routing

The GPIO traces between the FPGA and the adapter module connector on the FlexRIO FPGA module/Controller for FlexRIO have controlled electrical properties. All traces are length-matched to within 2.54 mm (100 mils), and pairs are matched to 0.25 mm (10 mils). Also, the PCB stack-up and trace widths are configured such that GPIO pair differential impedance is 100 Ω, and GPIO single-ended impedance is 50 Ω.

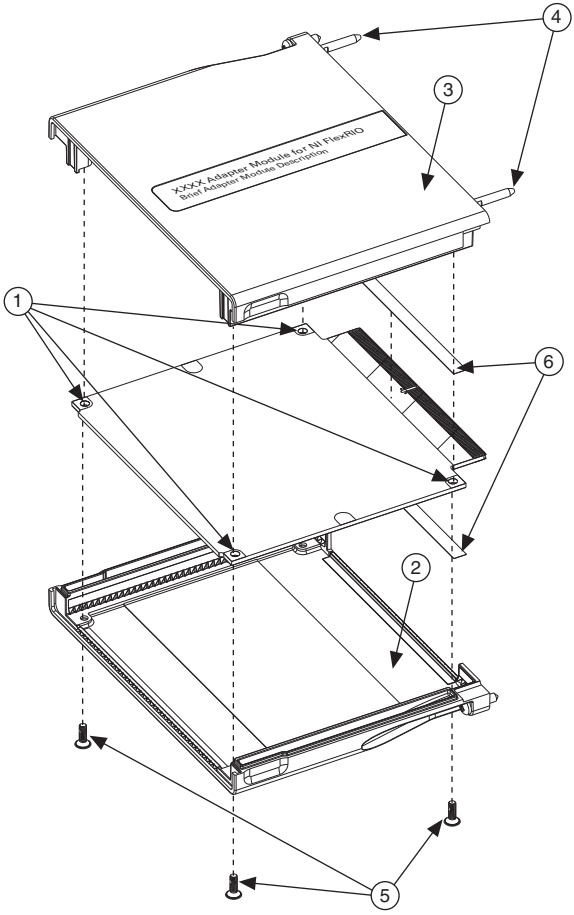
Similar care should be taken for high-speed trace routing on the adapter module. To do so, configure the PCB stack-up and trace widths to maintain the desired impedance, as well as matching trace lengths to reduce GPIO skew. Many PCB board houses have tools or utilities that recommend a particular stack-up based on the layer count and trace width used on a given PCB. You can relax these requirements for low-edge rate signals or static control lines.

Grounding Considerations

For EMC-compliant adapter modules, NI recommends incorporating the following grounding considerations into your design.

The required FlexRIO adapter module enclosure provides access to PXI/PXI Express chassis ground by way of the alignment pins and mounting screws. You can connect adapter module circuitry to this chassis ground by way of the four adapter module PCB mounting holes, as shown in Figure 5-8. You can also use this connection to connect any front panel connector shield to chassis/enclosure ground.

Figure 5-8. Adapter Module Enclosure with Mounted PCB



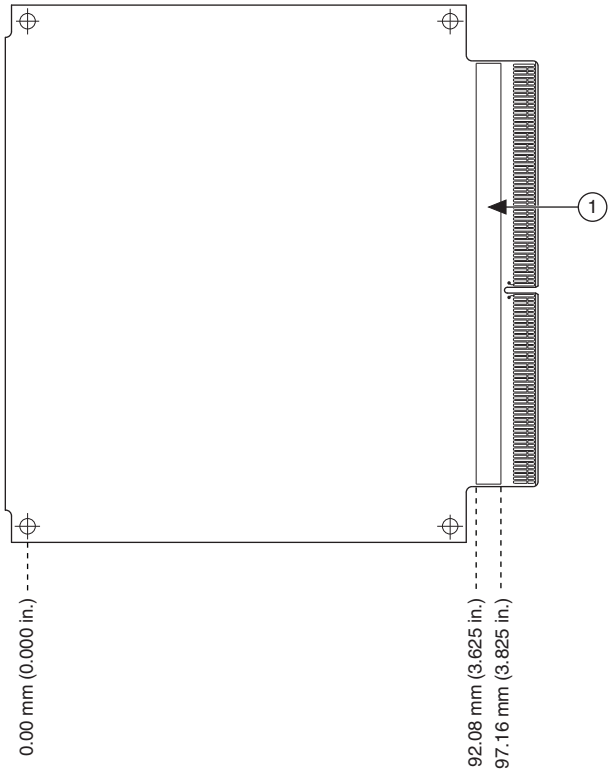
1	Exposed Ground Around Mounting Holes	4	Alignment Pins
2	Adapter Module Enclosure (Bottom)	5	PCB Mounting Screws
3	Adapter Module Enclosure (Top)	6	Mylar Insulators

Note To reduce possible noise emissions between chassis GND and digital GND, NI recommends adding provisions in your design for populating capacitors between chassis GND and digital GND connections on your circuitry design.

Mylar Insulators

Your FlexRIO Module Development Kit ships with three clear protective Mylar insulators (NI part number 195549A-01) sized to fit your PCB, 0.051 × 5.08 × 86.36 mm (0.002 × 0.2 × 3.4 in.). You *must* install two insulators as shown in Figures 5-8 and 5-9 on the PCB of each adapter module. Without these insulators, the copper EMI gaskets that seal the enclosure rub directly on the surface of the PCB during assembly, which may cause electrical shorts.

Figure 5-9. Mylar Insulator

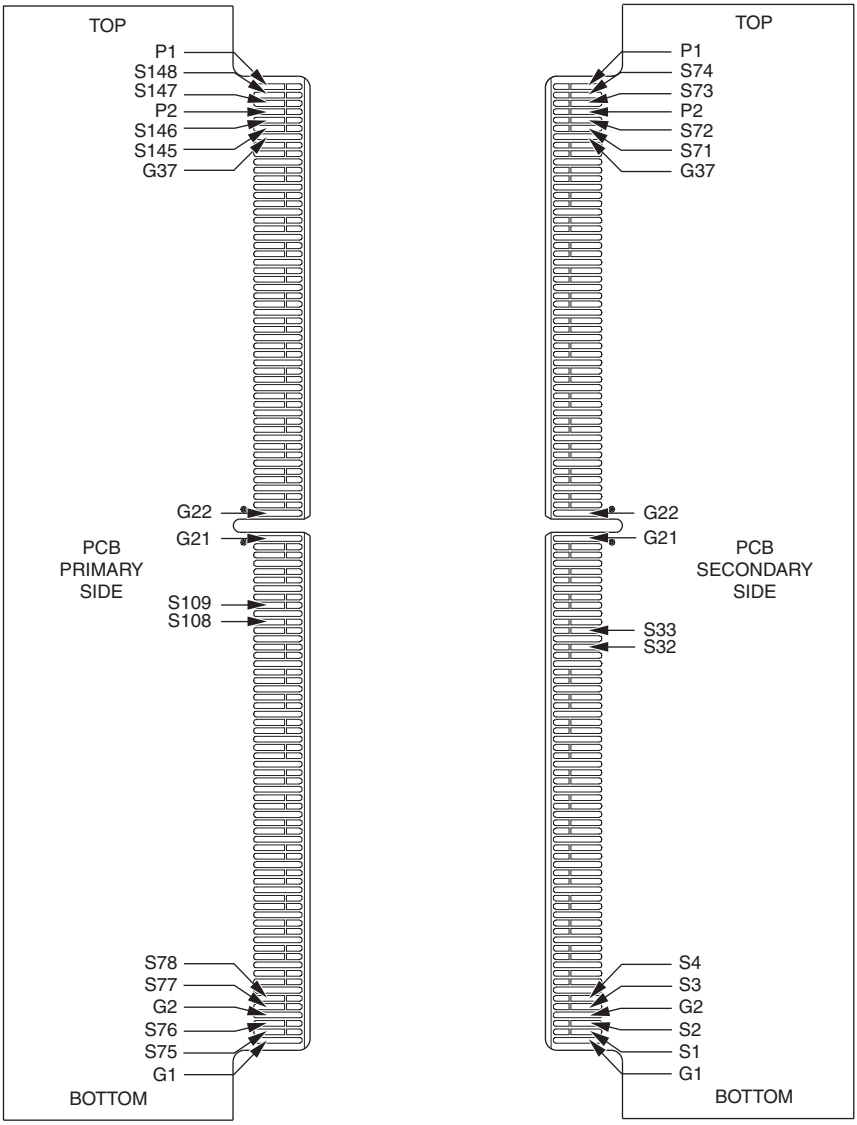


1 Mylar Insulator

Pin Locations

Figure 5-10 illustrates the card edge connector and its associated pin assignments. Use Figure 5-10 for quick reference only. For detailed information about this connector and how to create it, refer to Chapter 6, *Card Edge Connector*.

Figure 5-10. Physical Pin Locations



PCB Finishing

The final process performed during PCB manufacturing is surface finish. The surface finish provides a solderable surface for components, to prevent oxidation (tarnish), and to provide any mechanical structure to features that may need it. The fabrication house that you choose to manufacture your PCB may have different finishes available for selection.

PCB finish options are trending toward RoHS compliance. This standard means that the finishes conform to the reduction of hazardous materials requirements, usually by using silver and gold as the primary PCB finishes. Immersion silver is the most common finish for most of a PCB surface area. Immersion silver offers good shelf life, low tarnishing, good wetting for manufacturability, and fewer defects.

The adapter module card edge connector also has specific finishing requirements. For more information about these requirements, refer to the *Card Edge Connector Finishing* section of Chapter 6, *Card Edge Connector*.

6 Card Edge Connector

This chapter covers the mechanical aspects of the card edge connector construction and dimension details.

Connector Description

The FlexRIO FPGA module and Controller for FlexRIO use a high-density, high-performance card edge connector. The connector has a key mechanism slightly off-center to ensure correct orientation when mating with the adapter module.

Figure 5-1, *FlexRIO FPGA Device Assembled Front Panel and Example Adapter Module*, shows a detailed view of the assembled FlexRIO FPGA module front panel. The adapter module PCB design must fit into the assembled front panel slot in order to make contact with the FPGA module front connector. Any PCB designed to mate with the FPGA module front connector must exactly match the following card edge connector description to assure proper and reliable connectivity between the FlexRIO FPGA module and any custom adapter module.

The card edge connector that plugs into the FlexRIO FPGA module front connector is created as part of the PCB. The actual physical representation of the card edge connector is called the cell. The design files (pro/e, STEP, IGES, DXF, and PDF) shipped with your FlexRIO Adapter Module Development Kit provide a fully dimensioned drawing that you can use to create an accurate cell.

Note NI strongly recommends using the electronic design files when designing the PCB. NI also recommends using metric measurements whenever possible; English measurements may suffer from rounding errors. This manual provides dimensional diagrams for reference only. NI does not recommend using the manual reference diagrams as the primary resource when designing the PCB.

Note NI develops its electronic design files with the latest version of DXF software. Ensure that all of your design software is compatible with this software version. If you have trouble viewing the electronic design files, contact `ni.com/support`.

For more information about these design files, refer to the following chapters:

- NI 795*x*R/796*x*R: Chapter 3, *Interfacing Adapter Modules with NI 795xR and NI 796xR Modules*
- NI 797*x*R/793*x*R: Chapter 4, *Interfacing Adapter Modules with NI-793xR and NI 797xR Devices*

The design files install in the following location:

- **Windows XP**

  `C:\Documents and Settings\All Users\Shared Documents\National Instruments\FlexRIO\Module Development Kit\Design Files`

- **Windows 8/7/Vista**

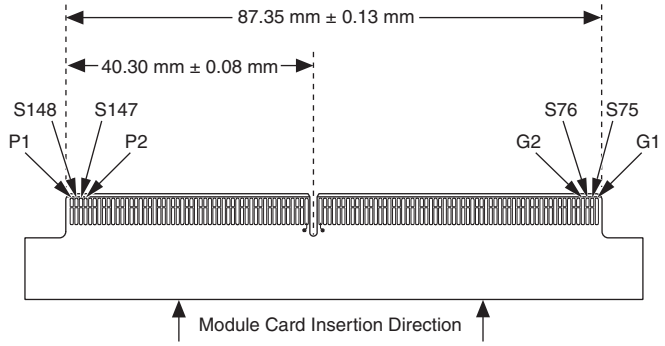
  `C:\Users\Public\Documents\National Instruments\FlexRIO\Module Development Kit\Design Files`

The card edge connector finger arrangement meets two design requirements. First, the adapter module has some protection from hot-swapping, in that the ground points should be the first to make contact during an insertion operation and the last to break contact during a removal. This contact design uses shorter, recessed signal and power contacts, which allows for longer ground contacts. The second requirement is that the density and fine-pitch nature of the connector encourages metal-to-metal contact at all times. If the PCB design allows the spring contact in the FlexRIO FPGA module to drag across bare PCB material, the FPGA module spring contact may scratch the PCB material and generate debris that could interfere with the electrical connection between the spring and the pad.

The gold fingers and the card edge connector should conform to the following physical design guidelines:

- Figure 6-1 shows the mechanical dimension of the portion that is inserted into the front of the FlexRIO FPGA module.

Figure 6-1. Card Edge Connector Keying Dimensions

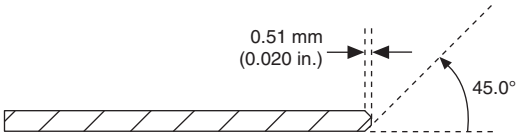


Note The connector key position must be in the correct location in order to align with the FlexRIO FPGA module front panel.

Note Card edge connector dimensions are defined in millimeters. Converting your measurement to inches may cause rounding errors.

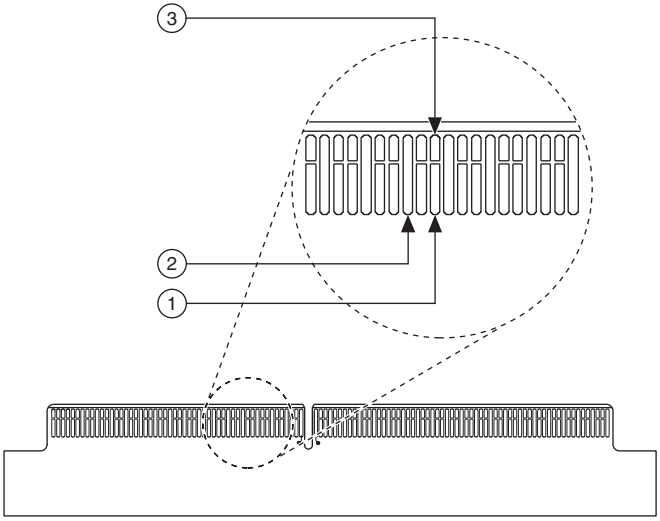
- Figure 6-2 shows the PCB connector edge.

Figure 6-2. Adapter Module PCB Chamfer at Gold Finger Edge



- Figure 6-3 shows the two different styles of finger design. The longer monolithic finger is used for the ground connections, and the split finger is used for signaling and power. These two designs are used together to insure that ground connections are the first to make contact when inserting the adapter module and are the last to break contact when removing the adapter module.

Figure 6-3. Gold Finger Electrical Connections



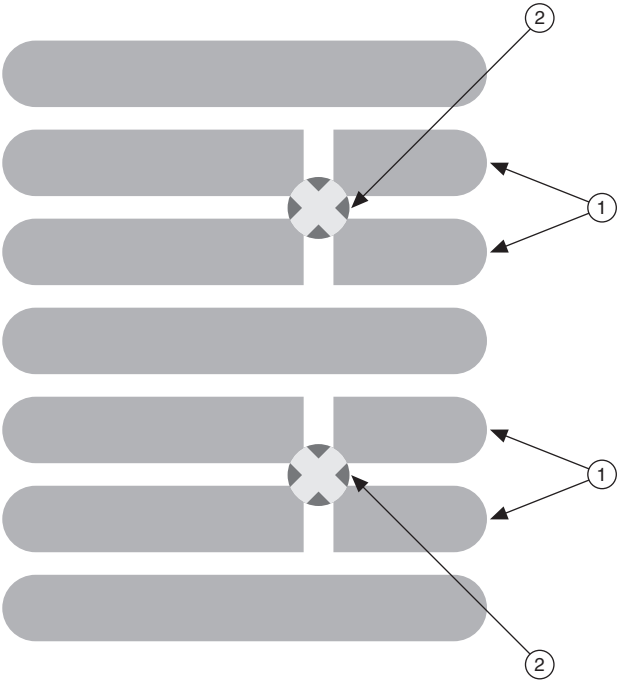
1 Split-Finger Pad—Connected to Signal/Power
2 Monolithic Finger Design for Ground Connections
3 Split-Finger Pad—Unconnected to Increase Life of Signal/Power Connector Pads

Card Edge Connector Finishing

The adapter module PCB card edge connector plugs into the connector located on the FlexRIO FPGA module. The spring contacts of this connector are gold plated. Therefore, for optimum performance, NI recommends using gold plating for the card edge connector fingers. Gold-on-gold connections provide the most reliable physical and electrical connection. Hard gold plating is the recommended finish for these fingers, hence the term *gold fingers*.

A proper gold plating process requires an electrical connection on the plated surface. Isolated gold fingers, though, do not have an electrical connection. To use the standard plating bar process instead of the selective gold plating process, NI recommends creating a 0.003 in. wide X-trace connection between gold finger pairs. Refer to Figure 6-4 for information about how to design the X-trace. After plating, you can remove these X-traces by drilling a 0.017 in. (+0.002/-0.001 in.) non-plated round hole using a square end mill-style drill bit to minimize copper burrs. You should drill 38 X-trace holes total.

Figure 6-4. X-trace Between Isolated Gold Finger Pairs



1 Isolated gold fingers 2 X-traces

ENIG is a lower-cost gold solution which uses immersion gold an all surfaces of the PCB. This process deposits a very thin layer of gold on the fingers. This method does not ensure long connector life for repeated insertions. For applications where you expect frequent adapter module insertion and removal, NI does not recommend the ENIG process.

Note NI does not recommend Hot Air Solder Leveling (HASL) finishing, as it is minimally controlled and the co-planarity of the finish can cause binding in the connector.

7

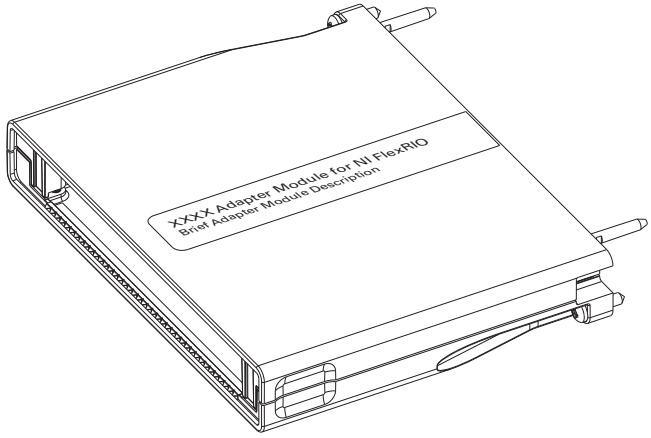
Module Enclosure

You must use the enclosure provided in the FlexRIO adapter module development kit to house the adapter module PCB. You can purchase additional enclosures from NI.

Note NI requires that the enclosure be included in the complete adapter module design to ensure proper connectivity and electrical protection.

Figure 7-1 shows the adapter module enclosure designed by NI. This enclosure is a metal housing for the adapter module that provides easy interfacing to the PXI/PXI Express system. The enclosure also provides electrical shielding, mechanical mounting of a custom PCB, and support for a custom front panel for mounting custom connectivity.

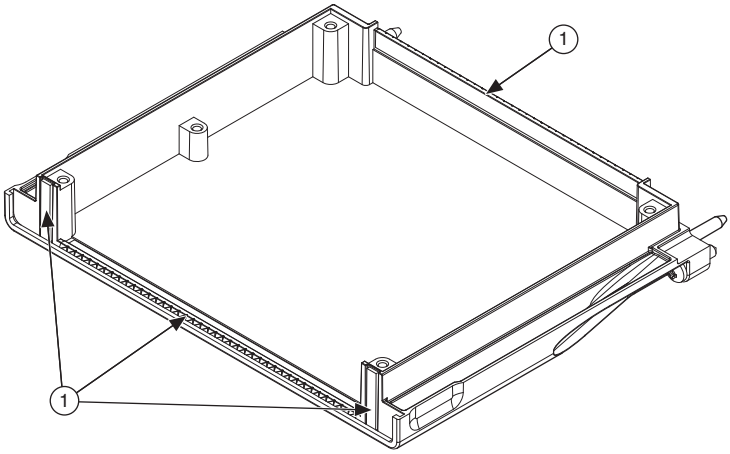
Figure 7-1. Adapter Module Enclosure



EMI Gaskets

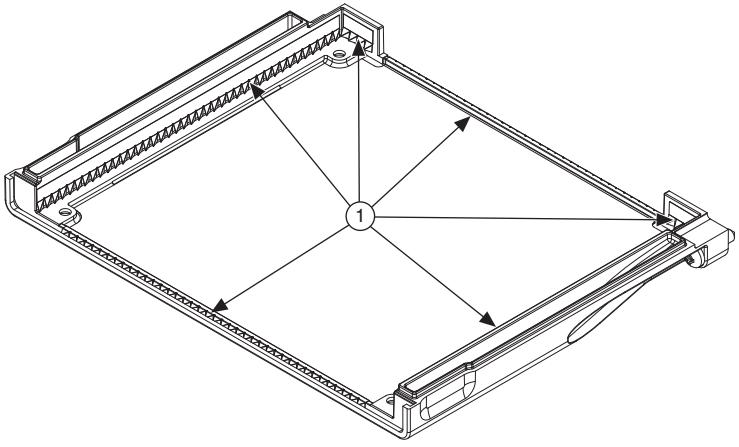
NI redesigned the adapter module enclosure (version 2.0) for the FlexRIO Adapter Module Development Kit 2.0 release. This new enclosure includes installed EMI gaskets, as shown in Figures 7-2 and 7-3, on each side of the enclosure for improved grounding, shielding, and EMI performance.

Figure 7-2. EMI Gasket Locations on Module Primary Side



1 EMI gaskets

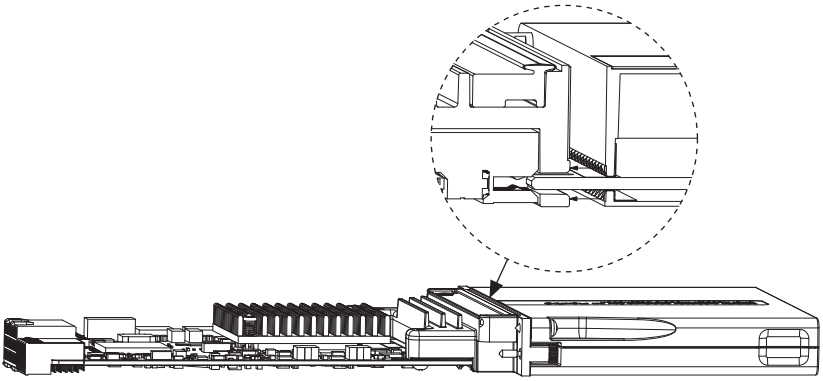
Figure 7-3. EMI Gasket Locations on Module Secondary Side



1 EMI gaskets

By better sealing the connection between the FlexRIO FPGA module/Controller for FlexRIO and the adapter module, these EMI gaskets also improve the EMI performance across the gap between the two devices, as shown in Figure 7-4.

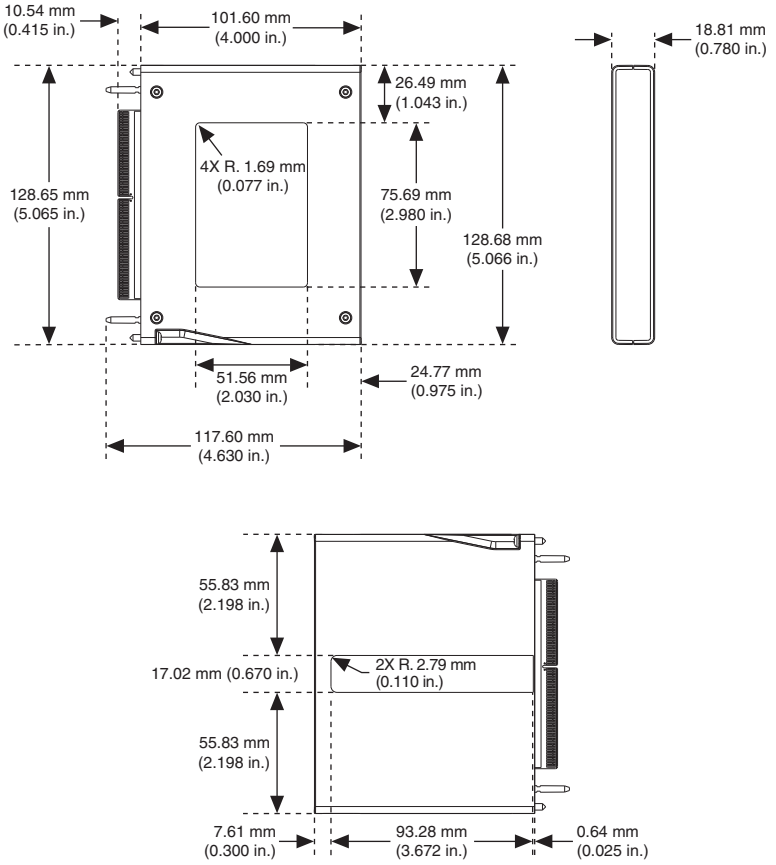
Figure 7-4. Improved Module Connections



Enclosure Dimensions

Figure 7-5 shows a dimensioned drawing of the enclosure.

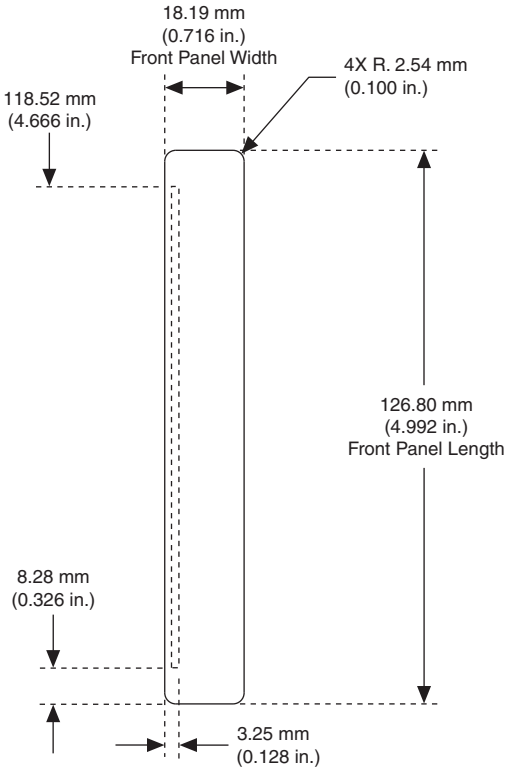
Figure 7-5. FlexRIO Adapter Module Enclosure Dimensions



Note Use Figure 7-5 for reference only. For detailed information about adapter module enclosure dimensions, refer to the electronic design files provided by NI.

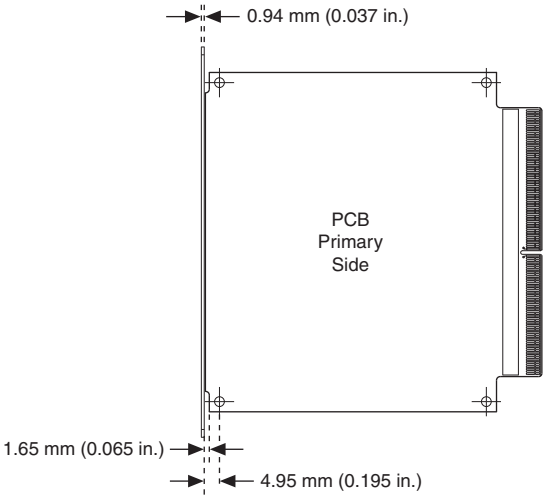
Figures 7-6 and 7-7 show the module enclosure front panel dimensions in relation to the PCB dimensions and placement.

Figure 7-6. Front Panel Dimensions and PCB Placement (Front View)



Note Use Figure 7-6 for reference only. For detailed information about adapter module enclosure dimensions, refer to the electronic design files provided by NI.

Figure 7-7. Front Panel Dimensions and PCB Placement (Top View)



Note Use Figure 7-7 for reference only. For detailed information about adapter module enclosure dimensions, refer to the electronic design files provided by NI.

You can completely customize the front panel of the adapter module enclosure. For initial prototyping, the adapter module front panel is optional. For the final design, however, NI recommends using the front panel to reduce electromagnetic emissions. For detailed dimensions on the front panel, PCB placement, and the I/O connector window, refer to Figure 5-3, *I/O Connector Area Clearance Dimensions*.

The front panel enclosure is also useful for labeling the connectors for their intended purpose. Outside services are available to do custom machining. The generation of a simple piece of metal such as the adapter module front panel is inexpensive, and there are several online services that can perform this operation, such as `www.emachineshop.com` or `www.bigbluesaw.com`.

Suggested Labeling

Note If you want to design the adapter module to have the same aesthetic, color scheme, and style as existing FlexRIO adapter modules, contact NI for supplier information.

Figure 7-8. Front Panel Dimensions and Labeling

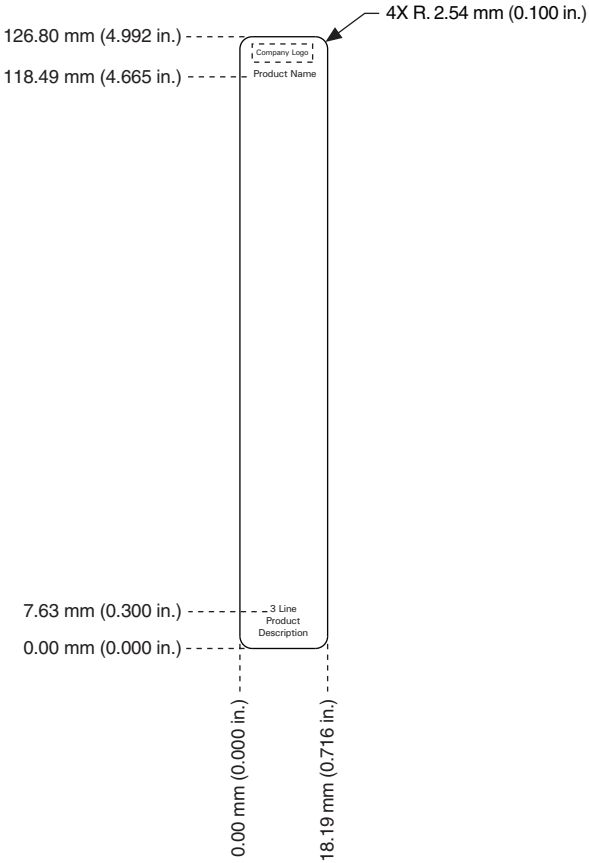
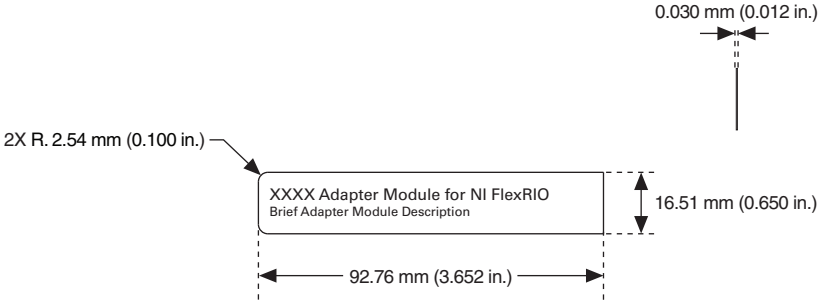


Figure 7-9. Primary Side Suggested Labeling and Dimensions



8

Installing the Adapter Module

This chapter explains how to install your custom adapter module with your Controller for FlexRIO or FlexRIO FPGA module.

Installing the Adapter Module with the FlexRIO FPGA Module

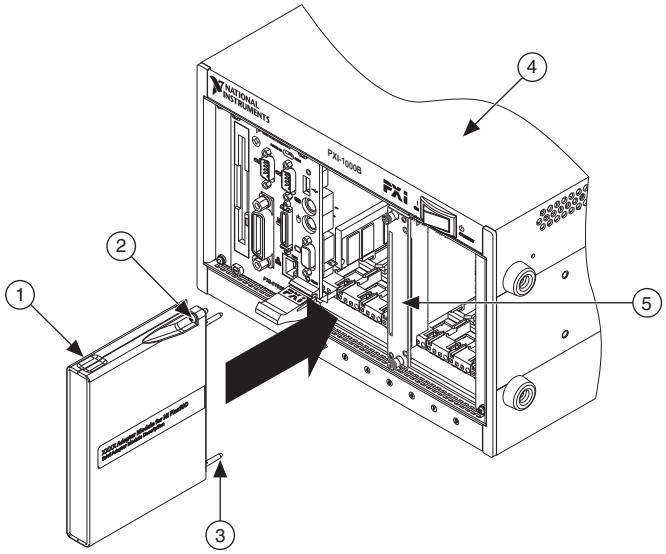
The following section contains steps for installing the adapter module with the FlexRIO FPGA module. For information about installing a Controller for FlexRIO and an adapter module, refer to the *Installing the Adapter Module with the Controller for FlexRIO* section.

Note You must install the FlexRIO FPGA module in a chassis before installing the adapter module. Refer to the getting started guide for your FlexRIO FPGA module for instructions about how to install the FlexRIO FPGA module.

Complete the following steps to connect the custom adapter module to the FlexRIO FPGA module.

1. Gently insert the guide pins and the high-density card edge of your adapter module into the corresponding connectors of the FlexRIO FPGA module, as shown in the following figure. The connection may be tight, but do not force the adapter module into place.

Figure 8-1. Installing the Adapter Module



1	Custom Adapter Module	4	PXI/PXI Express Chassis
2	Captive Screws	5	FlexRIO FPGA Module
3	Guide Pins		

2. Tighten the captive screws on the custom adapter module to secure it to the FlexRIO FPGA module. Torque the screws to 4.9 inch pounds using the #1 Phillips screwdriver included with the FPGA module shipping kit or another high quality screwdriver.
3. Launch LabVIEW to begin configuring your FlexRIO system.

Note MAX only recognizes FlexRIO FPGA modules in the chassis. Your adapter module does not appear in MAX.

Removing the Custom Adapter Module

To remove a custom adapter module from the FlexRIO FPGA device, you must disable power to the adapter module in LabVIEW FPGA. Refer to the *Adapter Module Removal Protocol* section for information about how to properly remove the adapter module.

Connectivity Options

Cabling options for your device are dependent on the connectors used in the adapter module design. Refer to `ni.com/products` to find cabling options appropriate for your application.

For detailed information about connecting I/O signals, refer to Chapter 3, *Interfacing Adapter Modules with NI 795xR and NI 796xR Modules* (NI 795*x*R/796*x*R), or Chapter 4, *Interfacing Adapter Modules with NI-793xR and NI 797xR Devices* (NI 797*x*R).

Installing the Adapter Module with the Controller for FlexRIO

The following section contains steps for installing the Controller for FlexRIO and an adapter module. For information about installing a FlexRIO FGPA module and an adapter module, refer to the *Installing the Adapter Module with the FlexRIO FPGA Module* section.

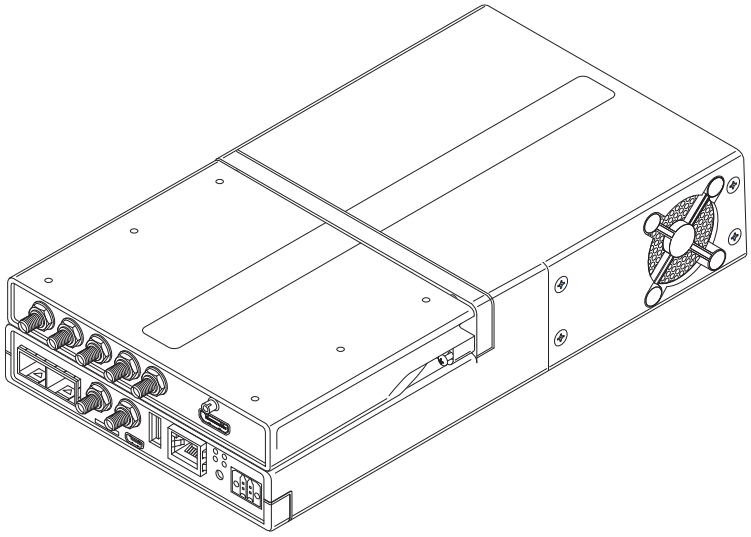
Note You must install the Controller for FlexRIO before installing the adapter module. Refer to the getting started guide for your Controller for FlexRIO for instructions about how to install the Controller for FlexRIO.

Complete the following steps to connect the custom adapter module to the Controller for FlexRIO.

1. Gently inser the guide pins and the high-density card edge of the FlexRIO adapter module into the corresponding connectors of the Controller for FlexRIO. The connection may be tight, but do not force the adapter module into place.
2. Tighten the captive screws on the FlexRIO adapter module to secure it to the Controller for FlexRIO.

The following figure shows the Controller for FlexRIO with the FlexRIO adapter module connected.

Figure 8-2. Controller for FlexRIO with FlexRIO Adapter Module



Removing the Custom Adapter Module

To remove a custom adapter module from the FlexRIO FPGA device, you must disable power to the adapter module in LabVIEW FPGA. Refer to the *Adapter Module Removal Protocol* section for information about how to properly remove the adapter module.

Connectivity Options

Cabling options for your device are dependent on the connectors used in the adapter module design. Refer to `ni.com/products` to find cabling options appropriate for your application.

For detailed information about connecting I/O signals, refer to Chapter 4, *Interfacing Adapter Modules with NI-793xR and NI 797xR Devices*.

9

Configuring Your Adapter Module for Use with NI 795xR/796xR Modules and LabVIEW FPGA

This chapter explain how to prepare your custom adapter module for use with NI 795*x*R/ NI 796*x*R devices and LabVIEW FPGA.

Refer to the following table for information about which compilers, constraints, and configuration files to use to develop your adapter module depending on the version of LabVIEW you are using.

Table 9-1. Recommended Files for Developing Adapter Modules

LabVIEW Version	Target/ Compiler	Logic	Constraints		Configuration File		LabVIEW/IP Interface
		.vhd	.ucf	.xdc	.tbc	.fam	.xml
2012 and earlier	Virtex-5 with ISE	X	X		X		X
2013	Virtex-5 with ISE	X	X		o	O	X
2014 and later	Virtex-5 with ISE	X	X		o	O	X
X = exclusive option O = either option (preferred) o = either option							

Note In software, FlexRIO adapter modules are referred to as *IO modules*.

To configure the adapter module for use with LabVIEW FPGA, complete the following steps:

1. Program the IO Module ID into the EEPROM.
2. Create the adapter module configuration file.
   a. If you are using FlexRIO Support 2012 SP1 or earlier, follow the instructions in the *Creating the Adapter Module Configuration (.tbc) File* section to create a `.tbc` file.
   b. If you are using FlexRIO Support 13.0 or later, follow the instructions in the *Creating the Adapter Module Configuration (.fam) File* section to create a `.fam` file.
3. Create or acquire the socketed component-level IP (CLIP) to create the interface between adapter module and the FPGA.
4. Configure the adapter module in a LabVIEW project.

This chapter describes steps 1 and 2 in greater detail. The proceeding sections explain how to define the module characteristics and establish the adapter module identification so that LabVIEW FPGA recognizes the adapter module. Chapter 11, *Creating Socketed Component-Level IP for Your Adapter Module and NI 795xR/796xR Modules*, describes steps 3 and 4 in more detail.

Programming the EEPROM

The FlexRIO device uses the IO Module ID to uniquely identify an adapter module that is physically connected to the FlexRIO FPGA module. The IO Module ID is a 32-bit number stored in the adapter module EEPROM that uniquely identifies a particular model of adapter module. All adapter modules of the same model must use the same IO Module ID, and different models of adapter modules cannot use the same IO Module ID.

Storing an IO Module ID on your adapter module EEPROM provides better electrical protection for both the adapter module and the FlexRIO FPGA module. For instance, the V_{ccoA}/V_{ccoB} levels must be configured properly to match the electrical requirements of each individual adapter module. Each FPGA VI is built for a specific adapter module, and therefore the FPGA VI has a concept of an expected adapter module that should use a particular configuration for V_{ccoA}/V_{ccoB}. When you insert an adapter module into a FlexRIO FPGA module, the FPGA module automatically attempts to detect which adapter module is connected. Use an IO Module ID to allow the FPGA module to detect whether the inserted adapter module matches the adapter module expected by LabVIEW FPGA. Adapter module power—including V_{ccoA}/V_{ccoB}—is enabled only if there is a match.

Note NI strongly recommends adding an EEPROM to your adapter module for identification purposes. This provides better electrical protection for both the adapter module and the FlexRIO FPGA module. It also improves the software configuration experience within LabVIEW FPGA. For more information about including an EEPROM in your adapter module design for identification purposes, refer to Chapter 3, *Interfacing Adapter Modules with NI 795xR and NI 796xR Modules*.

Note The FPGA parses all `.tbc` and `.fam` files. If the FPGA encounters an IO Module ID in a previously parsed `.tbc` or `.fam` file, the IO Module does not show up in the list.

The format of the 32-bit IO Module ID is as follows:

Bits<31..16>: Vendor ID

Bits<15..0>: Product ID

The Vendor ID is a 16-bit number which is unique for each adapter module manufacturer. Vendor IDs are assigned to individual manufacturers by NI. Each adapter module manufacturer should use their allotted Vendor ID for all adapter modules that they produce.

To obtain your adapter module Vendor ID, visit `ni.com/ask`, and create a new technical support request. For more information about obtaining your adapter module Vendor ID, refer to the *Registration* section of Chapter 1, *Before You Begin*.

The Product ID is a value that is chosen by the adapter module manufacturer. Each adapter module manufacturer should ensure that a unique Product ID is chosen for each type of adapter module.

The Vendor ID and Product ID combine to form the unique IO Module ID for each individual adapter module created by a manufacturer.

The following is an example IO Module ID:

IO Module ID: `0xFFFF0001`

Vendor ID: `0xFFFF`

Product ID: `0x0001`

Programming the EEPROM in LabVIEW

NI provides a LabVIEW host VI which you can use to program the IO Module ID into the adapter module. The VI library containing this VI is installed in the following location:

`<`*`LabVIEW`*`>\vi.lib\FlexRIO\FlexRIO_HostInterface.llb`

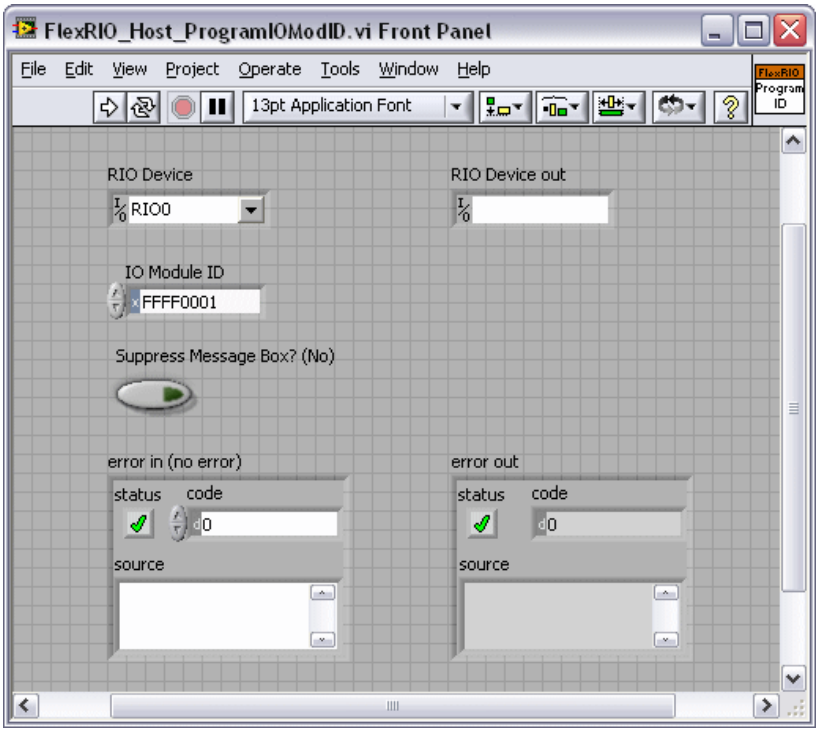
Complete the following steps to program the IO Module ID into your adapter module:

1. Launch LabVIEW and open the `FlexRIO_Host_ProgramIOModID.vi`.
2. On the front panel of the `FlexRIO_Host_ProgramIOModID.vi`, configure the **RIO Device** control to match the RIO device name of the FlexRIO FPGA module connected to your adapter module.

   Determine which RIO device name corresponds to your FlexRIO FPGA module by launching Measurement & Automation Explorer (MAX) and browsing to **Devices and Interfaces**. If you have only one RIO device in your system, the name for your device is typically `RIO0`.

3. On the front panel of `FlexRIO_Host_ProgramIOModID.vi`, configure the **IO Module ID** control to the correct 32-bit IO Module ID for your adapter module.

Figure 9-1. FlexRIO_Host_ProgramIOModID.vi Front Panel

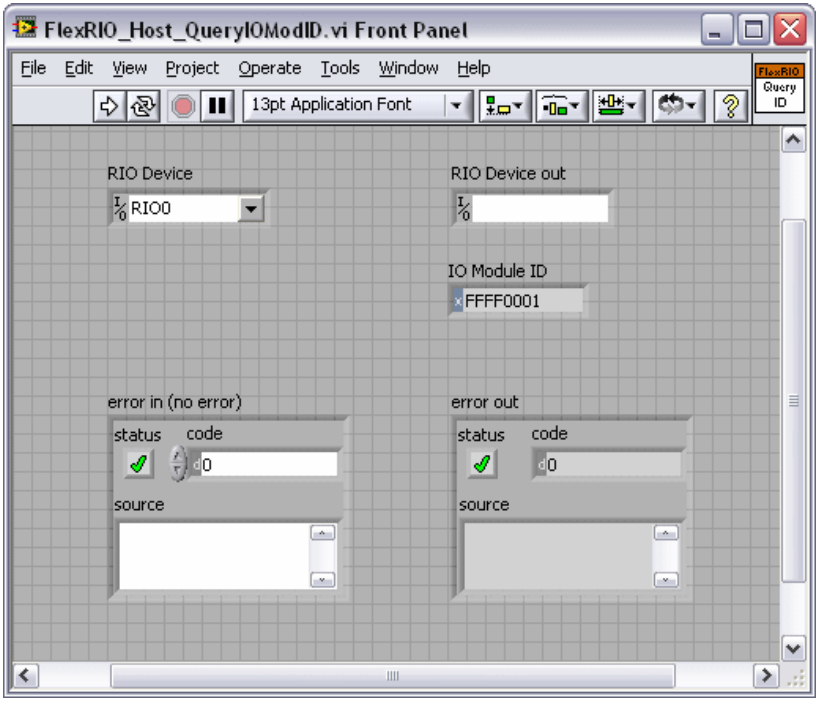


4. Run the VI. If the VI runs successfully without error, then your IO Module ID is programmed properly.

Complete the following steps to read the IO Module ID from the EEPROM of an attached adapter module:

1. Launch LabVIEW and open the `FlexRIO_Host_QueryIOModID.vi`.
2. On the front panel of the `FlexRIO_Host_QueryIOModID.vi`, configure the **RIO Device** control to match the RIO device name of the FlexRIO FPGA module connected to your adapter module. Determine which RIO device name corresponds to your FlexRIO FPGA module by launching MAX and browsing to **Devices and Interfaces**. If you have only one RIO device in your system, the name for your device is typically `RIO0`.
3. Run the VI. If the VI runs successfully without error, the IO Module ID indicator on the VI front panel returns the IO Module ID of the attached adapter module.

Figure 9-2. FlexRIO_Host_QueryIOMod.vi Front Panel



`FlexRIO_HostInterface.llb` contains additional VIs that you can use for reading and writing to and from arbitrary EEPROM locations. You can also use these VIs to issue raw I^2C bus cycles. For more information about these VIs, refer to the *FlexRIO Help*.

EEPROM Map

Table 9-2 describes the adapter module EEPROM map.

Table 9-2. EEPROM Map

Byte Address	Size (Bytes)	Field Name	Required?
0x0	2	Vendor ID	Yes
0x2	2	Product ID	Yes
0x4	4	Serial Number	No
0x8	24	Reserved	No
0x20	224	User Space	No

Adapter module manufacturers may optionally store a 32-bit serial number in the Serial Number EEPROM field.

The User Space occupies the remainder of the EEPROM. The adapter module manufacturer determines the layout and usage of this portion of the EEPROM. For example, the User Space can be used to store ADC calibration constants.

Access to the I^2C signals is shared with the adapter module socketed CLIP. This provides direct access to the EEPROM from the FPGA. For more information the EEPROM, refer to the *EEPROM Overview* section of Chapter 3, *Interfacing Adapter Modules with NI 795xR and NI 796xR Modules*. For additional information about I^2C signals, refer to Table 11-3, *I2C Core Interface Signals**.

Creating the Adapter Module Configuration (.tbc) File

Note This section is intended for users who are using FlexRIO Support 2012 SP1 or earlier. If you are using FlexRIO Support 13.0 or later, refer to the *Creating the Adapter Module Configuration (.fam) File* section to learn how to create a `.fam` adapter module configuration file.

To define certain electrical characteristics of the adapter module for LabVIEW, you need to create an adapter module configuration (`.tbc`) file. LabVIEW FPGA uses this information during the compilation process to properly configure the V_{ccoA}/V_{ccoB} voltage levels, to determine which IO Module ID to expect during the adapter module discovery sequence, and to properly configure the FPGA I/O standards for the GPIO signals on the adapter module connector interface.

The presence of an adapter module configuration file also allows your adapter module to be selectable in the adapter module configuration user interface within the LabVIEW project. All adapter module configuration files are automatically enumerated from the `IO Modules` directory on disk. The `IO Modules` directory is stored in the following location:

- **Windows XP**

  `C:\Documents and Settings\All Users\Shared Documents\National Instruments\FlexRIO\IO Modules`

- **Windows 8/7/Vista**

  `C:\Users\Public\Documents\National Instruments\FlexRIO\IO Modules`

NI recommends that you create a manufacturer-specific subfolder within the `IO Modules` directory and place your adapter module configuration files within this subfolder. Creating a manufacturer-specific folder for your `.tbc` files helps facilitate better organization of adapter module support files when you install adapter modules from multiple manufacturers on the same system. During device configuration and at VI build, the FlexRIO FPGA device support software automatically enumerates all files within the `IO Modules` directory inside subfolders of the directory.

Complete the following steps to create a `.tbc` file that defines your adapter module characteristics within LabVIEW.

1. Go to the FlexRIO `IO Modules` directory and create a manufacturer-specific folder that contains the `.tbc` file for the new adapter module.
2. Create your `.tbc` file.

   An example `.tbc` file is provided in the following location:

   - **Windows XP**

     ```
     C:\Documents and Settings\All Users\Shared Documents\
     National Instruments\FlexRIO\Module Development Kit\
     Examples\IO Module\ExampleIOModule.tbc
     ```

   - **Windows 8/7/Vista**

     ```
     C:\Users\Public\Documents\National Instruments\FlexRIO\
     Module Development Kit\Examples\IO Module\
     ExampleIOModule.tbc
     ```

   You can use this file as a template for your adapter module configuration file. If you use the NI example `.tbc` file, modify the required values described in the *Adapter Module Configuration (.tbc) Values* section and rename the file for your adapter module.
3. Save the `.tbc` file to the location you created in step 1.

Adapter Module Configuration (.tbc) Values

The `.tbc` file uses a standard INI-file syntax. The following sections contain detailed descriptions of each of the supported configuration values contained within the file. An example `.tbc` file is also provided.

General .tbc Values

Note The General section is required in your `.tbc` file.

The General section describes basic information about the adapter module. Table 9-3 lists the supported General configuration values.

Table 9-3. Supported General Configuration Values

Key Name	Data Type	Description
FormatVersion	Float	Specifies which version of the adapter module configuration file annotations are used to create the adapter module configuration file. Set this value to `1.0`. **Note**: If the `.tbc` contains the IoModSyncClock or IoModSyncClockSource keys, set this value to `1.1`.
Manufacturer	String	Specifies the name of the manufacturer that created the adapter module. This name displays in the IO Module Properties dialog box in the LabVIEW project with the adapter module. **Note**: The `Manufacturer.tbc` value cannot include the following characters: `, ; :` All `.tbc` files are enumerated in the IO Module Properties page. If the combination of the manufacturer and model name (for example, National Instruments NI 5761) is found in a previously parsed `.tbc` file, the Details window displays an error when you click the duplicate IO Module.
Model	String	Specifies the model name of the adapter module. This name displays in the LabVIEW project with the adapter module. **Note**: The `Model.tbc` value cannot include the following characters: `, ; :`
Description	String	Provides a general description of the capabilities and function of the adapter module. This information displays in the IO Module Properties dialog box in the LabVIEW project. You may use `\n` to format the string with end-of-line characters.

Table 9-3. Supported General Configuration Values (Continued)

Key Name	Data Type	Description
VccoALevel	Float	Specifies the voltage level (in volts) for the V_{ccoA} bank. The valid values are 1.2, 1.5, 1.8, 2.5, and 3.3. Values that include decimals must use the `.` character and not the `,` character. For example, 1.2V is valid, but 1,2V is not.
VccoBLevel	Float	Specifies the voltage level (in volts) for the V_{ccoB} bank. The valid values are 1.2, 1.5, 1.8, 2.5, and 3.3. Values that include decimals must use the `.` character and not the `,` character. For example, 1.2V is valid, but 1,2V is not.
IOModuleID	U32	Specifies the unique IO Module ID value that is stored in the identification EEPROM. Set this key to the value of the IO Module ID that you created in the *Programming the EEPROM* section. If your adapter module does not contain an EEPROM for identification purposes, remove this key from the adapter module configuration file.
DefaultCLIP	String	Specifies the name of the default CLIP implementation for the adapter module. This name is specified within the `<CLIPDeclaration>` tag in the CLIP XML file. When creating a new LabVIEW project, the FPGA device is auto-discovered with an adapter module inserted, and this CLIP is automatically selected for use with the adapter module. Refer to Chapter 11, *Creating Socketed Component-Level IP for Your Adapter Module and NI 795xR/796xR Modules*, for more information about CLIP.
(Optional) IoModSyncClock	String	Specifies how to support the clock for NI 796*x*R devices. If this key name is not present, the adapter module clock is not supported.
(Optional) IoModSyncClockSource	String	Specifies the default IoModSyncClock source for NI 796*x*R devices. Use this tag only when the IoModSyncClock is set to SUPPORTED or REQUIRED. If this key name is not present, PXI_CLK10 is selected by default.

Adapter Module IOModuleID

The adapter module `IOModuleID` key determines the IO Module ID for your adapter module. The `IOModuleID` value in the adapter module configuration file should be in hexadecimal format, prefixed with `0x`.

This example demonstrates how to specify the `IOModuleID` value in the adapter module configuration file:

```
IOModuleID=0xFFFF0001
```

In this example, `0xFFFF` is the Vendor ID, and `0x0001` is the Product ID.

Note If the adapter module you are configuring does not contain an EEPROM, you must remove the `IOModuleID` value from the `.tbc` file.

V_{cco} Levels

The adapter module configuration file specifies the V_{ccoA}/V_{ccoB} bank values. These values are fixed by the adapter module designer and cannot be changed at run-time within LabVIEW FPGA. The available voltage options for each V_{cco} bank are 1.2 V, 1.5 V, 1.8 V, 2.5 V, and 3.3 V.

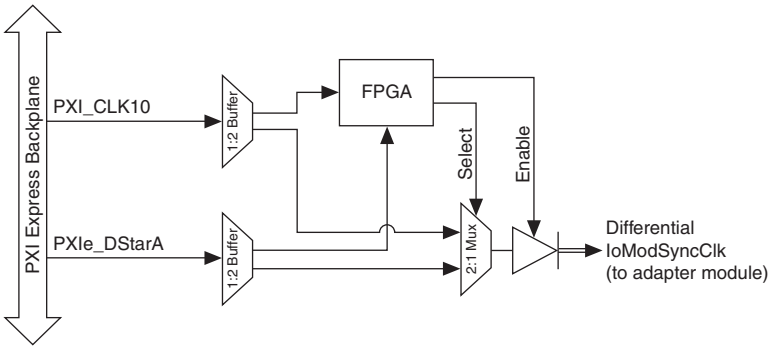
Note Values that include decimals must use the `.` character and not the `,` character. For example, 1.2V is valid, but 1,2V is not.

Caution The selected V_{ccoA}/V_{ccoB} value must match the electrical design of the corresponding adapter module. Failure to do so risks damaging the adapter module or the FlexRIO FPGA module. NI is *not* liable for any damage resulting from such misuse.

IoModSyncClk.tbc Values (NI 796xR Only)

The lexRIO PXI Express FPGA module provides the IoModSyncClk signal for synchronization between adapter modules. IoModSyncClk is a LVPECL clock that can be sourced either by PXI_CLK10 or by PXIe_DStarA, as shown in the following figure.

Figure 9-3. IoModSyncClk Source Block Diagram



For more information about the IoModSyncClk and how to use it in your adapter module design, refer to the *IoModSyncClk (NI 796xR Only)* section of Chapter 3, *Configuring Your Adapter Module for Use with NI 795xR/796xR Modules and LabVIEW FPGA*. Refer to the *NI PXI-795xR Base Clocks* topic or the *NI PXIe-796xR Base Clocks* topic in the *FlexRIO Help* book in the *LabVIEW Help* for more information about clock resources.

Enabling IoModSyncClk

You can enable IoModSyncClk by including optional keys within the **General** section of the `.tbc` file of the corresponding adapter module. The two optional keys are as follows:

Table 9-4. Optional Keys for Enabling IoModSyncClock

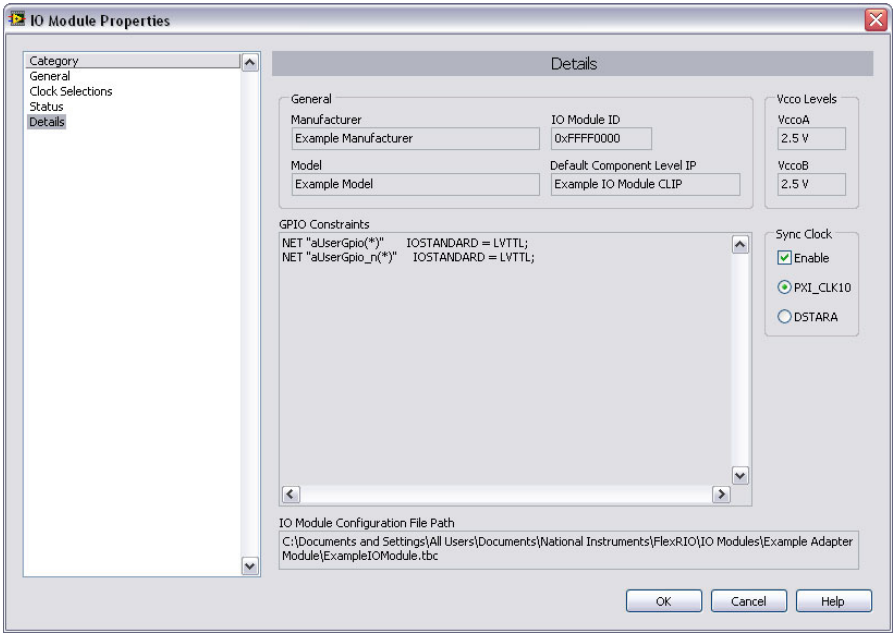
Key Name	Defined Values	Description
1. (Optional) IoModSyncClock— Specifies how to support the clock. If this key name is not present, then the adapter module clock is not supported.	UNSUPPORTED	IoModSyncClk is tristated.
	SUPPORTED	IoModSyncClk can be enabled or tristated from the **Sync Clock** section of the **IO Module Properties** menu. The default is enabled.
	REQUIRED	IoModSyncClock is always enabled.
	PXICLK10REQUIRED	IoModSyncClk is always enabled and is sourced by PXI_CLK10.
	DSTARAREQUIRED	IoModSyncClock is always enabled and is sourced by PXIe_DStarA.
2. (Optional) IoModSyncClockSource— Specifies the default clock source. IoModSyncClockSource is only used when IoModSyncClock is set to SUPPORTED or REQUIRED. If IoModSyncClockSource is not present, the default clock source is PXI_CLK10.	CLK10	IoModSyncClk is sourced by PXI_CLK10.
	DSTARA	IoModSyncClk is sourced by PXIe_DStarA.

When the IoModSyncClock key is set to SUPPORTED or REQUIRED, you can also select settings for the IoModSyncClk in the LabVIEW Project. To access this menu, right-click the **IO Module** item under the FPGA Target in the Project Explorer window and select **Properties** to display the **IO Module Properties** dialog box. The Sync Clock section in the **Details** category controls the IoModSyncClk settings.

When the IoModSyncClockSource key is set to SUPPORTED or REQUIRED in the `.tbc` file, the default IoModSyncClk source is selected. The value selected under the Details category in

the IO Module Properties dialog box overrides the default IoModSyncClockSource `.tbc` file key value.

Figure 9-4. IO Module Properties Sync Clock Enabled



At compile time, the FPGA selects the source for the IoModSyncClk, as well as the enabled or disabled state. To change these settings, you must recompile the FPGA VI.

Compatibility with the NI 795*x*R

An adapter module is compatible with an NI 795*x*R FPGA module if the corresponding `.tbc` file does not contain the IoModSyncClock key or if the IoModSyncClock key value is present and set to UNSUPPORTED or SUPPORTED. All other values result in an error that states that the adapter module is not supported.

Constraints .tbc Values

The Constraints section is required. The Constraints section specifies FPGA compilation constraints information for the FPGA pins dedicated to the adapter module interface. The syntax of data in this section is identical to the syntax of raw UCF data used by the Xilinx ISE tool. The constraints data is concatenated to the `.ucf` file that is used in the FPGA compilation.

Use the Constraints section to properly constrain the electrical I/O standard that is used for each adapter module general-purpose I/O (GPIO) pin. These constraints depend on the V_{ccoA}/V_{ccoB} settings, which are also specified in the `.tbc` file. Like the V_{ccoA}/V_{ccoB} levels, the Constraints

values should be fixed by the adapter module designer and cannot be changed at run-time from within LabVIEW FPGA. Include all physical-interface related constraints—such as I/O standard, drive strength, and termination—in the Constraints section.

Caution You must set the I/O standard constraint for the UserGPIO pins. If you do not constrain these FPGA I/O pins, the compiler assigns a default I/O standard, which could prevent proper I/O function and possibly damage the FlexRIO FPGA module or adapter module.

FlexRIO UserGpio pins may be set to any Xilinx I/O standard with the following conditions:

- The V_{ccoA}/V_{ccoB} rail is set to the required V_{cco} value for this I/O standard.
- The I/O standard does not require a reference voltage (V_{ref}).
- If DCI is enabled, the I/O standard must allow a 50 Ω resistor connected to VRN/VRP.

Table 9-5 lists the Xilinx I/O standards supported by the FlexRIO GPIO and V_{ccoA}/V_{ccoB} power rails.

Table 9-5. FlexRIO Supported Xilinx I/O Standards

I/O Standard	V_{cco}
LVTTL	3.3 V
LVCMOS*xx*[*]	1.2 V, 1.5 V, 1.8 V, 2.5 V, and 3.3 V
LVDCI*xx* (DCI R = 50 Ω)[*]	1.5 V, 1.8 V, 2.5 V, and 3.3 V
LVDS_25 (inputs use internal 100 Ω differential parallel termination)	2.5 V
[*] Where *xx* is determined by the selected voltage.	

Note The I/O standards listed in Table 9-5 are determined by Xilinx specifications and are subject to change. To determine the most current I/O standard constraints for your FlexRIO device, refer to the Xilinx documentation, available at `www.xilinx.com`. For a list of Xilinx documents applicable to FlexRIO applications, refer to Appendix C, *Xilinx Documentation References*.

Refer to the *NI 795xR/796xR FPGA I/O Bank Voltages* section of Chapter 3, *Configuring Your Adapter Module for Use with NI 795xR/796xR Modules and LabVIEW FPGA*, for more information about FPGA I/O bank voltages.

For example, the following Constraints section constrains GPIO pair 20 to use the LVTTL I/O standard:

```
[Constraints]
NET "aUserGpio(20)"          IOSTANDARD = LVTTL;
NET "aUserGpio_n(20)"        IOSTANDARD = LVTTL;
```

The next example Constraints section constrains GPIO pair 20 to the LVDCI_33 I/O standard, the LVTTL GClk Input to LVTTL, and the LVDS GClk input to LVDS_25. This example also enables the 100 Ω differential termination on the LVDSGClk input.

```
[Constraints]
NET "aUserGpio(20)"          IOSTANDARD = LVDCI_33;
NET "aUserGpio_n(20)"        IOSTANDARD = LVDCI_33;
NET "UserGClkLvttl"          IOSTANDARD = LVTTL;
NET "UserGClkLvds"           IOSTANDARD = LVDS_25;
NET "UserGClkLvds_n"         IOSTANDARD = LVDS_25;
INST "*IBUFGDSx*"            DIFF_TERM = TRUE;
```

Note 100 Ω differential termination is placed on the LVDSGClk global clock buffer.

NI recommends that you constrain all GPIO lines even if they are not used. The following snippet demonstrates what the unused constraint portion of your Constraints section may look like. For example, if GPIO lines 40 to 60 are unused, explicitly assign them all to LVCMOS*xx,* where *xx* is determined by the selected voltage for that bank.

```
# Unused GPIO pins
NET "aUserGpio(40)"          IOSTANDARD = LVCMOS25;
NET "aUserGpio_n(40)"        IOSTANDARD = LVCMOS25;
...
NET "aUserGpio(60)"          IOSTANDARD = LVCMOS25;
NET "aUserGpio_n(60)"        IOSTANDARD = LVCMOS25;
```

NI recommends that you assign GPIO lines one at a time. Although wildcards are allowed in the Xilinx constraint syntax, best performance comes from individual line assignment.

Note To ensure the correct compilation of your adapter module constraints information, NI recommends using `()` as the bus delimiters in both the `.ucf` and the `.tbc` files, instead of using <>. For example: `aUserGpio(*)`.

Example

The following is an example of a properly constructed `.tbc` file. Refer to the *Creating or Acquiring the IP for the FlexRIO Adapter Module* section of Chapter 11, *Creating Socketed Component-Level IP for Your Adapter Module and NI 795xR/796xR Modules*, for information about complete example files that ship with the FlexRIO Adapter Module Development Kit.

```
[General]
FormatVersion=1.1
Manufacturer=Example Manufacturer
Model=Example Model
Description=This is an example adapter module configuration file.
VccoALevel=3.3
VccoBLevel=3.3
IOModuleID=0xFFFF0001
DefaultCLIP=ExampleIOModuleCLIP
IoModSyncClock=SUPPORTED
IoModSyncClockSource=CLK10

[Constraints]
NET "aUserGpio(1)"              IOSTANDARD = LVCMOS25;
NET "aUserGpio_n(1)"            IOSTANDARD = LVCMOS25;
NET "aUserGpio(2)"              IOSTANDARD = LVCMOS25;
NET "aUserGpio_n(2)"            IOSTANDARD = LVCMOS25;
...
NET "aUserGpio(20)"             IOSTANDARD = LVCMOS25;
NET "aUserGpio_n(20)"           IOSTANDARD = LVCMOS25;
```

Note IoModSyncClock and IoModSyncClockSource `.tbc` file keys are supported by the NI 796*x*R only.

Creating the Adapter Module Configuration (.fam) File

Note This section is intended for users who are using FlexRIO Support 13.0 or later. If you are using FlexRIO Support 2012 or earlier, refer to the *Creating the Adapter Module Configuration (.tbc) File* section to learn how to create a `.tbc` adapter module configuration file.

To define certain electrical characteristics of the adapter module for LabVIEW, you need to create an adapter module configuration (`.fam`) file. LabVIEW FPGA uses this information during the compilation process to properly configure the V_{ccoA}/V_{ccoB} voltage levels to determine which IO Module ID to expect during the adapter module discovery sequence, and to properly configure the FPGA I/O standards for the GPIO signals on the adapter module connector interface.

The presence of an adapter module configuration file also allows your adapter module to be selectable in the adapter module configuration user interface within the LabVIEW project. All adapter module configuration files are automatically enumerated from the `IO Modules` directory on disk. The `IO Modules` directory is stored in the following location:

- **Windows XP**

  `C:\Documents and Settings\All Users\Shared Documents\National Instruments\FlexRIO\IO Modules`

- **Windows 8/7/Vista**

  `C:\Users\Public\Documents\National Instruments\FlexRIO\IO Modules`

NI recommends that you create a manufacturer-specific subfolder within the `IO Modules` directory and place your adapter module configuration files within this subfolder. Creating a manufacturer-specific folder for your `.fam` files helps facilitate better organization of adapter module support files when you install adapter modules from multiple manufacturers on the same system. During device configuration and at VI build, the FlexRIO FPGA device support software automatically enumerates all files within the `IO Modules` directory inside subfolders of the directory.

Complete the following steps to create a `.fam` file that defines your adapter module characteristics within LabVIEW.

1. Go to the FlexRIO `IO Modules` directory and create a manufacturer-specific folder that contains the `.fam` file for the new adapter module.
2. Create your `.fam` file.

   An example `.fam` file is provided in the following location:

   - **Windows XP**

     `C:\Documents and Settings\All Users\Shared Documents\National Instruments\FlexRIO\Module Development Kit\Examples\IO Module\ExampleIOModule.fam`

   - **Windows 8/7/Vista**

     `C:\Users\Public\Documents\National Instruments\FlexRIO\Module Development Kit\Examples\IO Module\ExampleIOModule.fam`

   You can use this file as a template for your adapter module configuration file. If you use the NI example `.fam` file, make sure to modify the required values described in the *Adapter Module Configuration (.fam) Values* section and rename the file for your adapter module.
3. Save the `.fam` file to the location you created in step 1.

Adapter Module Configuration (.fam) Values

The adapter module configuration (`.fam`) file uses a standard INI-file syntax. The following sections contain detailed descriptions of each of the supported configuration values contained within the file. An example `.fam` file is also provided. A `.fam` file consists of a Common section and a Socket-specific section.

Common .fam Values

Note The Common section is required in your `.fam` file.

The Common section describes basic information about the adapter module. Table 9-6 lists the supported Common configuration values.

Table 9-6. Supported Common Configuration Values

Key Name	Data Type	Description
FormatVersion	Integer	Specifies which version of the adapter module configuration file annotations are used to create the adapter module configuration (`.fam`) file. Set this value to `1`.
OldestCompatibleFormatVersion	Integer	Specifies which version of the adapter module configuration file annotations that the adapter module configuration (`.fam`) file is compatible with. Set this value to `1`.
Manufacturer	String	Specifies the name of the manufacturer that created the adapter module. This name displays in the IO Module Properties dialog box in the LabVIEW project with the adapter module. **Note**: The `Manufacturer .fam` value cannot include the following characters: `, ; :`

Table 9-6. Supported Common Configuration Values (Continued)

Key Name	Data Type	Description
Model	String	Specifies the model name of the adapter module. This name displays in the IO Modules Properties dialog box in the LabVIEW project when the adapter module is configured. **Note**: The `Model .fam` value cannot include the following characters: `, ; :` All `.fam` files are enumerated in the IO Module Properties page. If the combination of the manufacturer and model name (for example, National Instruments NI 5761) is found in a previously parsed `.fam` file, the Details window displays an error when you click the duplicate IO Module.
Description	String	Provides a general description of the capabilities and function of the adapter module. This information displays in the IO Module Properties dialog box in the LabVIEW project. You may use `\n` to format the string with end-of-line characters.
IOModuleID	U32	Specifies the unique IO Module ID value that is stored in the identification EEPROM. Set this key to the value of the IO Module ID that you created in the *Programming the EEPROM* section. If your adapter module does not contain an EEPROM for identification purposes, remove this key from the adapter module configuration file.
CompatibleCLIPSockets	String	Specifies which FPGA families are supported. If you are using the NI 795*x*R or the NI 796*x*R, enter `FlexRIO-IOModule` for this value. If you are using the NI 797*x*R, enter `FlexRIO-K7IOModule` for this value.

Socket-specific .fam Values

The socket-specific section of the `.fam` file describes information that is specific to your device's CLIP socket. Table 9-7 lists the supported configuration values for the NI 795*x*R and NI 796*x*R device sockets.

Note Name the socket-specific section `FlexRIO-IOModule`. Socket-specific sections are required for each socket family listed in the CompatibleCLIPSockets tag.

Table 9-7. Supported Socket-specific Configuration Values

Key Name	Data Type	Description
Default CLIP	String	Specifies the name of the default CLIP implementation for the adapter module. This name is specified within the `<CLIPDeclaration>` tag in the CLIP XML file. When you create a new LabVIEW project, the FPGA device is auto-discovered with an adapter module inserted, and this CLIP is automatically selected for use with the adapter module. Refer to Chapter 11, *Creating Socketed Component-Level IP for Your Adapter Module and NI 795xR/796xR Modules*, for more information about CLIP.
VccoALevel	Float	Specifies the voltage level (in volts) to configure the V_{ccoA} bank for. The default value is 2.5. Valid values are: 1.2, 1.5, 1.8, 2.5, and 3.3. Values that include decimals must use the `.` character and not the `,` character. For example, 1.2V is valid, but 1,2V is not.
VccoBLevel	Float	Specifies the voltage level (in volts) to configure the V_{ccoB} bank for. The default value is 2.5. Valid values are: 1.2, 1.5, 1.8, 2.5, and 3.3. Values that include decimals must use the `.` character and not the `,` character. For example, 1.2V is valid, but 1,2V is not.

Table 9-7. Supported Socket-specific Configuration Values (Continued)

Key Name	Data Type	Description
(Optional) IoModSyncClock Refer to Table 9-8 for IoModSyncClock defined values and descriptions.	String	Specifies how to support the clock for NI 795*x*R and NI 796*x*R devices. If this key name is not present, the adapter module clock is not supported.
(Optional) IoModSyncClockSource Refer to Table 9-8 for IoModSyncClockSource defined values and descriptions.	String	Specifies the default IoModSyncClock source for NI 795*x*R and NI 796*x*R devices. Use this tag only when the IoModSyncClock is set to SUPPORTED or REQUIRED. If this key name is not present, PXI_CLK10 is selected by default.

V_{ccoA}/V_{ccoB} Levels

The adapter module configuration file specifies the V_{ccoA}/V_{ccoB} bank values. These values are fixed by the adapter module designer and cannot be changed at run-time within LabVIEW FPGA. The available voltage options for each V_{cco} bank are 1.2 V, 1.5 V, 1.8 V, 2.5 V, and 3.3 V.

Note Values that include decimals must use the `.` character and not the `,` character. For example, 1.2V is valid, but 1,2V is not.

Caution The selected V_{ccoA}/V_{ccoB} value must match the electrical design of the corresponding adapter module. Failure to do so risks damaging the adapter module or the FlexRIO FPGA module. NI is *not* liable for any damage resulting from such misuse.

Adapter Module IOModuleID

The adapter module `IOModuleID` key determines the IO Module ID for your adapter module. The `IOModuleID` value in the adapter module configuration file must be in hexadecimal format, prefixed with `0x`.

This example demonstrates how to specify the `IOModuleID` value in the adapter module configuration file:

```
IOModuleID=0xFFFF0001
```

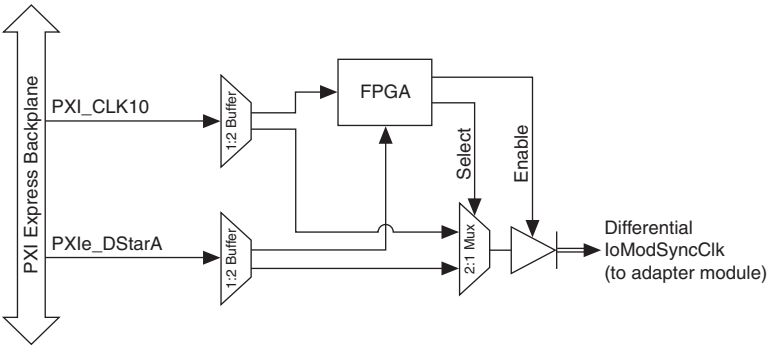
In this example, `0xFFFF` is the Vendor ID, and `0x0001` is the Product ID.

Note If the adapter module you are configuring does not contain an EEPROM, you must remove the `IOModuleID` value from the `.fam` file.

IoModSyncClk.fam Values (NI 796xR Only)

The FlexRIO PXI Express FPGA module provides the IoModSyncClk signal for synchronization between adapter modules. IoModSyncClk is a LVPECL clock that can be sourced either by PXI_CLK10 or by PXIe_DStarA, as shown in Figure 9-5.

Figure 9-5. IoModSyncClk Source Block Diagram



For more information about the IoModSyncClk and how to use it in your adapter module design, refer to the *IoModSyncClk (NI 796xR Only)* section of Chapter 3, *Interfacing Adapter Modules with NI 795xR and NI 796xR Modules*. Refer to the *NI PXIe-796xR Base Clocks* topic in the *FlexRIO Help* book in the *LabVIEW Help* for more information about clock resources.

Enabling IoModSyncClk

You can enable IoModSyncClk by including optional keys within the **Socket-Specific** section of the `.fam` file of the corresponding adapter module. The two optional keys are as follows:

Table 9-8. Optional Keys for Enabling IoModSyncClock

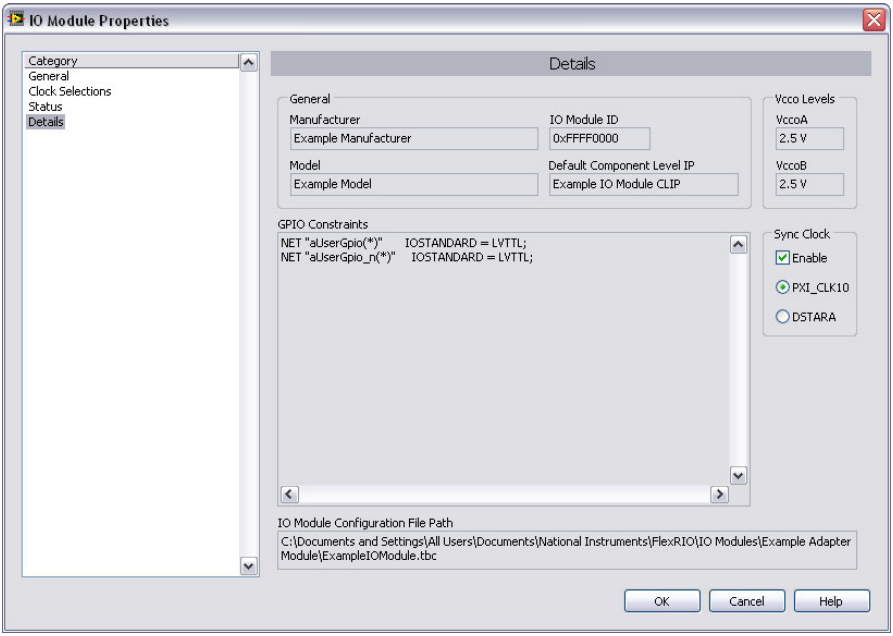
Key Name	Defined Values	Description
1. IoModSyncClock (Optional)—Specifies how to support the clock. If this key name is not present, then the adapter module clock is not supported.	UNSUPPORTED	IoModSyncClk is tristated.
	SUPPORTED	IoModSyncClk can be enabled or tristated from the **Sync Clock** section of the **IO Module Properties** menu. The default selection is enabled.
	REQUIRED	IoModSyncClock is always enabled.
	PXICLK10REQUIRED	IoModSyncClk is always enabled and is sourced by PXI_CLK10.
	DSTARAREQUIRED	IoModSyncClock is always enabled and is sourced by PXIe_DStarA.
2. IoModSyncClockSource (Optional)—Specifies the default clock source. IoModSyncClockSource is only used when IoModSyncClock is set to SUPPORTED or REQUIRED. If IoModSyncClockSource is not present, the default clock source is PXI_CLK10.	CLK10	IoModSyncClk is sourced by PXI_CLK10.
	DSTARA	IoModSyncClk is sourced by PXIe_DStarA.

When the IoModSyncClock key is set to SUPPORTED or REQUIRED, you can also select settings for the IoModSyncClk in the LabVIEW Project. To access this menu, right-click the **IO Module** item under the FPGA Target in the Project Explorer window and select **Properties** to display the **IO Module Properties** dialog box. The Sync Clock section in the **Details** category controls the IoModSyncClk settings.

When the IoModSyncClockSource key is set to SUPPORTED or REQUIRED in the `.fam` file, the default IoModSyncClk source is selected. The value selected under the Details category in

the IO Module Properties dialog box overrides the default IoModSyncClockSource `.fam` file key value.

Figure 9-6. IO Module Properties Sync Clock Enabled



At compile time, the FPGA selects the source for the IoModSyncClk, as well as the enabled or disabled state. To change these settings, you must recompile the FPGA VI.

FlexRIO-IOModule Constraints .fam Values

Note The FlexRIO-IOModule Constraints section is required in your `.fam` file.

The FlexRIO-IOModule Constraints section specifies FPGA compilation constraints information for the FPGA pins dedicated to the adapter module interface. The syntax of data in this section is identical to the syntax of raw `.ucf` data used by the Xilinx ISE tool. The constraints data is concatenated to the `.ucf` file that is used in the FPGA compilation.

Use the FlexRIO-IOModule Constraints section to properly constrain the electrical I/O standard that is used for each adapter module general-purpose I/O (GPIO) pin. These constraints depend on the V_{ccoA}/V_{ccoB}settings, which are also specified in the `.fam` file. Like the V_{ccoA}/V_{ccoB} levels, the FlexRIO-IOModule values should be fixed by the adapter module designer and cannot be changed at run-time from within LabVIEW FPGA. Include all physical-interface related constraints—such as I/O standard, drive strength, and termination—in the FlexRIO-IOModule Constraints section.

Caution You must set the I/O standard constraint for the UserGPIO pins. If you do not constrain these FPGA I/O pins, the compiler assigns a default I/O standard, which could prevent proper I/O function and possibly damage the FlexRIO FPGA module or adapter module.

FlexRIO UserGpio pins may be set to any Xilinx I/O standard with the following conditions:

- The V_{ccoA}/V_{ccoB} rail is set to the required V_{cco} value for this I/O standard.
- The I/O standard does not require a reference voltage (V_{ref}).
- If DCI is enabled, the I/O standard must allow a 50 Ω resistor connected to VRN/VRP.

Table 9-9 lists the Xilinx I/O standards supported by the FlexRIO GPIO and $V_{cco}A/V_{cco}B$ power rails.

Table 9-9. FlexRIO Supported Xilinx I/O Standards

I/O Standard	V_{cco}
LVTTL	3.3 V
LVCMOS*xx*[*]	1.2 V, 1.5 V, 1.8 V, 2.5 V, and 3.3 V
LVDCI*xx* (DCI R = 50 Ω)[*]	1.5 V, 1.8 V, 2.5 V, and 3.3 V
LVDS_25 (inputs use internal 100 Ω differential parallel termination)	2.5 V
[*] Where *xx* is determined by the selected voltage.	

Note The I/O standards listed in Table 9-10 are determined by Xilinx specifications and are subject to change. To determine the most current I/O standard constraints for your FlexRIO device, refer to the Xilinx documentation, available at `www.xilinx.com`. For a list of Xilinx documents applicable to FlexRIO applications, refer to Appendix C, *Xilinx Documentation References*.

Refer to the *NI 795xR/796xR FPGA I/O Bank Voltages* section of Chapter 3, *Interfacing Adapter Modules with NI 795xR and NI 796xR Modules*, for more information about FPGA I/O bank voltages.

For example, the following Constraints section constrains adapter module GPIO pair 20 to use the LVTTL I/O standard:

```
[FlexRIO-IOModule Constraints]
NET "aUserGpio(20)"IOSTANDARD = LVTTL;
NET "aUserGpio_n(20)"IOSTANDARD = LVTTL;
```

The next example Constraints section constrains GPIO pair 20 to the LVDCI_33 I/O standard, the LVTTL GClk Input to LVTTL, and the LVDS GClk input to LVDS_25. This example also enables the 100 Ω differential termination on the LVDSGClk input.

```
[FlexRIO-IOModule Constraints]
NET "aUserGpio(20)"          IOSTANDARD = LVDCI_33;
NET "aUserGpio_n(20)"        IOSTANDARD = LVDCI_33;
NET "UserGClkLvttl"          IOSTANDARD = LVTTL;
NET "UserGClkLvds"           IOSTANDARD = LVDS_25;
NET "UserGClkLvds_n"         IOSTANDARD = LVDS_25;
INST "*IBUFGDSx*"            DIFF_TERM = TRUE;
```

Note 100 Ω differential termination is placed on the LVDSGClk global clock buffer.

NI recommends that you constrain all GPIO lines even if they are not used. The following snippet demonstrates what the unused constraint portion of your FlexRIO-IOModule Constraints section may look like. For example, if GPIO lines 40 to 60 are unused, explicitly assign them all to LVCMOS25.

```
# Unused GPIO pins
NET "aUserGpio(40)"          IOSTANDARD = LVCMOS25;
NET "aUserGpio_n(40)"        IOSTANDARD = LVCMOS25;
...
NET "aUserGpio(60)"          IOSTANDARD = LVCMOS25;
NET "aUserGpio_n(60)"        IOSTANDARD = LVCMOS25;
```

Note To ensure the correct compilation of your adapter module constraints information, NI recommends using `()` as the bus delimiters in both the `.ucf` and the `.fam` files, instead of using <>. For example: `aUserGpio(20)`.

Example .fam File

The following is an example of a properly constructed `.fam` file for the NI 795*x*R and NI 796*x*R devices. Refer to the *Creating or Acquiring the IP for the FlexRIO Adapter Module* section of Chapter 11, *Creating Socketed Component-Level IP for Your Adapter Module and NI 795xR/796xR Modules*, for information about complete example files that ship with the FlexRIO Adapter Module Development Kit.

```
[Common]
FormatVersion=1
OldestCompatibleVersion=1
Manufacturer=Example Manufacturer
Model=Example Model
Description=This is an example adapter module configuration file.
IOModuleID=0xFFFF0001
CompatibleCLIPSockets=FlexRIO-IOModule

[FlexRIO-IOModule]
DefaultCLIP=ExampleIOModuleCLIP
IoModSyncClock=SUPPORTED
IoModSyncClockSource=CLK10
VccoALevel=2.5
VccoBLevel=2.5

[FlexRIO-IOModule Constraints]
NET "aUserGpio(1)"              IOSTANDARD = LVCMOS25;
NET "aUserGpio_n(1)"            IOSTANDARD = LVCMOS25;
NET "aUserGpio(2)"              IOSTANDARD = LVCMOS25;
NET "aUserGpio_n(2)"            IOSTANDARD = LVCMOS25;
...
NET "aUserGpio(20)"             IOSTANDARD = LVCMOS25;
NET "aUserGpio_n(20)"           IOSTANDARD = LVCMOS25;
```

Note IoModSyncClock and IoModSyncClockSource `.fam` file keys are supported by the NI 796*x*R only.

10

Configuring Your Adapter Module for Use with NI-793xR/797xR Modules and LabVIEW FPGA

This chapter explain how to prepare your custom adapter module for use with NI-793*x*R/ NI 797*x*R devices and LabVIEW FPGA.

Refer to the following table for information about which compilers, constraints, and configuration files to use to develop your adapter module depending on the version of LabVIEW you are using.

Table 10-1. Recommended Files for Developing Adapter Modules

LabVIEW Version	Target/ Compiler	Logic	Constraints		Configuration File		LabVIEW Interface
		.vhd	.ucf	.xdc	.tbc	.fam	.xml
2013	K7 with ISE*	X	X			X	X
2014 and later	K7 with Vivado	X		X		X	X
* You can target only the NI 7975R in ISE in LabVIEW 2013. All other NI-793*x*R and NI 797*x*R targets require the filesystem associated with LabVIEW 2014 or later.							

Note In software, FlexRIO adapter modules are referred to as *IO modules*.

To configure the adapter module for use with LabVIEW FPGA, complete the following steps:

1. Program the IO Module ID into the EEPROM.
2. Create the adapter module configuration file.
3. Create or acquire the socketed component-level IP (CLIP) to create the interface between adapter module and the FPGA.
4. Configure the adapter module in a LabVIEW project.

This chapter describes steps 1 and 2 in greater detail. The proceeding sections explain how to define the module characteristics and establish the adapter module identification so that

LabVIEW FPGA recognizes the adapter module. Chapter 12, *Creating Socketed Component-Level IP for Your Adapter Module and NI-793xR/NI 797xR Modules*, describes steps 3 and 4 in more detail.

Programming the EEPROM

The FlexRIO device uses the IO Module ID to uniquely identify an adapter module that is physically connected to the FlexRIO FPGA module. The IO Module ID is a 32-bit number stored in the adapter module EEPROM that uniquely identifies a particular model of adapter module. All adapter modules of the same model must use the same IO Module ID, and different models of adapter modules cannot use the same IO Module ID.

Storing an IO Module ID on your adapter module EEPROM provides better electrical protection for both the adapter module and the NI-793*x*R/NI 797*x*R module. For instance, the V_{cco} levels must be configured properly to match the electrical requirements of each individual adapter module. Each FPGA VI is built for a specific adapter module, and therefore the FPGA VI has a concept of an expected adapter module that should use a particular configuration for V_{ccoA}. When you insert an adapter module into an NI-793*x*R/NI 797*x*R, the NI-793*x*R/NI 797*x*R automatically attempts to detect which adapter module is connected. Use an IO Module ID to allow the NI-793*x*R/NI 797*x*R to detect whether the inserted adapter module matches the adapter module expected by LabVIEW FPGA. Adapter module power—including V_{cco}—is enabled only if there is a match.

Note NI strongly recommends adding an EEPROM to your adapter module for identification purposes. This provides better electrical protection for both the adapter module and the NI-793*x*R/NI 797*x*R. It also improves the software configuration experience within LabVIEW FPGA. For more information about including an EEPROM in your adapter module design for identification purposes, refer to Chapter 4, *Interfacing Adapter Modules with NI-793xR and NI 797xR Devices*.

Note The FPGA parses all `.fam` files. If the FPGA encounters an IO Module ID in a previously parsed `.fam` file, the IO Module does not show up in the list.

The format of the 32-bit IO Module ID is as follows:

Bits<31..16>: Vendor ID

Bits<15..0>: Product ID

The Vendor ID is a 16-bit number which is unique for each adapter module manufacturer. Vendor IDs are assigned to individual manufacturers by NI. Each adapter module manufacturer should use their allotted Vendor ID for all adapter modules that they produce.

To obtain your adapter module Vendor ID, visit `ni.com/ask`, and create a new tecnical support request. For more information about obtaining your adapter module Vendor ID, refer to the *Registration* section of Chapter 1, *Before You Begin*.

The Product ID is a value that is chosen by the adapter module manufacturer. Each adapter module manufacturer should ensure that a unique Product ID is chosen for each type of adapter module.

The Vendor ID and Product ID combine to form the unique IO Module ID for each individual adapter module created by a manufacturer.

The following is an example IO Module ID:

IO Module ID: `0xFFFF0001`

Vendor ID: `0xFFFF`

Product ID: `0x0001`

Programming the EEPROM in LabVIEW

NI provides a LabVIEW host VI which you can use to program the IO Module ID into the adapter module. The VI library containing this VI is installed in the following location:

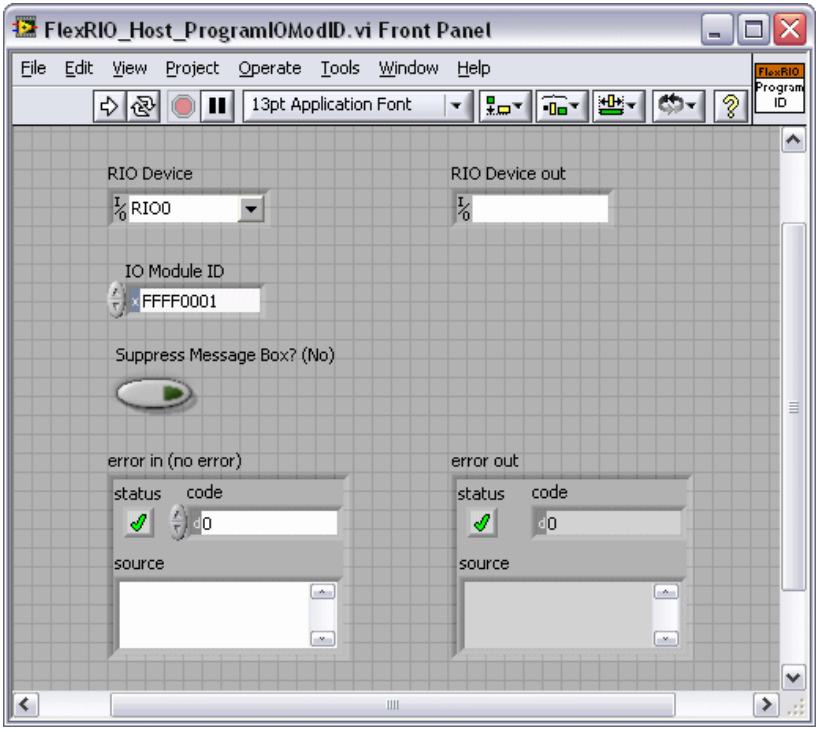
`<`*`LabVIEW`*`>\vi.lib\FlexRIO\FlexRIO_HostInterface.llb`

Complete the following steps to program the IO Module ID into your adapter module:

1. Launch LabVIEW and open the `FlexRIO_Host_ProgramIOModID.vi`.
2. On the front panel of the `FlexRIO_Host_ProgramIOModID.vi`, configure the **RIO Device** control to match the RIO device name of the NI-793*x*R/NI 797*x*R connected to your adapter module.

   Determine which RIO device name corresponds to your NI-793*x*R/NI 797*x*R by launching Measurement & Automation Explorer (MAX) and browsing to **Devices and Interfaces**. If you have only one RIO device in your system, the name for your device is typically `RIO0`.
3. On the front panel of `FlexRIO_Host_ProgramIOModID.vi`, configure the **IO Module ID** control to the correct 32-bit IO Module ID for your adapter module.

Figure 10-1. FlexRIO_Host_ProgramIOModID.vi Front Panel

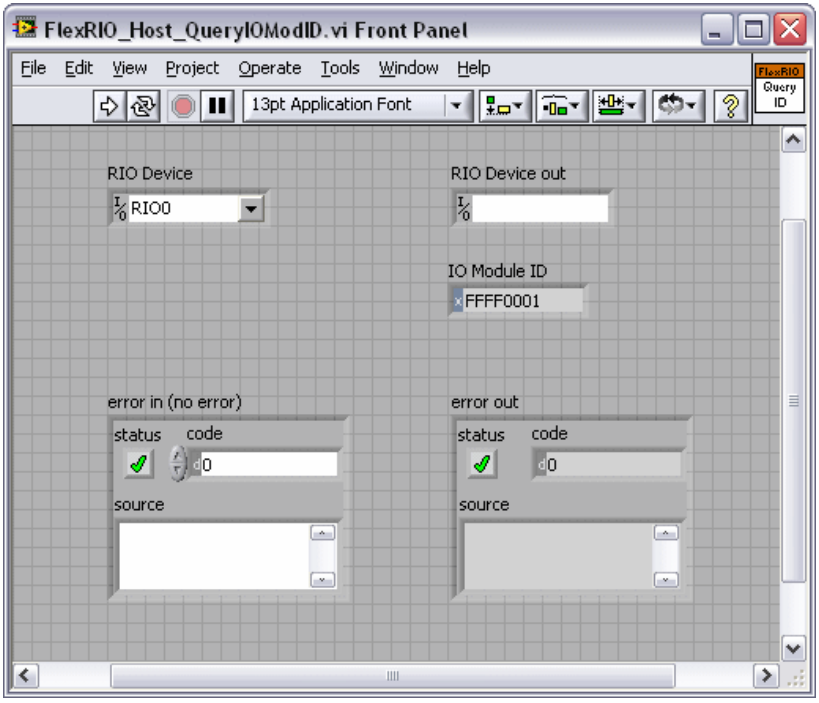


4. Run the VI. If the VI runs successfully without error, then your IO Module ID is programmed properly.

Complete the following steps to read the IO Module ID from the EEPROM of an attached adapter module:

1. Launch LabVIEW and open the `FlexRIO_Host_QueryIOModID.vi`.
2. On the front panel of the `FlexRIO_Host_QueryIOModID.vi`, configure the **RIO Device** control to match the RIO device name of the NI-793*x*R/NI 797*x*R connected to your adapter module. Determine which RIO device name corresponds to your NI-793*x*R/NI 797*x*R by launching MAX and browsing to **Devices and Interfaces**. If you have only one RIO device in your system, the name for your device is typically `RIO0`.
3. Run the VI. If the VI runs successfully without error, the IO Module ID indicator on the VI front panel returns the IO Module ID of the attached adapter module.

Figure 10-2. FlexRIO_Host_QueryIOMod.vi Front Panel



`FlexRIO_HostInterface.llb` contains additional VIs that you can use for reading and writing to and from arbitrary EEPROM locations. You can also use these VIs to issue raw I^2C bus cycles. For more information about these VIs, refer to the *FlexRIO Help*.

EEPROM Map

Table 10-2 describes the adapter module EEPROM map.

Table 10-2. EEPROM Map

Byte Address	Size (Bytes)	Field Name	Required?
0x0	2	Vendor ID	Yes
0x2	2	Product ID	Yes
0x4	4	Serial Number	No
0x8	24	Reserved	No
0x20	224	User Space	No

Adapter module manufacturers may optionally store a 32-bit serial number in the Serial Number EEPROM field.

The User Space occupies the remainder of the EEPROM. The adapter module manufacturer determines the layout and usage of this portion of the EEPROM. For example, the User Space can be used to store ADC calibration constants.

Access to the I^2C signals is shared with the adapter module socketed CLIP. This provides direct access to the EEPROM from the FPGA. For more information the EEPROM, refer to the *EEPROM Overview* section of Chapter 4, *Interfacing Adapter Modules with NI-793xR and NI 797xR Devices*. For additional information about I^2C signals, refer to Table 12-2, *I^2C Core Interface Signals**.

Creating the Adapter Module Configuration (.fam) File

Note This section is intended for users who are using FlexRIO Support 13.0 or later. If you are using FlexRIO Support 2012 or earlier, refer to the *Creating the Adapter Module Configuration (.fam) File* section to learn how to create a `.tbc` adapter module configuration file.

To define certain electrical characteristics of the adapter module for LabVIEW, you need to create an adapter module configuration (`.fam`) file. LabVIEW FPGA uses this information during the compilation process to properly configure the V_{cco} voltage levels to determine which IO Module ID to expect during the adapter module discovery sequence, and to properly configure the FPGA I/O standards for the GPIO signals on the adapter module connector interface.

The presence of an adapter module configuration file also allows your adapter module to be selectable in the adapter module configuration user interface within the LabVIEW project. All adapter module configuration files are automatically enumerated from the `IO Modules` directory on disk. The `IO Modules` directory is stored in the following location:

- **Windows XP**

  `C:\Documents and Settings\All Users\Shared Documents\National Instruments\FlexRIO\IO Modules`

- **Windows 8/7/Vista**

  `C:\Users\Public\Documents\National Instruments\FlexRIO\IO Modules`

NI recommends that you create a manufacturer-specific subfolder within the `IO Modules` directory and place your adapter module configuration files within this subfolder. Creating a manufacturer-specific folder for your `.fam` files helps facilitate better organization of adapter module support files when you install adapter modules from multiple manufacturers on the same

system. During device configuration and at VI build, the FlexRIO support software automatically enumerates all files within the `IO Modules` directory inside subfolders of the directory.

Complete the following steps to create a `.fam` file that defines your adapter module characteristics within LabVIEW.

1. Go to the FlexRIO `IO Modules` directory and create a manufacturer-specific folder that contains the `.fam` file for the new adapter module.
2. Create your `.fam` file.

   An example `.fam` file is provided in the following location:

   - **Windows XP**

     `C:\Documents and Settings\All Users\Shared Documents\National Instruments\FlexRIO\Module Development Kit\Examples\IO Module\ExampleIOModule.fam`

   - **Windows 8/7/Vista**

     `C:\Users\Public\Documents\National Instruments\FlexRIO\Module Development Kit\Examples\IO Module\ExampleIOModule.fam`

   You can use this file as a template for your adapter module configuration file. If you use the NI example `.fam` file, make sure to modify the required values described in the *Adapter Module Configuration (.fam) Values* section and rename the file for your adapter module.
3. Save the `.fam` file to the location you created in step 1.

Adapter Module Configuration (.fam) Values

The adapter module configuration (`.fam`) file uses a standard INI-file syntax. The following sections contain detailed descriptions of each of the supported configuration values contained within the file. An example `.fam` file is also provided. A `.fam` file consists of a Common section and a Socket-specific section.

Common .fam Values

Note The Common section is required in your `.fam` file.

The Common section describes basic information about the adapter module. Table 10-3 lists the supported Common configuration values.

Table 10-3. Supported Common Configuration Values

Key Name	Data Type	Description
FormatVersion	Integer	Specifies which version of the adapter module configuration file annotations are used to create the adapter module configuration (`.fam`) file. Set this value to `1`.
OldestCompatibleFormatVersion	Integer	Specifies which version of the adapter module configuration file annotations that the adapter module configuration (`.fam`) file is compatible with. Set this value to `1`.
Manufacturer	String	Specifies the name of the manufacturer that created the adapter module. This name displays in the IO Module Properties dialog box in the LabVIEW project with the adapter module. **Note**: The `Manufacturer .fam` value cannot include the following characters: `, ; :`
Model	String	Specifies the model name of the adapter module. This name displays in the IO Modules Properties dialog box in the LabVIEW project when the adapter module is configured. **Note**: The `Model .fam` value cannot include the following characters: `, ; :` All `.fam` files are enumerated in the IO Module Properties page. If the combination of the manufacturer and model name (for example, National Instruments NI 5761) is found in a previously parsed `.fam` file, the Details window displays an error when you click the duplicate IO Module.

Table 10-3. Supported Common Configuration Values (Continued)

Key Name	Data Type	Description
Description	String	Provides a general description of the capabilities and function of the adapter module. This information displays in the IO Module Properties dialog box in the LabVIEW project. You may use `\n` to format the string with end-of-line characters.
IOModuleID	U32	Specifies the unique IO Module ID value that is stored in the identification EEPROM. Set this key to the value of the IO Module ID that you created in the *Programming the EEPROM* section. If your adapter module does not contain an EEPROM for identification purposes, remove this key from the adapter module configuration file.
CompatibleCLIPSockets	String	Specifies which FPGA families are supported. Enter `FlexRIO-K7IOModule` for this value.

Socket-specific .fam Values

The socket-specific section of the `.fam` file describes information that is specific to your device's CLIP socket. Table 10-4 lists the supported configuration values for the NI-793*x*R/NI 797*x*R device sockets.

Note Name the socket-specific section `FlexRIO-K7IOModule`. Socket-specific sections are required for each socket family listed in the CompatibleCLIPSockets tag.

Table 10-4. Supported Socket-specific Configuration Values

Key Name	Data Type	Description
Default CLIP	String	Specifies the name of the default CLIP implementation for the adapter module. This name is specified within the `<CLIPDeclaration>` tag in the CLIP XML file. When you create a new LabVIEW project, the FPGA device is auto-discovered with an adapter module inserted, and this CLIP is automatically selected for use with the adapter module. Refer to Chapter 12, *Configuring Your Adapter Module for Use with NI-793xR/797xR Modules and LabVIEW FPGA*, for more information about CLIP.
VccoLevel	Float	Specifies the voltage level (in volts) to configure the V_{cco} bank for. The default value is 2.5. Valid values are: 1.2, 1.5, 1.8, 2.5, and 3.3. Values that include decimals must use the `.` character and not the `,` character. For example, 1.2V is valid, but 1,2V is not.
(Optional) IoModSyncClock Refer to Table 10-7 for IoModSyncClock defined values and descriptions.	String	Specifies how to support the clock for NI 797*x*R devices. If this key name is not present, the adapter module clock is not supported. This key is ignored on NI-793*x*R devices.
(Optional) IoModSyncClockSource Refer to Table 10-7 for IoModSyncClockSource defined values and descriptions.	String	Specifies the default IoModSyncClock source for NI 797*x*R devices. Use this tag only when the IoModSyncClock is set to SUPPORTED or REQUIRED. If this key name is not present, PXI_CLK10 is selected by default. This key is ignored on NI-793*x*R devices.
(Optional) Fam3v3Stage*	Float	Specifies the stage in which the 3.3 V power rail is turned on. Valid values are 0, 1, and 2.
(Optional) Fam12vStage*	Float	Specifies the stage in which the 12 V power rail is turned on. Valid values are 0, 1, and 2.
(Optional) FamVccoStage*	Float	Specifies the stage in which the V_{cco} power rail is turned on. Valid values are 0, 1, and 2.

Table 10-4. Supported Socket-specific Configuration Values (Continued)

Key Name	Data Type	Description
(Optional) Stage0Delay*	Float	Describes the length of the delay after Stage 0 completes. Valid values are 0 ms to 500 ms.
(Optional) Stage1Delay*	Float	Describes the length of the delay after Stage 1 completes. Valid values are 0 ms to 500 ms.
(Optional) Stage2Delay*	Float	Describes the length of the delay after Stage 2 completes. Valid values are 0 ms to 500 ms.
* The `.fam` file uses default values if you do not specify all of these values. Refer to Table 9-8 for default values for the power rail sequence.		

Caution To avoid timeouts, the total sequence delay (Stage0Delay + Stage1Delay + Stage2Delay) cannot exceed 500 ms.

Note If you do not want to use default values, you must specify a value for every key. If you do not specify a value for every key, the `.fam` file specifies default values for every key.

Table 10-5. Power Rail Sequence Default Values

Key Name	Default Values
Fam3v3Stage	1
Fam12vStage	0
FamVccoStage	0
Stage0Delay	10 ms
Stage1Delay	0 ms
Stage2Delay	0 ms

The default value for power rail sequence values does not match the typical behavior of the NI 795*x*R and NI 796*x*R devices. If you are using an NI-793*x*R or NI 797*x*R device, but want to use similar power supply sequencing to the NI 795*x*R and NI 796*x*R devices, refer to the following table for values that represent typical behavior of the NI 795*x*R and NI 796*x*R devices.

Table 10-6. NI 795xR and NI 796xR Representative Power Rail Sequence Values

Key Name	Representative NI 795*x*R/796*x*R Value
Fam3v3Stage	2
Fam12vStage	0
FamVccoStage	1
Stage0Delay	25 ms
Stage1Delay	15 ms
Stage2Delay	0 ms

V_{cco} Level

The adapter module configuration file specifies the V_{cco} bank values. These values are fixed by the adapter module designer and cannot be changed at run-time within LabVIEW FPGA. The available voltage options for the V_{cco} bank are 1.2V, 1.5V, 1.8V, 2.5V, and 3.3V.

Note Values that include decimals must use the `.` character and not the `,` character. For example, 1.2V is valid, but 1,2V is not.

Caution The selected V_{cco} value must match the electrical design of the corresponding adapter module. Failure to do so risks damaging the adapter module or the NI-793*x*R/NI 797*x*R. NI is *not* liable for any damage resulting from such misuse.

Adapter Module IOModuleID

The adapter module `IOModuleID` key determines the IO Module ID for your adapter module. The `IOModuleID` value in the adapter module configuration file must be in hexadecimal format, prefixed with `0x`.

This example demonstrates how to specify the `IOModuleID` value in the adapter module configuration file:

```
IOModuleID=0xFFFF0001
```

In this example, `0xFFFF` is the Vendor ID, and `0x0001` is the Product ID.

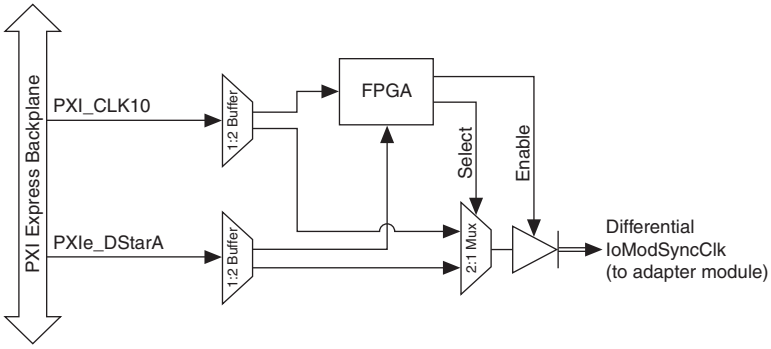
Note If the adapter module you are configuring does not contain an EEPROM, you must remove the `IOModuleID` value from the `.fam` file.

IoModSyncClk.fam Values

Note The NI-793*x*R does not have access to the IoModSyncClk line. The following information applies to NI 797*x*R devices only. All IoModSyncClk keys in the `.fam` file are ignored on NI-793*x*R modules.

The NI 797*x*R provides the IoModSyncClk signal for synchronization between adapter modules. IoModSyncClk is a LVPECL clock that can be sourced either by PXI_CLK10 or by PXIe_DStarA, as shown in the following figure.

Figure 10-3. IoModSyncClk Source Block Diagram



For more information about the IoModSyncClk and how to use it in your adapter module design, refer to the *IoModSyncClk* section of Chapter 4, *Interfacing Adapter Modules with NI-793xR and NI 797xR Devices*. Refer to the *NI PXIe-797xR Base Clocks* topic in the *FlexRIO Help* for more information about clock resources.

Enabling IoModSyncClk

You can enable IoModSyncClk by including optional keys within the **Socket-Specific** section of the `.fam` file of the corresponding adapter module. The two optional keys are as follows:

Table 10-7. Optional Keys for Enabling IoModSyncClock

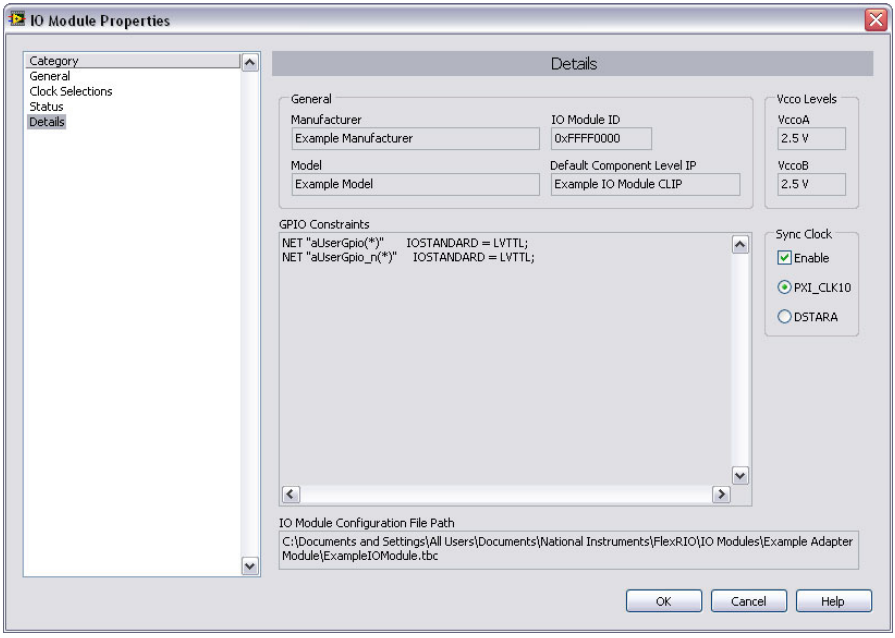
Key Name	Defined Values	Description
1. IoModSyncClock (Optional)—Specifies how to support the clock. If this key name is not present, then the adapter module clock is not supported.	UNSUPPORTED	IoModSyncClk is tristated.
	SUPPORTED	IoModSyncClk can be enabled or tristated from the **Sync Clock** section of the **IO Module Properties** menu. The default selection is enabled.
	REQUIRED	IoModSyncClock is always enabled.
	PXICLK10REQUIRED	IoModSyncClk is always enabled and is sourced by PXI_CLK10.
	DSTARAREQUIRED	IoModSyncClock is always enabled and is sourced by PXIe_DStarA.
2. IoModSyncClockSource (Optional)—Specifies the default clock source. IoModSyncClockSource is only used when IoModSyncClock is set to SUPPORTED or REQUIRED. If IoModSyncClockSource is not present, the default clock source is PXI_CLK10.	CLK10	IoModSyncClk is sourced by PXI_CLK10.
	DSTARA	IoModSyncClk is sourced by PXIe_DStarA.

When the IoModSyncClock key is set to SUPPORTED or REQUIRED, you can also select settings for the IoModSyncClk in the LabVIEW Project. To access this menu, right-click the **IO Module** item under the FPGA Target in the Project Explorer window and select **Properties** to display the **IO Module Properties** dialog box. The Sync Clock section in the **Details** category controls the IoModSyncClk settings.

When the IoModSyncClockSource key is set to SUPPORTED or REQUIRED in the `.fam` file, the default IoModSyncClk source is selected. The value selected under the Details category in

the IO Module Properties dialog box overrides the default IoModSyncClockSource `.fam` file key value.

Figure 10-4. IO Module Properties Sync Clock Enabled



At compile time, the FPGA selects the source for the IoModSyncClk, as well as the enabled or disabled state. To change these settings, you must recompile the FPGA VI.

FlexRIO-K7IOModule Constraints .fam Values

Note The FlexRIO-K7IOModule Constraints section is required in your `.fam` file in LabVIEW 2013. This constraints section is ignored in LabVIEW 2014 and later if you include it in the `.fam` file.

The FlexRIO-K7IOModule Constraints section specifies FPGA compilation constraints information for the FPGA pins dedicated to the adapter module interface. The syntax of data in this section is identical to the syntax of raw UCF data used by the Xilinx ISE tool. The constraints data is concatenated to the UCF file that is used in the FPGA compilation.

Use the FlexRIO-K7IOModule Constraints section to properly constrain the electrical I/O standard that is used for each adapter module general-purpose I/O (GPIO) pin. These constraints depend on the V_{cco} settings, which are also specified in the `.fam` file. Like the V_{cco} levels, the FlexRIO-K7IOModule Constraints values should be fixed by the adapter module designer and cannot be changed at run-time from within LabVIEW FPGA. Include all physical-interface

related constraints—such as I/O standard, drive strength, and termination—in the FlexRIO-K7IOModule Constraints section.

Caution You must set the I/O standard constraint for the UserGPIO pins. If you do not constrain these FPGA I/O pins, the compiler assigns a default I/O standard, which could prevent proper I/O function and possibly damage the NI-793*x*R/NI 797*x*R or adapter module.

You can set FlexRIO UserGpio pins to any Xilinx I/O standard with the following conditions:

- The V_{cco} rail is set to the required V_{cco} value for this I/O standard.
- The I/O standard does not require a reference voltage (V_{ref}).

Table 10-8 lists the Xilinx I/O standards supported by the FlexRIO GPIO and V_{cco} power rails on NI-793*x*R and NI 797*x*R devices.

Table 10-8. FlexRIO Supported Xilinx I/O Standards

I/O Standard	V_{cco}
LVTTL	3.3V
LVCMOS*xx*[*]	1.2V, 1.5V, 1.8V, 2.5V, and 3.3V
LVDS_25 (inputs use internal 100 Ω differential parallel termination)	2.5V
[*] Where *xx* is determined by the selected voltage.	

Note The I/O standards listed in the above table are determined by Xilinx specifications and are subject to change. To determine the most current I/O standard constraints for your FlexRIO device, refer to the Xilinx documentation, available at `www.xilinx.com`. For a list of Xilinx documents applicable to FlexRIO applications, refer to Appendix C, *Xilinx Documentation References*.

Refer to the *NI-793xR and NI 797xR FPGA I/O Bank Voltages* section of Chapter 4, *Interfacing Adapter Modules with NI-793xR and NI 797xR Devices*, for more information about FPGA I/O bank voltages.

For example, the following FlexRIO-K7IOModule Constraints section constrains GPIO pair 20 to use the LVTTL I/O standard:

```
[FlexRIO-K7IOModule Constraints]
NET "aUserGpio(20)"          IOSTANDARD = LVTTL;
NET "aUserGpio_n(20)"        IOSTANDARD = LVTTL;
```

The next example Constraints section constrains GPIO pair 20 to the LVCMOS25 I/O standard, and Gpio38 (a MRCC line) to the LVDS_25 I/O standard.

```
[FlexRIO-K7IOModule Constraints]
NET "aUserGpio(20)"          IOSTANDARD = LVCMOS25;
NET "aUserGpio_n(20)"        IOSTANDARD = LVCMOS25;
NET "aUserGpio(38)"          IOSTANDARD = LVDS_25;
NET "aUserGpio_n(38)"        IOSTANDARD = LVDS_25;
```

NI recommends that you constrain all GPIO lines even if they are not used. The following snippet demonstrates what the unused constraint portion of your FlexRIO-K7IOModule Constraints section may look like. For example, if GPIO lines 40 to 60 are unused, explicitly assign them all to LVCMOS25.

```
# Unused GPIO pins
NET "aUserGpio(40)"          IOSTANDARD = LVCMOS25;
NET "aUserGpio_n(40)"        IOSTANDARD = LVCMOS25;
...
NET "aUserGpio(60)"          IOSTANDARD = LVCMOS25;
NET "aUserGpio_n(60)"        IOSTANDARD = LVCMOS25;
```

Note To ensure the correct compilation of your adapter module constraints information, NI recommends using `()` as the bus delimiters in both the `.ucf` and the `.fam` files, instead of using <>. For example: `aUserGpio(20)`.

FlexRIO-K7IOModule Vivado Constraints .fam Values

Note The FlexRIO-K7IOModule Vivado Constraints section is required in your `.fam` file for LabVIEW FPGA 2014 and later. This constraints section is ignored in LabVIEW 2013 if you include it in the `.fam` file.

Note Use the FlexRIO-K7IOModule Constraints section for NI 797*x*R devices in LabVIEW 2013.

You cannot directly specify constraints for Vivado in `.fam` files. To specify constraints for your project, configure the `.fam` file to point to the `.xdc` file that specifies the constraints for your project. The following code shows an example of how to point to the NI 5782 `.xdc` file from the `.fam` file:

```
[FlexRIO-K7IOModule Vivado Constraints]
File=NI5782IOModuleK7.xdc
```

The `.xdc` constraints file that you point to specifies FPGA compilation constraints information for the FPGA pins dedicated to the adapter module interface.

Note The `.xdc` file must be located in the same directory as the `.fam` file.

The syntax of data in this section is identical to the syntax of raw XDC data used by the Xilinx Vivado tool. The constraints data is concatenated to the XDC file that is used in the FPGA compilation.

Use the `.xdc` constraints file to properly constrain the electrical I/O standard that is used for each adapter module general-purpose I/O (GPIO) pin. These constraints depend on the V_{cco} settings, which are also specified in the `.fam` file. Like the V_{cco} levels, the FlexRIO-IOModule values should be fixed by the adapter module designer and cannot be changed at run-time from within LabVIEW FPGA. Include all physical-interface related constraints—such as I/O standard, drive strength, and termination—in the FlexRIO-IOModule Constraints section.

Caution You must set the I/O standard constraint for the UserGPIO pins. If you do not constrain these FPGA I/O pins, the compiler assigns a default I/O standard, which could prevent proper I/O function and possibly damage the NI-793*x*R/NI 797*x*R or adapter module.

FlexRIO UserGpio pins may be set to any Xilinx I/O standard with the following conditions:

- The V_{cco} rail is set to the required V_{cco} value for this I/O standard.
- The I/O standard does not require a reference voltage (V_{ref}).

Table 10-9 lists the Xilinx I/O standards supported by the FlexRIO GPIO and V_{cco} power rails on NI 797*x*R devices.

Table 10-9. FlexRIO Supported Xilinx I/O Standards

I/O Standard	V_{cco}
LVTTL	3.3V
LVCMOS*xx*[*]	1.2V, 1.5V, 1.8V, 2.5V, and 3.3V
LVDS_25 (inputs use internal 100 Ω differential parallel termination)	2.5V
[*] Where *xx* is determined by the selected voltage.	

Note The I/O standards listed in Table 10-9 are determined by Xilinx specifications and are subject to change. To determine the most current I/O standard constraints for your FlexRIO device, refer to the Xilinx documentation, available at `www.xilinx.com`. For a list of Xilinx documents applicable to FlexRIO applications, refer to Appendix C, *Xilinx Documentation References*.

Refer to the *NI-793xR and NI 797xR FPGA I/O Bank Voltages* section of Chapter 4, *Interfacing Adapter Modules with NI-793xR and NI 797xR Devices*, for more information about FPGA I/O bank voltages.

For example, the following FlexRIO-K7IOModule Vivado Constraints is located in the `.xdc` file that accompanies the `.fam` file. The following snippet demonstrates what the beginning of the clocking section of the `.xdc` may look like. It sets up the GPIO lines to use the LVDS_25 IO standard and enable the 100 Ω differential termination.

```
# Clock signals.
set_property IOSTANDARD LVDS_25  [get_ports {aUserGpio[38]}]
set_property DIFF_TERM TRUE [get_ports {aUserGpio[38]}]
set_property IOSTANDARD LVDS_25  [get_ports {aUserGpio_n[38]}]
set_property DIFF_TERM TRUE [get_ports {aUserGpio_n[38]}]
```

NI recommends that you constrain all GPIO lines even if they are not used. The following example Constraints section demonstrates what the unused constraint section of your `.xdc` file may look like. For example, if GPIO lines 40 to 60 are unused, explicitly assign them all to LVCMOS25.

```
# Unused GPIO pins
set_property IOSTANDARD LVCMOS25 [get_ports {aUserGpio[40]}]
set_property IOSTANDARD LVCMOS25 [get_ports {aUserGpio_n[40]}]
...
set_property IOSTANDARD LVCMOS25 [get_ports {aUserGpio[60]}]
set_property IOSTANDARD LVCMOS25 [get_ports {aUserGpio_n[60]}]
```

Note The preceding code examples are snippets of the example design files located at the Start menu. Refer to the *Creating the Adapter Module Configuration (.fam) File* section for information about the location of these design files.

Note Refer to the Vivado documentation for information about `.xdc` syntax.

Example .fam File for LabVIEW 2013

The following is an example of an abridged `.fam` file for use with NI 797*x*R devices and LabVIEW 2013. Refer to the *Creating or Acquiring the IP for the FlexRIO Adapter Module* section of Chapter 12, *Creating Socketed Component-Level IP for Your Adapter Module and NI-793xR/NI 797xR Modules*, for information about complete example files that ship with the FlexRIO Adapter Module Development Kit.

```
[Common]
FormatVersion=1
OldestCompatibleFormatVersion=1
Manufacturer=Example Manufacturer
Model=Example Model
Description=This is an example adapter module configuration file.
IOModuleID=0xFFFF0001
```

```
CompatibleCLIPSockets=FlexRIO-K7IOModule

[FlexRIO-K7IOModule]
DefaultCLIP=ExampleIOModuleCLIP
IoModSyncClock=SUPPORTED
IoModSyncClockSource=CLK10
VccoLevel=2.5
Fam3v3Stage=1
Fam12vStage=0
FamVccoStage=0
Stage0Delay=10
Stage1Delay=0
Stage2Delay=0

[FlexRIO-K7IOModule Constraints]
NET "aUserGpio(1)"          IOSTANDARD = LVCMOS25;
NET "aUserGpio_n(1)"        IOSTANDARD = LVCMOS25;
NET "aUserGpio(2)"          IOSTANDARD = LVCMOS25;
NET "aUserGpio_n(2)"        IOSTANDARD = LVCMOS25;
...
NET "aUserGpio(20)"         IOSTANDARD = LVCMOS25;
NET "aUserGpio_n(20)"       IOSTANDARD = LVCMOS25;
```

Example .fam (NI-793*x*R/NI 797*x*R) for LabVIEW 2014 and later

The following is an example of an abridged `.fam` file for use with NI-793*x*R/NI 797*x*R devices and LabVIEW 2014 or later. Note that the constraints for this file are specified in the `.xdc` file. Refer to the *Creating or Acquiring the IP for the FlexRIO Adapter Module* section of Chapter 12, *Creating Socketed Component-Level IP for Your Adapter Module and NI-793xR/NI 797xR Modules*, for information about complete example files that ship with the FlexRIO Adapter Module Development Kit.

```
[Common]
FormatVersion=1
OldestCompatibleFormatVersion=1
Manufacturer=Example Manufacturer
Model=Example Model
Description=This is an example adapter module configuration file.
IOModuleID=0xFFFF0001
```

```
CompatibleCLIPSockets=FlexRIO-K7IOModule

[FlexRIO-K7IOModule]
DefaultCLIP=ExampleIOModuleCLIP
IoModSyncClock=SUPPORTED
IoModSyncClockSource=CLK10
VccoLevel=2.5
Fam3v3Stage=1
Fam12vStage=0
FamVccoStage=0
Stage0Delay=10
Stage1Delay=0
Stage2Delay=0

[FlexRIO-K7IOModule Vivado Constraints]
File=ExampleIOModuleCLIPsK7.xdc
```

Configuring .fam files for Compatibility with both LabVIEW 2013 and LabVIEW 2014

If you configure an adapter module for compatibility with both LabVIEW 2013 and LabVIEW 2014, refer to the following example .fam file:

```
[Common]
FormatVersion=1
OldestCompatibleFormatVersion=1
Manufacturer=Example Manufacturer
Model=Example Model
Description=This is an example adapter module configuration file.
IOModuleID=0xFFFF0001
CompatibleCLIPSockets=FlexRIO-K7IOModule

[FlexRIO-K7IOModule]
DefaultCLIP=ExampleIOModuleCLIP
IoModSyncClock=SUPPORTED
IoModSyncClockSource=CLK10
VccoLevel=2.5
Fam3v3Stage=1
Fam12vStage=0
FamVccoStage=0
```

```
Stage0Delay=10
Stage1Delay=0
Stage2Delay=0
[FlexRIO-K7IOModule Constraints]
NET "aUserGpio(1)"          IOSTANDARD = LVCMOS25;
NET "aUserGpio_n(1)"        IOSTANDARD = LVCMOS25;
NET "aUserGpio(2)"          IOSTANDARD = LVCMOS25;
NET "aUserGpio_n(2)"        IOSTANDARD = LVCMOS25;
...
NET "aUserGpio(20)"         IOSTANDARD = LVCMOS25;
NET "aUserGpio_n(20)"       IOSTANDARD = LVCMOS25;
[FlexRIO-K7IOModule Vivado Constraints]
File=ExampleIOModuleCLIPsK7.xdc
```

11

Creating Socketed Component-Level IP for Your Adapter Module and NI 795xR/796xR Modules

To connect external signals from your adapter module to your LabVIEW VI, you need to add your own custom HDL code to the LabVIEW FPGA project. Creating custom socketed component-level IP (CLIP) allows you to program the FPGA to interface with the adapter module circuitry.

FlexRIO devices support two types of CLIP: user-defined and socketed. User-defined CLIP allows you to insert HDL IP into a LabVIEW target, enabling VHDL code to communicate directly with an FPGA VI. Socketed CLIP provides the same IP integration functionality of the user-defined CLIP, while also providing direct communication between the adapter module connector interface and the FPGA VI. For a graphical overview of CLIP and its relationship with FlexRIO hardware, refer to Figure 2-4, *LabVIEW FPGA, CLIP, and Hardware Integration Diagram (Virtex-5)*.

To add CLIP to a LabVIEW project, the LabVIEW FPGA Module must support the IP that you use as CLIP. The CLIP must also have an accompanying XML declaration file. The XML declaration file describes the elements of the IP, which the LabVIEW FPGA Module uses to add the IP to the LabVIEW project. To use CLIP with your adapter module, complete the following steps:

1. Create or acquire the IP, and, if needed, create optional adapter module IP constraints by setting them in the `.ucf` file.
2. Create a CLIP declaration XML file to define the adapter module I/O for LabVIEW FPGA.
3. Configure your adapter module to use CLIP in the LabVIEW project.

These steps are described in greater detail below.

Using the CLIP Wizard

You can also use the CLIP Wizard to help create your custom CLIP or to edit existing CLIP. Complete the following steps to access the CLIP Wizard:

1. Right-click the desired target in the Project Explorer window and select **Properties**.

2. In the Category section of the FPGA Target Properties dialog box, select **Component-Level IP**.
3. On the right side of the FPGA Target Properties dialog box, click the **Create File** icon to create new CLIP or, to edit existing CLIP, select the desired CLIP from the list and click the **Modify File** icon.

Note If your `.fam` or `.tbc` file is in the location listed in Chapter 9, *Creating Socketed Component-Level IP for Your Adapter Module and NI 795xR/796xR Modules*, you can select the configuration values for your adapter module in the CLIP Wizard, which auto-populates your XML declaration file. If you do *not* use the CLIP Wizard, follow the instructions in the *ExampleIOModuleCLIPV5.xml* section or the *Configuring the FlexRIO Adapter Module in LabVIEW* section of this chapter to create your XML declaration file.

Creating or Acquiring the IP for the FlexRIO Adapter Module

To use socketed CLIP with your FPGA adapter module, you must first create or acquire IP, in the form of VHDL code or other HDL code wrapped in VHDL code, to compile into the FPGA target. This IP performs the following tasks:

- Directly controls the FPGA pins on the adapter module connector interface.
- Provides a signal interface to LabVIEW FPGA, which allows LabVIEW to interact with the adapter module. The author of the adapter module CLIP can customize the interface exposed to LabVIEW FPGA.
- Implements any additional adapter module-specific functionality.

To help with adapter module component-level IP development, National Instruments provides example adapter module socketed CLIP files.

The example CLIP for NI 795*x*R and NI 796*x*R devices is composed of the following files:

- `ExampleIOModuleCLIPV5.vhd`
- `ExampleIOModuleCLIPV5.ucf`
- `ExampleIOModuleCLIPV5.xml`

The example socketed CLIP files are installed in the following location:

- **Windows XP**

  `C:\Documents and Settings\All Users\Shared Documents\National Instruments\FlexRIO\Module Development Kit\Examples\IO Module`

- **Windows 8/7/Vista**

  `C:\Users\Public\Documents\National Instruments\FlexRIO\Module Development Kit\Examples\IO Module`

The following sections describe the process of creating a new adapter module socketed CLIP using the example adapter module CLIP. For additional information about creating CLIP, refer to the *Integrating Third-Party IP (FPGA Module)* and the *NI FlexRIO Help* books of the *FPGA Module* in the *LabVIEW Help*.

Tip When developing new socketed CLIP from the provided example files, first copy the example adapter module CLIP files to another location on your disk drive so that you can safely modify them.

ExampleIOModuleCLIPV5.vhd

Note The ExampleIOModuleCLIPV5.vhd section contains steps to create CLIP for Virtex-5 devices.

`ExampleIOModuleCLIPV5.vhd` is the top-level VHDL file of the socketed CLIP and defines the top-level port interface to the CLIP. LabVIEW FPGA uses this interface when compiling the CLIP into LabVIEW FPGA. This section explains the changes you must make to the example file in order to adapt this CLIP to your adapter module.

1. Incorporate your IP into this VHDL file, writing new VHDL code or instantiating existing IP as necessary. The adapter module connector I/O signals are listed in the ExampleIOModuleCLIP port interface. These signals are the fixed interface to the adapter module and are therefore required in the port interface for your adapter module CLIP. Tables 11-1, 11-2, and 11-3 list the additional signals available for use in your CLIP interface.

Table 11-1. GPIO and CLK Signals from Adapter Module

Signal Name	Direction	Data Type	Description
aUserGpio	Bidirectional	std_logic_ vector (65 downto 0)	66 GPIO FPGA pins (non-inverted).
aUserGpio_n	Bidirectional	std_logic_ vector (65 downto 0)	66 GPIO FPGA pins (inverted).
rIoModGpioEn	To CLIP	std_logic	Adapter module GPIO buffer enable signal. When this signal is logic high, it is safe to enable the buffers for the GPIO signals. When this signal is logic low, the I/O buffers for the GPIO signals must be tristated.

Table 11-1. GPIO and CLK Signals from Adapter Module (Continued)

Signal Name	Direction	Data Type	Description
UserGClkLvds	To CLIP	std_logic	Differential external clock input (non-inverted). This signal is routed via a global clock pin of the FPGA.
UserGClkLvds_n	To CLIP	std_logic	Differential external clock input (inverted). This signal is routed via a global clock pin of the FPGA.
UserGClkLvttl	To CLIP	std_logic	Single-ended external clock input. This signal is routed via a global clock pin of the FPGA.

Table 11-2. CLK Signals to LabVIEW FPGA

Signal Name	Direction	Data Type	Description
IoModClipClock0	From CLIP	std_logic	External clock signal routed to LabVIEW FPGA. This signal is exposed as an external clock resource in LabVIEW FPGA named `IO Module Clock 0`.
IoModClipClock1	From CLIP	std_logic	External clock signal routed to LabVIEW FPGA. This signal is exposed as an external clock resource in LabVIEW FPGA named `IO Module Clock 1`.

Table 11-3. I²C Core Interface Signals*

Signal Name	Direction	Data Type	Description
rLvFpgaReqI2cBus	From CLIP	std_logic	Drive a `1` to this port to request control of the shared I²C bus. Only request the I²C bus if the adapter module is properly identified and powered. When you finish accessing the I²C bus, you must always release access to the I²C bus.
rLvFpgaAckI2cBus	To CLIP	std_logic	This signal indicates a `1` when I²C bus access is granted after you assert rLvFpgaReqI2cBus. Commands sent to the I²C controller are ignored if this signal indicates a `0`.
rLvFpgaI2cGo	From CLIP	std_logic	Drive a `1` to this signal to send a command to the I²C controller. You must configure all I²C controller inputs before driving this signal high. The I²C controller latches the command input when it detects a rising edge on this signal and begins the I²C transaction. Do not reassert this signal until LvFPGAI2CDone indicates a `1`.
rLvFpgaI2cStart	From CLIP	std_logic	Drive a `1` to this signal to send an I²C start pattern at the beginning of your I²C transaction.
rLvFpgaI2cStop	From CLIP	std_logic	Drive a `1` to this signal to send an I²C stop pattern at the end of your I²C transaction.
rLvFpgaI2cRd	From CLIP	std_logic	Drive a `1` to this signal to indicate a read transaction, meaning the I²C controller returns RdData from the bus. Drive a `0` to this signal to indicate a write transaction, meaning the I²C controller drives the WtData value to the bus.
rLvFpgaI2cWtData	From CLIP	std_logic_ vector (7 downto 0)	This is the 8-bit data value written to the bus by the I²C controller during a Write transaction.

Table 11-3. I²C Core Interface Signals* (Continued)

Signal Name	Direction	Data Type	Description
rLvFpgaI2cAck	To CLIP	std_logic	Valid after an I²C transaction. A `1` indicates that the I²C controller read an Ack from the bus.
rClktoSocket	To CLIP	std_logic	This is a copy of the 40MHz RioClk used by default in LV FPGA diagrams. All I²C signals—denoted by the *r* prefix—are synchronous to this clock.
rLvFpgaI2cDone	To CLIP	std_logic	A `1` indicates that the I²C controller is done. Wait for `1` before reading the outputs.
rLvFpgaI2cRdData	To CLIP	std_logic_ vector (7 downto 0)	This is the 8-bit data value read from the I²C bus during a Read transaction. Read this value after the completion of a read transaction. A`0` indicates that the bus is busy.
* All I2C core interface signals are synchronized to the rClkToSocket signal.			

2. Expose an interface to LabVIEW FPGA so that your adapter module is accessible from the LabVIEW VI. The collection of signals for this interface is customizable and therefore may be tuned to the particular application domain in which you intend to use your adapter module. Each of these signals must also be added to the port map of the top-level VHDL file for your CLIP. These signals have the following requirements:
   - Signal direction may be only To CLIP or From CLIP. The Bidirectional signal direction is not supported.
   - The signal data type must be one of the following:
     – std_logic
     – std_logic_vector (7 downto 0)
     – std_logic_vector (15 downto 0)
     – std_logic_vector (31 downto 0)
     – std_logic_vector (63 downto 0)
   - If you developed your CLIP using the provided example CLIP, you should rename the top-level entity name and VHDL file name to something appropriate for your adapter module.

Using External Clocks

When importing an external clock for use in your adapter module CLIP and/or in your LabVIEW FPGA VI, you must consider several factors. These considerations apply to CLIP logic and UCF constraints covering these incoming clock signals. Refer to the *Signal Descriptions* section of Chapter 3, *Interfacing Adapter Modules with NI 795xR and NI 796xR Modules*, for a list of the clock-capable adapter module GPIO interface pins.

NI requires that you include the following logic considerations within any CLIP clocking interface:

- Any clock shared with the LV FPGA VI that uses IoModCLIPClock<0..1> or is exported from the socketed CLIP must be stable and free running at all times. If these clocks are unstable, such as during a power-up or reset, you *must not* drive them to LabVIEW FPGA generated logic. Use a clock buffer with an “enable” control to gate an unstable clock, such as a BUFGCE.
- All LV FPGA clocks are stable and free running unless you assert a Reset. After you assert a Reset, assert any DCM/PLL logic using these clocks. Do not drive unstable clocks to LabVIEW FPGA logic or any other logic.

Note NI does not recommend using the IoModClipClock for any new designs. Instead of using IoModClipClock, NI recommends exporting the clock from the socketed CLIP.

Note A Reset may be held for as little as one clock cycle. If your logic requires a longer hold during Reset, you should implement application appropriate logic, such as a shift register, to define the necessary hold time.

NI recommends also incorporating the following considerations into your CLIP clocking interface.

- Use a global clock buffer (BUFG) or regional clock buffer (BUFR) to drive an incoming clock onto the FPGA clock network. For more information about the differences between clock buffer options and the limitations of each, refer to Xilinx documentation, available at `www.xilinx.com`. For a list of Xilinx documents applicable to FlexRIO applications, refer to Appendix C, *Xilinx Documentation References*.
- Add OFFSET timing constraints for all adapter module signals captured or driven with the external clock.
- Any LabVIEW FPGA controls and indicators placed in external clock domains could result in warnings if you access them while the external clock is disabled. If you require controls or indicators in external clock domains, structure host code to access these controls or indicators only when the external clock is running.

ExampleIOModuleCLIPV5.ucf

Note The ExampleIOModuleCLIPV5.ucf file provides constraints for the NI 795*x*R and NI 796*x*R devices.

Each IP implementation may require custom constraints settings. Each adapter module CLIP implementation may optionally be packaged with a constraints file which is used during the compilation process. This constraints file should generally provide timing-related constraints only. Do not include constraints for the FPGA I/O pin signal standards in the adapter module CLIP constraints file.

The FPGA I/O pin signal standards are constrained in the adapter module configuration file (`.fam`). Refer to Chapter 9, *Configuring Your Adapter Module for Use with NI 795xR/796xR Modules and LabVIEW FPGA*, for information about the pin signal constraints.

Note To ensure correct compilation of your adapter module constraints information, NI recommends using `()` as the bus delimiters in both the `.ucf` and the `.fam` files, instead of using <>. For example: `aUserGpio(*)`.

Note FlexRIO CLIPs are not instantiated as part of the top-level design. NI recommends using `%CLIPInstancePath%` in your constraints information to ensure that the Xilinx compiler can locate the signal name. If you do not direct the compiler to the signal using `%CLIPInstancePath%`, the compiler may return an error.

If you developed your CLIP using the provided example CLIP, modify the example adapter module constraints file to properly constrain your IP and rename the file to something more appropriate for your adapter module. For more information about timing constraints, refer to Xilinx documentation, available at `www.xilinx.com`. For a list of Xilinx documents applicable to FlexRIO applications, refer to Appendix C, *Xilinx Documentation References*.

ExampleIOModuleCLIPV5.xml

Note You only need to complete the steps in this section if you did *not* use the CLIP Wizard to create your CLIP. The CLIP Wizard auto-populates this XML declaration file based on the information that you select in the CLIP Wizard.

Note The ExampleIOModuleCLIPV5.xml file provides an example of a CLIP declaration file for the NI 795*x*R and NI 796*x*R devices.

The final step of creating a new adapter module socketed CLIP is to create a CLIP declaration file. The CLIP declaration file performs the following tasks:

- Identifies paths to each of the source files that provide the implementation for your CLIP.
- Defines the top-level entity name for your CLIP.

- Defines the collection of signals that are exposed both to the LabVIEW FPGA side of the CLIP interface and to the hardware socketed side of the CLIP interface.
- Defines the name of the CLIP socket(s) compatible with this CLIP.
- Defines the collection of adapter modules compatible with this CLIP.
- Describes and defines how your VHDL ports appear in the LabVIEW FPGA project.

Complete the following steps to adapt the example adapter module socketed CLIP XML file to your new adapter module CLIP.

1. Add the LabVIEW FPGA port interface signals to the list of interface signals in your XML file. All CLIP declaration XML files contain an `<InterfaceList>` section which defines three different types of interfaces: `Fabric`, `LabVIEW`, and `Socket`. Each of these interface types is denoted by specifying the appropriate value for `<InterfaceType>` within a named `<Interface>` section.

   For example, the `ExampleIOModuleCLIPV5.xml` file defines three types of interfaces. One of the interfaces in your CLIP XML file must define each signal in the top-level port interface for your socketed CLIP. The `Fabric` and `Socket` interfaces, however, are fixed for all adapter module CLIP items and cannot be changed. You can customize the LabVIEW FPGA interface. The interface must account for all LabVIEW FPGA CLIP signals that you define in the top-level port interface for your CLIP. In this example, the LabVIEW FPGA interface has three signals: Clock In, Data In, and Data Out. As you adapt this CLIP XML file to your adapter module, you must remove these signals and add the signals defined in your port map. For more information about `<Signal>` tag syntax, refer to the *Integrating Third-Party IP (FPGA Module)* topic in the *FPGA Module* book of the *LabVIEW Help*.
2. Update the value of the `<HDLName>` tag to match the top-level entity name of your CLIP.
3. Define the paths to each of the source files that provides the implementation for your CLIP. Update the `<ImplementationList>` tag to define these paths. Each of the `<Path>` tags defines a path to a source file, relative to the location of the XML file. Each of the files defined in the `<ImplementationList>` is included in the compilation.
4. Define this socketed CLIP as compatible with your physical adapter module. This information is specified in the `<CompatibleIOModuleList>` tag. This list may have one or more `<IOModule>` tags, each of which identifies a particular adapter module that is compatible with this CLIP. If your adapter module has an adapter module IO Module ID, defined in the `.fam` file and programmed into the EEPROM, you must uniquely identify your adapter module using this value. If your adapter module does not have an IO Module ID, you can still identify it by name. The syntax of the `<IOModule>` value is as follows:

   To identify an adapter module by IO Module ID:
   `IOModuleID:<`*`IO Module ID`* (in hexadecimal)`>`
   For example: `IOModuleID:0xFFFF0001`

   To identify an adapter module by name:
   `Name:<`*`Manufacturer Name`*`>::<`*`Model Name`*`>`
   For example: `Name:Example Manufacturer::Example Module`

5. Rename the <CLIPDeclaration> tag name to an appropriate user-visible name for your socketed CLIP. This name is displayed in the LabVIEW FPGA configuration user interface. Add an appropriate description of the CLIP to the <Description> tag. Rename the CLIP XML file to something appropriate to your CLIP.

For more information about adding a CLIP declaration file to a LabVIEW FPGA project, refer to the *Integrating Third-Party IP (FPGA Module)* topic in the *FPGA Module* book of the *LabVIEW Help*. This topic in the *LabVIEW Help* contains the list of tags that you can use in the declaration file. In addition to these XML tags, FlexRIO adapter module declaration files for socketed CLIP require tags that define the interface between the FPGA and the adapter module circuitry. The following table lists the XML tags used in the socketed CLIP declaration file. The following socketed CLIP XML tags are required in your declaration file.

Table 11-4. Socketed CLIP XML Tags

Tag	Parent Tag	Description
CLIPVersion	CLIPDeclaration	Specifies the version of this CLIP implementation. Use this tag to specify the version of the CLIP as a whole. To specify the CLIPVersion, use the following format: <*major*>.<*minor*>.<*revision*> Example: 1.0.0
Description	CLIPDeclaration	Provides a description of the CLIP visible in the user interface. You may use \n to format the string with end-of-line characters.
CompatibleIO ModuleList	CLIPDeclaration	Identifies the collection of adapter modules compatible with this CLIP. This tag contains one or more IOModule tags.

Table 11-4. Socketed CLIP XML Tags (Continued)

Tag	Parent Tag	Description

IOModule	CompatibleIO ModuleList	Identifies an individual adapter module that this CLIP is compatible with. You can identify the adapter module either by IO Module ID or by Name. If your adapter module has an IO Module ID, always use the IO Module ID for identification, rather than the name.
		• To identify by IO Module ID, use the following syntax:
		`IOModuleID:<IO Module ID>`
		Example: `IOModuleID:0xFFFF0001`
		• To identify by name, use the following syntax:
		`Name:<Manufacturer>::<Model>`
		Example: `Name:Example Manufacturer::Example Module`

CompatibleCLIP SocketList	CLIPDeclaration	Identifies the collection of CLIP sockets compatible with this CLIP.
Socket	CompatibleCLIP SocketList	Identifies an individual CLIP socket compatible with this CLIP. For adapter module socketed CLIP, define this tag as follows: `FlexRIO-IOModule`
SupportedDevice Families	CLIPDeclaration	Identifies which devices families are supported. Valid values are `Virtex-5`. You can also specify `Unlimited` to support all families.

Note To assist with creating FlexRIO adapter module CLIP XML files, National Instruments provides an XML schema file. This file is named `AdapterModuleCLIPDeclaration.xsd` and is installed in the following location:

- **Windows XP**

  `C:\Documents and Settings\All Users\Shared Documents\National Instruments\FlexRIO\Module Development Kit\Design Files`

- **Windows 8/7/Vista**

  `C:\Users\Public\Documents\National Instruments\FlexRIO\Module Development Kit\Design Files`

Configuring the FlexRIO Adapter Module in LabVIEW

Before creating a new LabVIEW project that uses your CLIP, first ensure that your CLIP is stored within the `IO Modules` directory on disk, inside of your manufacturer-specific subfolder that you created earlier. NI recommends that you store your socketed CLIP alongside your adapter module `.fam` or `.tbc` file.

Storing your CLIP within the `IO Modules` directory on disk assists in integrating your adapter module support files within the LabVIEW development environment. LabVIEW automatically discovers any CLIP items stored within the `IO Modules` directory and makes them selectable within your project. Using the `IO Modules` directory also allows you to develop a custom adapter module installer package that automatically installs support for your adapter module.

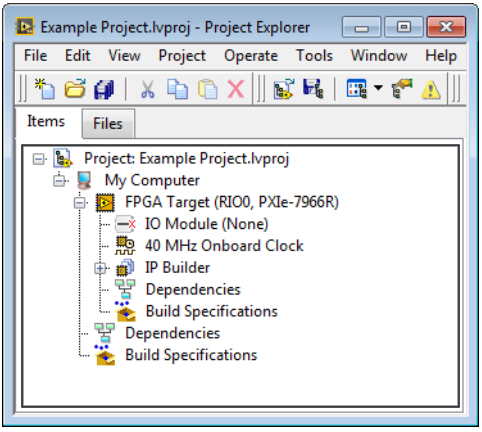
Note The FlexRIO software refers to adapter modules as *IO modules*.

Adding Your Adapter Module and Module I/O in LabVIEW

To use your adapter module with LabVIEW, complete the following steps:

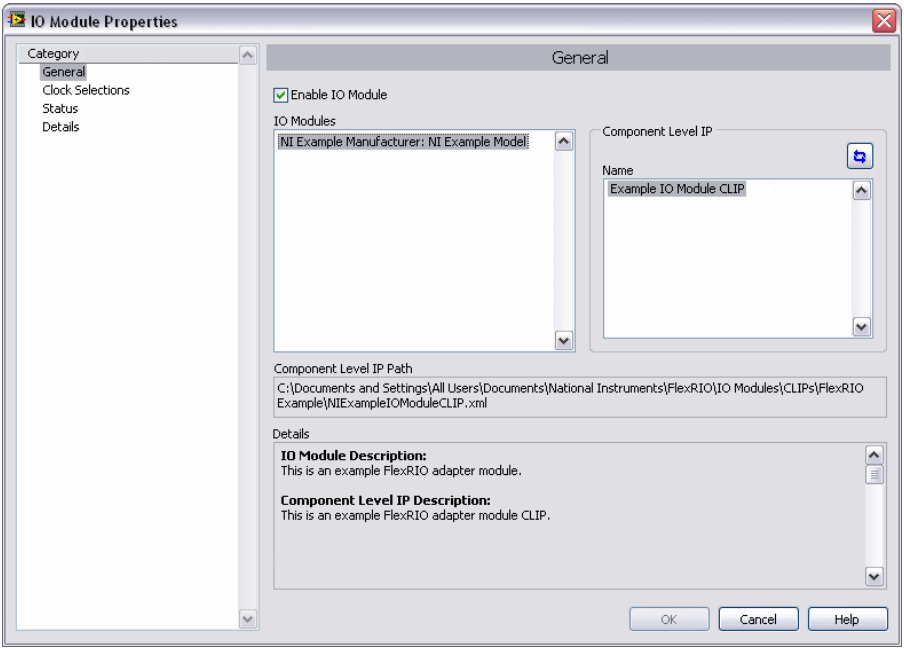
1. Create a new LabVIEW project.
2. Add your FPGA target to the project. The FlexRIO FPGA module appears in the **LabVIEW Project Explorer** window with an unconfigured adapter module.

Figure 11-1. FPGA Target

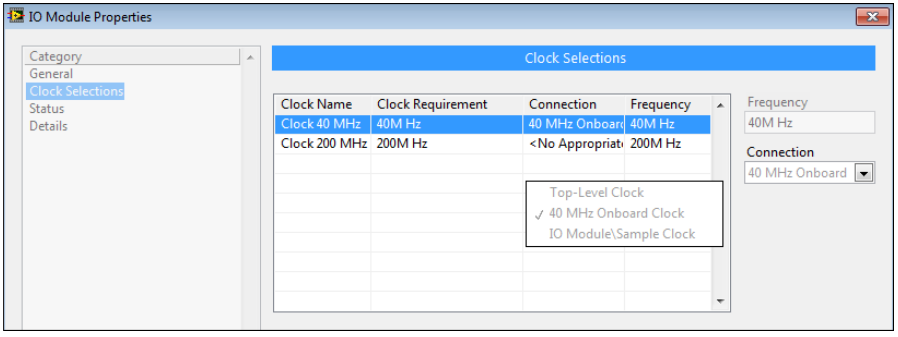


3. To configure your adapter module, right-click the **IO Module** item under the FPGA Target and select **Properties** to display the **IO Module Properties** dialog box.

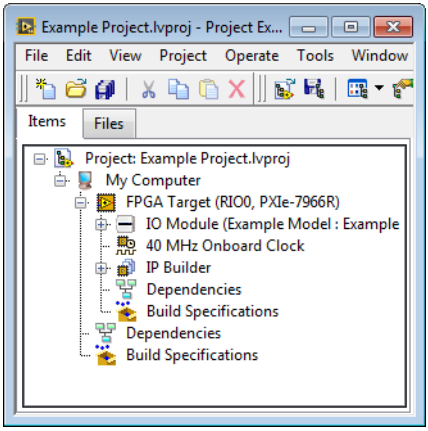
4. Check the box for **Enable IO Module** to enable the adapter module, and select your adapter module from the **IO Modules** list.
5. The adapter module socketed CLIP you created appears in the **Component Level IP** list. Select your adapter module CLIP from the **Component Level IP** list. The adapter module and CLIP descriptions display in the **Details** box. If there were errors with your `.fam` file or the CLIP XML file, syntax errors are displayed. You must correct these errors in order for the items to appear correctly.



6. In the **Clock Selections** category, configure any necessary CLIP clock signals.



7. Click **OK**. Your LabVIEW FPGA I/O displays underneath the IO Module in the project, and your adapter module is now ready for use.

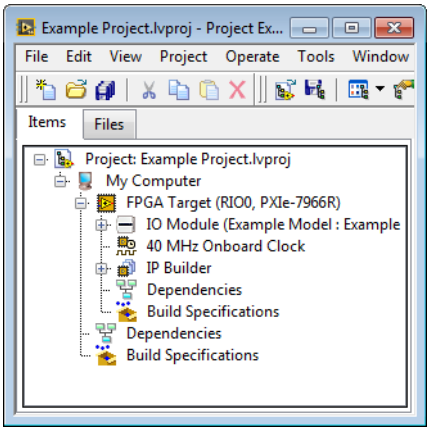


If you were unable to select your adapter module or your CLIP, refer to Appendix B, *Troubleshooting*, for possible solutions.

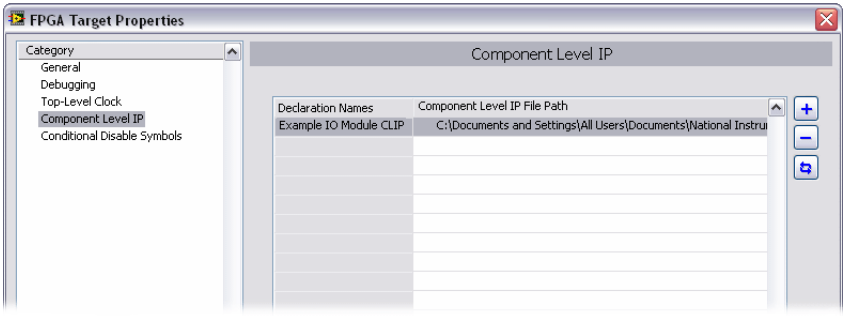
Manually Adding CLIP to Your LabVIEW Project

For convenience, LabVIEW also supports storing your adapter module socketed CLIP in a location external to the `IO Modules` directory. For example, you may wish to store your adapter module CLIP in the same location as your LabVIEW FPGA project files. In this situation, LabVIEW does not automatically discover your adapter module CLIP and you must manually add your adapter module CLIP to your LabVIEW project. To manually add your adapter module CLIP to the LabVIEW project, complete the following steps:

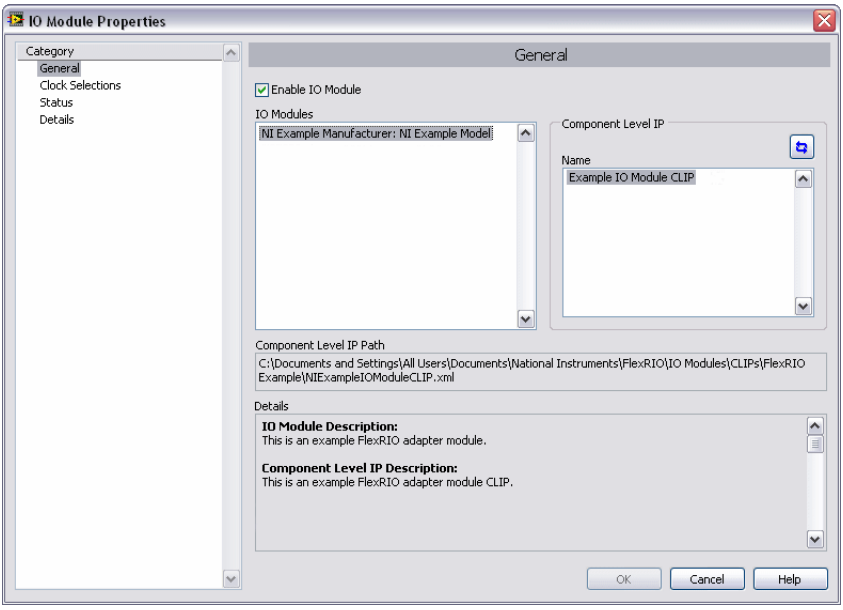
1. Create a new LabVIEW project.
2. Add your FPGA target to the project. The FlexRIO FPGA module appears in the **LabVIEW Project Explorer** window with an unconfigured adapter module.



3. Right-click the FPGA target and select **Properties** from the shortcut menu to display the **FPGA Target Properties** dialog box.
4. Select **Component Level IP** from the **Category** list to display the **Component Level IP FPGA Target Properties** page.
5. Click the **Add** button. A file browser window launches, prompting you for the location of your CLIP XML file. Browse to the XML file location, select it, and click **OK**.



6. Click **OK** on the **FPGA Target Properties** dialog. Your adapter module CLIP has now been added to your FPGA target and is available for selection from within the IO Module Properties dialog box.



Refer to the *Adding Your Adapter Module and Module I/O in LabVIEW* section to configure your adapter module with the CLIP you added to your LabVIEW project.

12

Creating Socketed Component-Level IP for Your Adapter Module and NI-793xR/NI 797xR Modules

To connect external signals from your adapter module to your LabVIEW VI, you need to add your own custom HDL code to the LabVIEW FPGA project. Creating custom socketed component-level IP (CLIP) allows you to program the FPGA to interface with the adapter module circuitry.

FlexRIO devices support two types of CLIP: user-defined and socketed. User-defined CLIP allows you to insert HDL IP into a LabVIEW target, enabling VHDL code to communicate directly with an FPGA VI. Socketed CLIP provides the same IP integration functionality of the user-defined CLIP, while also providing direct communication between the adapter module connector interface and the FPGA VI. For a graphical overview of CLIP and its relationship with FlexRIO hardware, refer to Chapter 2, *FlexRIO Solution Architecture Overview*.

To add CLIP to a LabVIEW project, LabVIEW FPGA Module must support the IP that you use as CLIP. The CLIP must also have an accompanying XML declaration file. The XML declaration file describes the elements of the IP, which LabVIEW FPGA Module uses to add the IP to the LabVIEW project. To use CLIP with your adapter module, complete the following steps:

1. Create or acquire the IP, and, if needed, create optional adapter module IP constraints by setting them in the `.ucf/.xdc` file.
2. Create a CLIP declaration XML file to define the adapter module I/O for LabVIEW FPGA.
3. Configure your adapter module to use CLIP in the LabVIEW project.

These steps are described in greater detail below.

Using the CLIP Wizard

You can also use the CLIP Wizard to help create your custom CLIP or to edit existing CLIP. Complete the following steps to access the CLIP Wizard:

1. Right-click the desired target in the Project Explorer window and select **Properties**.
2. In the Category section of the FPGA Target Properties dialog box, select **Component-Level IP**.

3. On the right side of the FPGA Target Properties dialog box, click the **Create File** icon to create new CLIP or, to edit existing CLIP, select the desired CLIP from the list and click the **Modify File** icon.

Note If your `.fam` or `.tbc` file is in the location listed in the *Creating the Adapter Module Configuration (.fam) File* or *Creating the Adapter Module Configuration (.fam) File* section of Chapter 10, *Configuring Your Adapter Module for Use with NI-793xR/797xR Modules and LabVIEW FPGA*, you can select the configuration values for your adapter module in the CLIP Wizard, which auto-populates your XML declaration file. If you do *not* use the CLIP Wizard, follow the instructions in the *ExampleIOModuleCLIPK7.xml* section of this chapter to create your XML declaration file.

Creating or Acquiring the IP for the FlexRIO Adapter Module

To use socketed CLIP with your adapter module, you must first create or acquire IP, in the form of VHDL code or other HDL code wrapped in VHDL code, to compile into the FPGA target. This IP performs the following tasks:

- Directly controls the FPGA pins on the adapter module connector interface.
- Provides a signal interface to LabVIEW FPGA, which allows LabVIEW to interact with the adapter module. The author of the adapter module CLIP can customize the interface exposed to LabVIEW FPGA.
- Implements any additional adapter module-specific functionality.

To help with adapter module component-level IP development, National Instruments provides example adapter module socketed CLIP files.

The example CLIP for NI 797*x*R devices running on LabVIEW 2013 is composed of the following files:

- `ExampleIOModuleCLIPK7.vhd`
- `ExampleIOModuleCLIPK7.ucf`
- `ExampleIOModuleCLIPK7.xml`

The example CLIP for NI-793*x*R/NI 797*x*R devices running on LabVIEW 2014 and later is composed of the following files:

Note FlexRIO Support 15.1 is the earliest version of FlexRIO Support that supports the NI-793*x*R devices.

- `ExampleIOModuleCLIPK7.vhd`
- `ExampleIOModuleCLIPK7.xdc`
- `ExampleIOModuleCLIPK7.xml`

The example socketed CLIP files are installed in the following location:

- **Windows XP**

  `C:\Documents and Settings\All Users\Shared Documents\National Instruments\FlexRIO\Module Development Kit\Examples\IO Module`

- **Windows 8/7/Vista**

  `C:\Users\Public\Documents\National Instruments\FlexRIO\Module Development Kit\Examples\IO Module`

The following sections describe the process of creating a new adapter module socketed CLIP using the example adapter module CLIP. For additional information about creating CLIP, refer to the *Integrating Third-Party IP (FPGA Module)* and the *FlexRIO Help* books of the *FPGA Module* in the *LabVIEW Help*.

Tip When developing new socketed CLIP from the provided example files, first copy the example adapter module CLIP files to another location on your disk drive so that you can safely modify them.

ExampleIOModuleCLIPK7.vhd

`ExampleIOModuleCLIPK7.vhd` is the top-level VHDL file of the socketed CLIP and defines the top-level port interface to the CLIP. LabVIEW FPGA uses this interface when compiling the CLIP into LabVIEW FPGA. This section explains the changes you must make to the example file in order to adapt this CLIP to your adapter module.

Incorporate your IP into this VHDL file, writing new VHDL code or instantiating existing IP as necessary. The adapter module connector I/O signals are listed in the ExampleIOModuleCLIP port interface. These signals are the fixed interface to the adapter module and are therefore required in the port interface for your adapter module CLIP. Tables 12-1, 12-2, and 12-3 list the additional signals available for use in your CLIP interface.

Table 12-1. GPIO and CLK Signals from Adapter Module

Signal Name	Direction	Data Type	Description
aUserGpio	Bidirectional	std_logic_ vector (67 downto 0)	68 GPIO FPGA pins (non-inverted).
aUserGpio_n	Bidirectional	std_logic_ vector (67 downto 0)	68 GPIO FPGA pins (inverted).
rIoModGpioEn	To CLIP	std_logic	Adapter module GPIO buffer enable signal. When this signal is logic high, it is safe to enable the buffers for the GPIO signals. When this signal is logic low, the I/O buffers for the GPIO signals must be tristated.

Table 12-2. I^2C Core Interface Signals*

Signal Name	Direction	Data Type	Description
rLvFpgaReqI2cBus	From CLIP	std_logic	Drive a `1` to this port to request control of the shared I^2C bus. Only request the I^2C bus if the adapter module is properly identified and powered. When you finish accessing the I^2C bus, you must always release access to the I^2C bus.
rLvFpgaAckI2cBus	To CLIP	std_logic	This signal indicates a `1` when I^2C bus access is granted after you assert rLvFpgaReqI2cBus. Commands sent to the I^2C controller are ignored if this signal indicates a `0`.

Table 12-2. I²C Core Interface Signals* (Continued)

Signal Name	Direction	Data Type	Description
rLvFpgaI2cGo	From CLIP	std_logic	Drive a `1` to this signal to send a command to the I²C controller. You must configure all I²C controller inputs before driving this signal high. The I²C controller latches the command input when it detects a rising edge on this signal and begins the I²C transaction. Do not reassert this signal until LvFPGAI2CDone indicates a `1`.
rLvFpgaI2cStart	From CLIP	std_logic	Drive a `1` to this signal to send an I²C start pattern at the beginning of your I²C transaction.
rLvFpgaI2cStop	From CLIP	std_logic	Drive a `1` to this signal to send an I²C stop pattern at the end of your I²C transaction.
rLvFpgaI2cRd	From CLIP	std_logic	Drive a `1` to this signal to indicate a read transaction, meaning the I²C controller returns RdData from the bus. Drive a `0` to this signal to indicate a write transaction, meaning the I²C controller drives the WtData value to the bus.
rLvFpgaI2cWtData	From CLIP	std_logic_ vector (7 downto 0)	This is the 8-bit data value written to the bus by the I²C controller during a Write transaction.
rLvFpgaI2cAck	To CLIP	std_logic	Valid after an I²C transaction. A `1` indicates that the I²C controller read an Ack from the bus.
rClktoSocket	To CLIP	std_logic	This is a copy of the 40MHz RioClk used by default in LV FPGA diagrams. All I²C signals—denoted by the *r* prefix—are synchronous to this clock.

Table 12-2. I²C Core Interface Signals* (Continued)

Signal Name	Direction	Data Type	Description
rLvFpgaI2cDone	To CLIP	std_logic	A `1` indicates that the I²C controller is done. Wait for `1` before reading the outputs.
rLvFpgaI2cRdData	To CLIP	std_logic_ vector (7 downto 0)	This is the 8-bit data value read from the I²C bus during a Read transaction. Read this value after the completion of a read transaction. A `0` indicates that the bus is busy.
* All I2C core interface signals are synchronized to the rClkToSocket signal.			

The signals in the following table interact with the Time to Digital Converter (TDC) on the NI-793*x*R/NI 797*x*R modules. They are reserved for future use.

Note Leave all inputs open or unconnected.

Note Drive all outputs with a `0`.

Table 12-3. TDC Circuitry

Signal Name	Direction	Data Type	Description
TdcAssertClk	To CLIP	std_logic	Reserved for future use.
tTdcAssert	From CLIP	std_logic	
Clk100	To CLIP	std_logic	
c100TdcDeassert	From CLIP	std_logic	
rTdcPulseWidth	To CLIP	std_logic_vector (15 downto 0)	
rTdcPulseWidthValid	To CLIP	std_logic	

Using External Clocks

When importing an external clock for use in your adapter module CLIP and/or in your LabVIEW FPGA VI, you must consider several factors. These considerations apply to CLIP logic and UCF constraints covering these incoming clock signals. Refer to the *NI-793xR and NI 797xR Signal Descriptions* section of Chapter 4, *Interfacing Adapter Modules with NI-793xR and NI 797xR Devices*, for a list of the clock-capable adapter module GPIO interface pins.

NI requires that you include the following logic considerations within any CLIP clocking interface:

- Any clock shared with the LV FPGA VI that is exported from the socketed CLIP must be stable and free running at all times. If these clocks are unstable, such as during a power-up or reset, you *must not* drive them to LabVIEW FPGA generated logic. Use a clock buffer with an "enable" control to gate an unstable clock, such as a BUFGCE.
- All LV FPGA clocks are stable and free running unless you assert a Reset. After you assert a Reset, assert any DCM/PLL logic using these clocks. Do not drive unstable clocks to LabVIEW FPGA logic or any other logic.

Note A Reset may be held for as little as one clock cycle. If your logic requires a longer hold during Reset, you should implement application appropriate logic, such as a shift register, to define the necessary hold time.

NI recommends also incorporating the following considerations into your CLIP clocking interface.

- Use a global clock buffer (BUFG) or regional clock buffer (BUFR) to drive an incoming clock onto the FPGA clock network. For more information about the differences between clock buffer options and the limitations of each, refer to Xilinx documentation, available at www.xilinx.com. For a list of Xilinx documents applicable to FlexRIO applications, refer to Appendix C, *Xilinx Documentation References*.
- Add OFFSET timing constraints for all adapter module signals captured or driven with the external clock.
- Any LabVIEW FPGA controls and indicators placed in external clock domains could result in warnings if you access them while the external clock is disabled. If you require controls or indicators in external clock domains, structure host code to access these controls or indicators only when the external clock is running.

ExampleIOModuleCLIPK7.ucf

Note The ExampleIOModuleCLIPK7.ucf file provides constraints for the NI 797*x*R devices used with LabVIEW 2013.

Each IP implementation may require custom constraints settings. Each adapter module CLIP implementation may optionally be packaged with a constraints file which is used during the compilation process. This constraints file should generally provide timing-related constraints only. Do not include constraints for the FPGA I/O pin signal standards in the adapter module CLIP constraints file.

The FPGA I/O pin signal standards are constrained in the adapter module configuration file (`.fam`). Refer to Chapter 10, *Configuring Your Adapter Module for Use with NI-793xR/797xR Modules and LabVIEW FPGA*, for information about the pin signal constraints.

Note To ensure correct compilation of your adapter module constraints information, NI recommends using `()` as the bus delimiters in both the `.ucf` and the `.fam` files, instead of using <>. For example: `aUserGpio(*)`.

Note FlexRIO CLIPs are not instantiated as part of the top-level design. NI recommends using `%CLIPInstancePath%` in your constraints information to ensure that the Xilinx compiler can locate the signal name. If you do not direct the compiler to the signal using `%CLIPInstancePath%`, the compiler may return an error.

If you developed your CLIP using the provided example CLIP, modify the example adapter module constraints file to properly constrain your IP and rename the file to something more appropriate for your adapter module. For more information about timing constraints, refer to Xilinx documentation, available at `www.xilinx.com`. For a list of Xilinx documents applicable to FlexRIO applications, refer to Appendix C, *Xilinx Documentation References*.

ExampleIOModuleCLIPK7.xdc

Note The ExampleIOModuleCLIPK7.xdc file provides constraints for the NI-793*x*R/NI 797*x*R devices used with LabVIEW 2014 and later.

Note FlexRIO Support 15.1 is the earliest version of FlexRIO Support that supports the NI-793*x*R devices.

Each IP implementation may require custom constraints settings. Each adapter module CLIP implementation requires a `.xdc` file that accompanies a `.fam` file and an additional optional `.xdc` file that may accompany the `.vhd` files. The `.xdc` file that accompanies the `.fam` file should contain all of the FPGA I/O pin signal standards. The additional `.xdc` files generally provide timing-related constraints only, and do not include constraints for the FPGA I/O pin signal standards.

Refer to Chapter 10, *Configuring Your Adapter Module for Use with NI-793xR/797xR Modules and LabVIEW FPGA*, for information about the pin signal constraints.

Note FlexRIO CLIPs are not instantiated as part of the top-level design. NI recommends using `%CLIPInstancePath%` in your constraints information to ensure that the Xilinx compiler can locate the signal name. If you do not direct the compiler to the signal using `%CLIPInstancePath%`, the compiler may return an error.

If you developed your CLIP using the provided example CLIP, modify the example adapter module constraints file to properly constrain your IP and rename the file to something more appropriate for your adapter module. For more information about timing constraints, refer to Xilinx documentation, available at `www.xilinx.com`. For a list of Xilinx documents applicable to FlexRIO applications, refer to Appendix C, *Xilinx Documentation References*.

ExampleIOModuleCLIPK7.xml

Note You only need to complete the steps in this section if you did *not* use the CLIP Wizard to create your CLIP. The CLIP Wizard auto-populates this XML declaration file based on the information that you select in the CLIP Wizard.

Note The ExampleIOModuleCLIPK7.xml file provides an example of a CLIP declaration file for the NI-793*x*R/NI 797*x*R devices.

The final step of creating a new adapter module socketed CLIP is to create a CLIP declaration file. The CLIP declaration file performs the following tasks:

- Identifies paths to each of the source files that provide the implementation for your CLIP.
- Defines the top-level entity name for your CLIP.
- Defines the collection of signals that are exposed both to the LabVIEW FPGA side of the CLIP interface and to the hardware socketed side of the CLIP interface.
- Defines the name of the CLIP socket(s) compatible with this CLIP.
- Defines the collection of adapter modules compatible with this CLIP.
- Describes and defines how your VHDL ports appear in the LabVIEW FPGA project.

Complete the following steps to adapt the example adapter module socketed CLIP XML file to your new adapter module CLIP.

1. Add the LabVIEW FPGA port interface signals to the list of interface signals in your XML file. All CLIP declaration XML files contain an `<InterfaceList>` section which defines three different types of interfaces: `Fabric`, `LabVIEW`, and `Socket`. Each of these interface types is denoted by specifying the appropriate value for `<InterfaceType>` within a named `<Interface>` section.

   For example, the `ExampleIOModuleCLIPK7.xml` file defines three types of interfaces. One of the interfaces in your CLIP XML file must define each signal in the top-level port interface for your socketed CLIP. The `Fabric` and `Socket` interfaces, however, are fixed for all adapter module CLIP items and cannot be changed. You can customize the LabVIEW FPGA interface. The interface must account for all LabVIEW FPGA CLIP signals that you define in the top-level port interface for your CLIP. In this example, the LabVIEW FPGA interface has three signals: Clock In, Data In, and Data Out. As you adapt this CLIP XML file to your adapter module, you must remove these signals and add the signals defined in your port map. For more information about `<Signal>` tag syntax, refer to the *Integrating Third-Party IP (FPGA Module)* topic in the *FPGA Module* book of the *LabVIEW Help*.
2. Update the value of the `<HDLName>` tag to match the top-level entity name of your CLIP.
3. Define the paths to each of the source files that provides the implementation for your CLIP. Update the `<ImplementationList>` tag to define these paths. Each of the `<Path>` tags defines a path to a source file, relative to the location of the XML file. Each of the files defined in the `<ImplementationList>` is included in the compilation.

4. Define this socketed CLIP as compatible with your physical adapter module. This information is specified in the <CompatibleIOModuleList> tag. This list may have one or more <IOModule> tags, each of which identifies a particular adapter module that is compatible with this CLIP. If your adapter module has an adapter module IO Module ID, defined in the .fam file and programmed into the EEPROM, you must uniquely identify your adapter module using this value. If your adapter module does not have an IO Module ID, you can still identify it by name. The syntax of the <IOModule> value is as follows:

   To identify an adapter module by IO Module ID:
   IOModuleID:<*IO Module ID* (in hexadecimal)>
   For example: IOModuleID:0xFFFF0001

   To identify an adapter module by name:
   Name:<*Manufacturer Name*>::<*Model Name*>
   For example: Name:Example Manufacturer::Example Module

5. Rename the <CLIPDeclaration> tag name to an appropriate user-visible name for your socketed CLIP. This name is displayed in the LabVIEW FPGA configuration user interface. Add an appropriate description of the CLIP to the <Description> tag. Rename the CLIP XML file to something appropriate to your CLIP.

For more information about adding a CLIP declaration file to a LabVIEW FPGA project, refer to the *Integrating Third-Party IP (FPGA Module)* topic in the *FPGA Module* book of the *LabVIEW Help*. This topic in the *LabVIEW Help* contains the list of tags that you can use in the declaration file. In addition to these XML tags, FlexRIO adapter module declaration files for socketed CLIP require tags that define the interface between the FPGA and the adapter module circuitry. The following table lists the XML tags used in the socketed CLIP declaration file. The following socketed CLIP XML tags are required in your declaration file.

Table 12-4. Socketed CLIP XML Tags

Tag	Parent Tag	Description
CLIPVersion	CLIPDeclaration	Specifies the version of this CLIP implementation. Use this tag to specify the version of the CLIP as a whole. To specify the CLIPVersion, use the following format: <*major*>.<*minor*>.<*revision*> Example: 1.0.0
Description	CLIPDeclaration	Provides a description of the CLIP visible in the user interface. You may use \n to format the string with end-of-line characters.

Table 12-4. Socketed CLIP XML Tags (Continued)

Tag	Parent Tag	Description
CompatibleIO ModuleList	CLIPDeclaration	Identifies the collection of adapter modules compatible with this CLIP. This tag contains one or more `IOModule` tags.

IOModule	CompatibleIO ModuleList	Identifies an individual adapter module that this CLIP is compatible with. You can identify the adapter module either by IO Module ID or by Name. If your adapter module has an IO Module ID, always use the IO Module ID for identification, rather than the name.
		• To identify by IO Module ID, use the following syntax:
		`IOModuleID:<IO Module ID>`
		Example: `IOModuleID:0xFFFF0001`
		• To identify by name, use the following syntax:
		`Name:<Manufacturer>::<Model>`
		Example: `Name:Example Manufacturer::Example Module`

CompatibleCLIP SocketList	CLIPDeclaration	Identifies the collection of CLIP sockets compatible with this CLIP.
Socket	CompatibleCLIP SocketList	Identifies an individual CLIP socket compatible with this CLIP. For adapter module socketed CLIP, define this tag as follows: `FlexRIO-K7IOModule`
SupportedDevice Families	CLIPDeclaration	Identifies which devices families are supported. Valid values are `Kintex-7`. You can also specify `Unlimited` to support all families.

Note To assist with creating FlexRIO adapter module CLIP XML files, National Instruments provides an XML schema file. This file is named `AdapterModuleCLIPDeclaration.xsd` and is installed in the following location:

- **Windows XP**

  `C:\Documents and Settings\All Users\Shared Documents\National Instruments\FlexRIO\Module Development Kit\Design Files`

- **Windows 8/7/Vista**

  `C:\Users\Public\Documents\National Instruments\FlexRIO\Module Development Kit\Design Files`

Configuring the FlexRIO Adapter Module in LabVIEW

Before creating a new LabVIEW project that uses your CLIP, first ensure that your CLIP is stored within the `IO Modules` directory on disk, inside of your manufacturer-specific subfolder that you created earlier.

Storing your CLIP within the `IO Modules` directory on disk assists in integrating your adapter module support files within the LabVIEW development environment. LabVIEW automatically discovers any CLIP items stored within the `IO Modules` directory and makes them selectable within your project. Using the `IO Modules` directory also allows you to develop a custom adapter module installer package that automatically installs support for your adapter module.

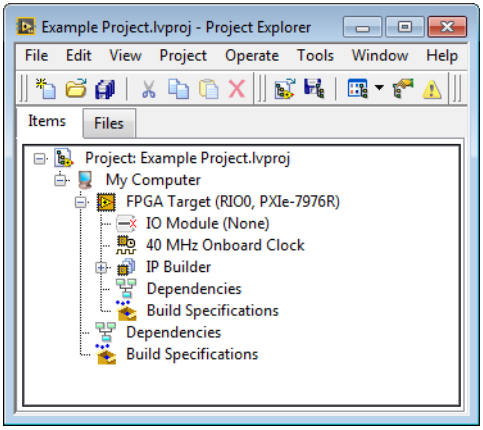
Note The FlexRIO software refers to adapter modules as *IO modules*.

Adding Your Adapter Module and Module I/O in LabVIEW

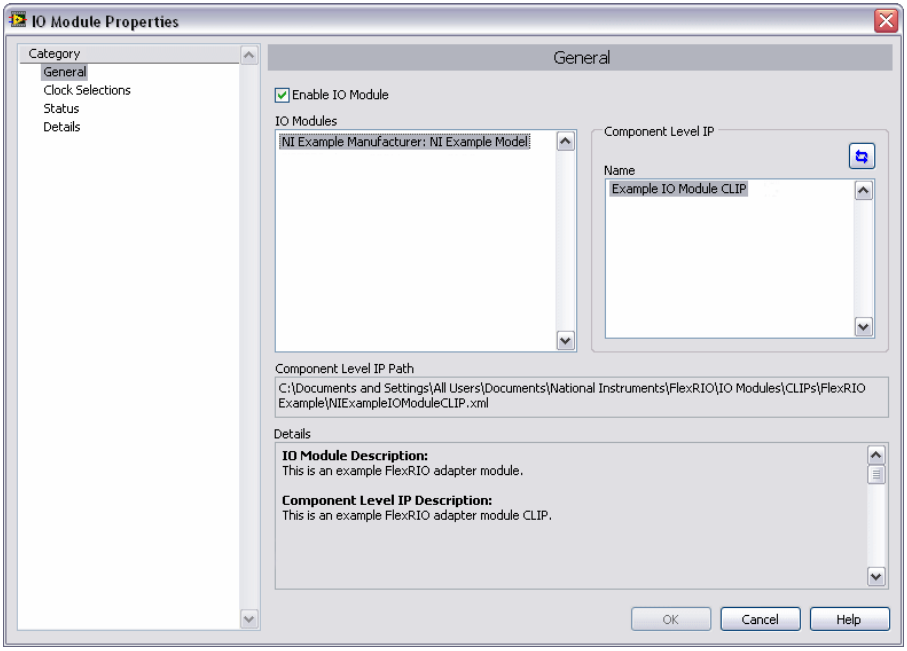
To use your adapter module with LabVIEW, complete the following steps:

1. Create a new LabVIEW project.
2. Add your FPGA target to the project. The FlexRIO FPGA module appears in the **LabVIEW Project Explorer** window with an unconfigured adapter module.

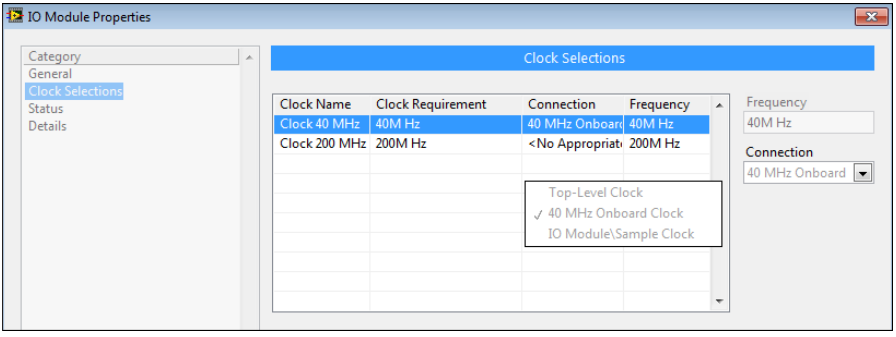
Figure 12-1. FPGA Target



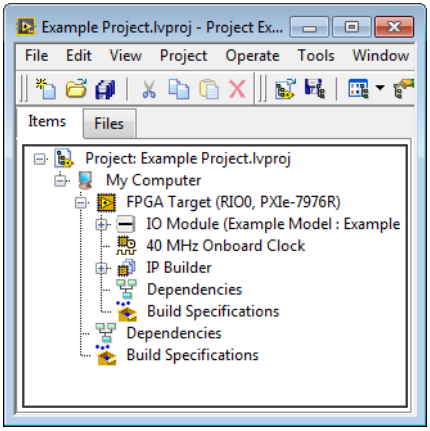
3. To configure your adapter module, right-click the **IO Module** item under the FPGA Target and select **Properties** to display the **IO Module Properties** dialog box.
4. Check the box for **Enable IO Module** to enable the adapter module, and select your adapter module from the **IO Modules** list.
5. The adapter module socketed CLIP you created appears in the **Component Level IP** list. Select your adapter module CLIP from the **Component Level IP** list. The adapter module and CLIP descriptions display in the **Details** box. If there were errors with your `.fam` file or the CLIP XML file, syntax errors are displayed. You must correct these errors in order for the items to appear correctly.



6. In the **Clock Selections** category, configure any necessary CLIP clock signals.



7. Click **OK**. Your LabVIEW FPGA I/O displays underneath the IO Module in the project, and your adapter module is now ready for use.

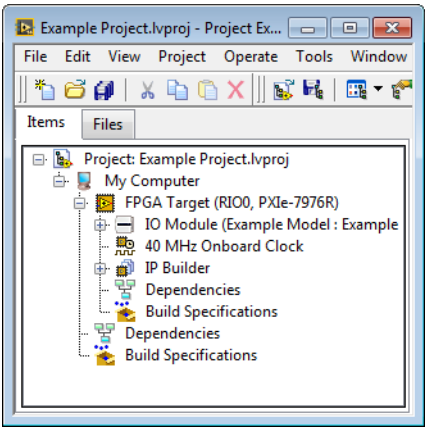


If you were unable to select your adapter module or your CLIP, refer to Appendix B, *Troubleshooting*, for possible solutions.

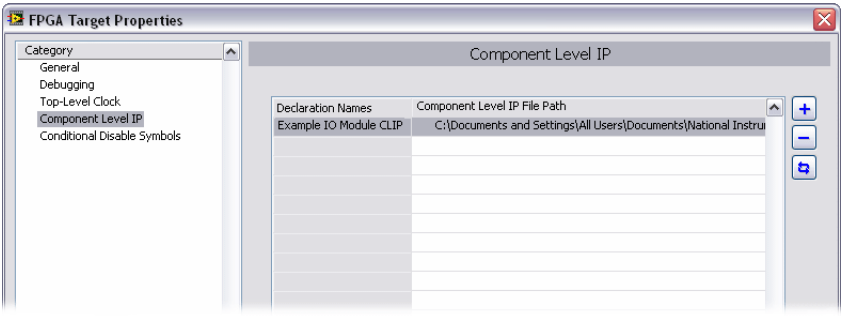
Manually Adding CLIP to Your LabVIEW Project

For convenience, LabVIEW also supports storing your adapter module socketed CLIP in a location external to the `IO Modules` directory. For example, you may wish to store your adapter module CLIP in the same location as your LabVIEW FPGA project files. In this situation, LabVIEW does not automatically discover your adapter module CLIP and you must manually add your adapter module CLIP to your LabVIEW project. To manually add your adapter module CLIP to the LabVIEW project, complete the following steps:

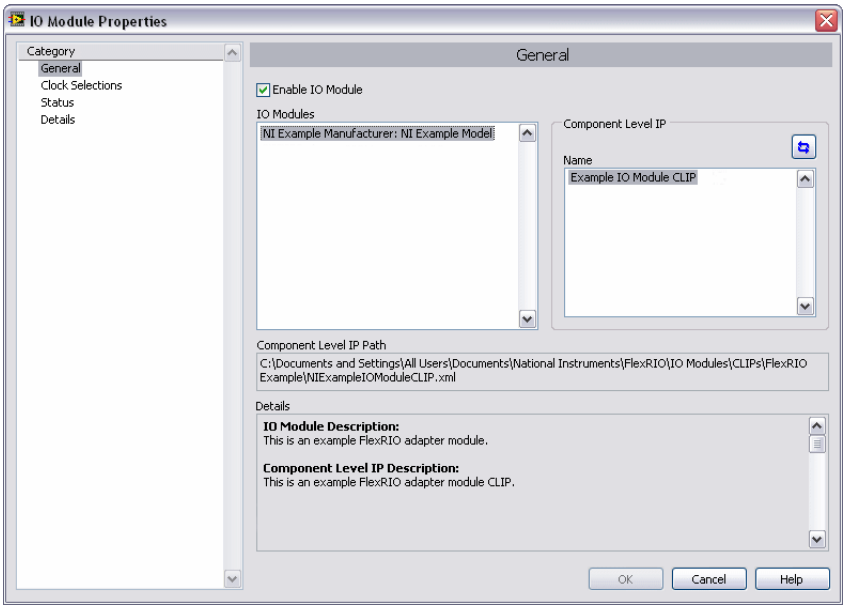
1. Create a new LabVIEW project.
2. Add your FPGA target to the project. Your device appears in the **LabVIEW Project Explorer** window with an unconfigured adapter module.



3. Right-click the FPGA target and select **Properties** from the shortcut menu to display the **FPGA Target Properties** dialog box.
4. Select **Component Level IP** from the **Category** list to display the **Component Level IP FPGA Target Properties** page.
5. Click the **Add** button. A file browser window launches, prompting you for the location of your CLIP XML file. Browse to the XML file location, select it, and click **OK**.



6. Click **OK** on the **FPGA Target Properties** dialog. Your adapter module CLIP has now been added to your FPGA target and is available for selection from within the IO Module Properties dialog box.



Refer to the *Adding Your Adapter Module and Module I/O in LabVIEW* section to configure your adapter module with the CLIP you added to your LabVIEW project.

13

Designing and Debugging Component-Level IP

This chapter explores best practices for designing the CLIP and provides debugging tips for issues that commonly occur during development. For more information about designing and debugging CLIP, refer to Chapter 11, *Creating Socketed Component-Level IP for Your Adapter Module and NI 795xR/796xR Modules* (NI 795xR and NI 796xR), and Chapter 12, *Creating Socketed Component-Level IP for Your Adapter Module and NI-793xR/NI 797xR Modules* (NI 797xR and NI-793xR), of this manual, as well as the *Integrating Third-Party IP (FPGA Module)* book in the *LabVIEW Help*.

Synchronous vs Asynchronous Interfaces

Data can pass either synchronously or asynchronously across the boundary separating LabVIEW FPGA from the CLIP.

Synchronous interfaces have the following characteristics.

- Data is sent and received using the same clock
- Data integrity is ensured

Asynchronous interfaces have the following characteristics.

- Data is sent and received using different clocks
- Data transferring from a high frequency clock domain to a low frequency clock domain can be lost

Multi-bit interfaces should always be synchronous. Asynchronous multi-bit interfaces should be used only under the following circumstances:

- The interface does not have specific arrival time requirements
- The interface data does not need to arrive deterministically
- The interface is used in multiple different clock domains

NI strongly recommends that your designs use only synchronous LabVIEW FPGA/CLIP interfaces due to the difficulty analyzing and constraining asynchronous LabVIEW FPGA/CLIP interfaces. Synchronous interfaces provide the following benefits.

- LabVIEW FPGA safely removes synchronization registers, which guard against metastability. This reduces latency introduced by the synchronization registers. Refer to the Passing Data Between Component-Level IP and VIs topic in the *LabVIEW Help*
- No additional constraints are required to successfully compile

If an asynchronous interface for a signal, such as a trigger, is necessary, it should be used in moderation. For more information about using asynchronous interfaces, refer to the *Considerations for Asynchronous Data Interfaces* section.

Instead of relying on LabVIEW FPGA to synchronize the data as it crosses to or from the CLIP, NI recommends including all clock domain crossings within the CLIP. This forces the LabVIEW FPGA/CLIP interface to be synchronous, greatly reducing the design complexity.

Figure 13-1 illustrates two D-type flip-flops asynchronously sending a signal from the CLIP to LabVIEW FPGA.

Figure 13-1. Asynchronous Interfaces Between LabVIEW FPGA and CLIP

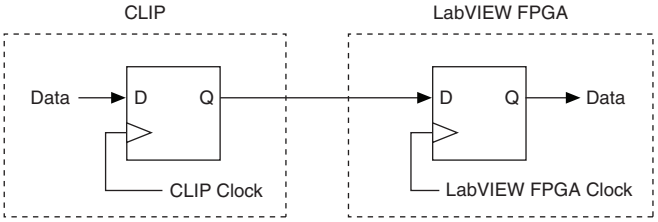


Figure 13-2 illustrates two flip-flops sending data synchronously from the CLIP into LabVIEW FPGA, while Figure 13-3 illustrates two flip-flops sending data synchronously from LabVIEW FPGA to the CLIP. Note that the clock driving the logic on both sides of the path can originate from either the CLIP or LabVIEW FPGA.

Figure 13-2. Interfaces Synchronous to CLIP Clock

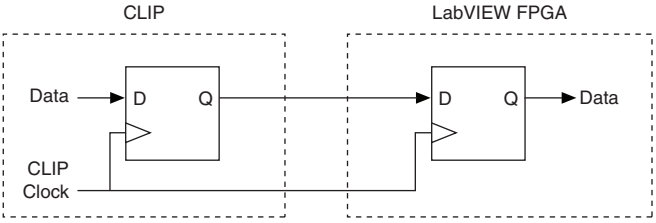
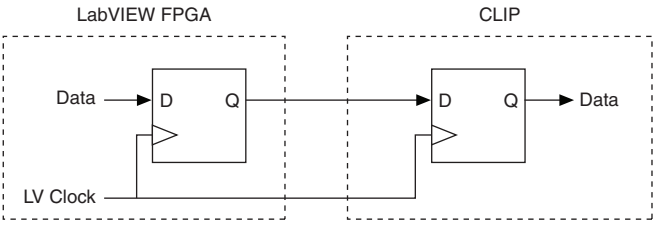


Figure 13-3. Interfaces Synchronous to LabVIEW FPGA Clock



Defining Synchronous CLIP Interfaces

The following section explains how to implement a synchronous CLIP interface. For information about adding data interfaces to the list of I/O that the CLIP exposes to LabVIEW, refer to Chapter 11, *Creating Socketed Component-Level IP for Your Adapter Module and NI 795xR/796xR Modules*, (NI 795*x*R and NI 796*x*R devices) or Chapter 12, *Creating Socketed Component-Level IP for Your Adapter Module and NI-793xR/NI 797xR Modules*, (NI 797*x*R or NI-793*x*R devices) of this manual, or to the *CLIP Tutorial: Adding Component-Level IP to an FPGA Project (FPGA Module)* topic of the *LabVIEW Help*.

Implementing a synchronous interface to communicate between LabVIEW FPGA and the CLIP requires properly creating the CLIP VHDL, CLIP XML, LabVIEW project, and the FPGA VI.

Configuring the Top-Level CLIP HDL File

The following HDL snippet demonstrates how to drive the synchronous interface using a LabVIEW FPGA clock (LVClock). You must import the clock into the CLIP as part of the port map. LVClock then latches the data from LabVIEW FPGA with sLVDataOut. The data returning to LabVIEW FPGA (sLVDataIn) is the output of a flip-flop that is driven by LVClock.

```
SynchronousLabVIEWData: process (aResetSl, LVClock)
begin
    if aResetSl = '1' then
        sLVDataIn <= (others => '0');
    elsif rising_edge(LVClock) then
        sLVDataIn <= sLVDataOut;
    end if;
end process;
```

The following HDL snippet demonstrates how to drive the synchronous interface using a clock originating in the CLIP (ClipClock). You must export the ClipClock through the CLIP port map. The data to and from LabVIEW FPGA, called sClipDataIn and sClipDataOut, respectively, are latched using ClipClock.

```
SynchronousCLIPData: process (aResetSl, ClipClock)
begin
    if aResetSl = '1' then
        sClipDataIn <= (others => '0');
    elsif rising_edge(ClipClock) then
        sClipDataIn <= sClipDataOut;
    end if;
end process;
```

Creating the CLIP XML

The top-level VHDL port map defines the inputs and outputs of the CLIP. The inputs and outputs are further defined and constrained in the CLIP XML, which LabVIEW FPGA uses to

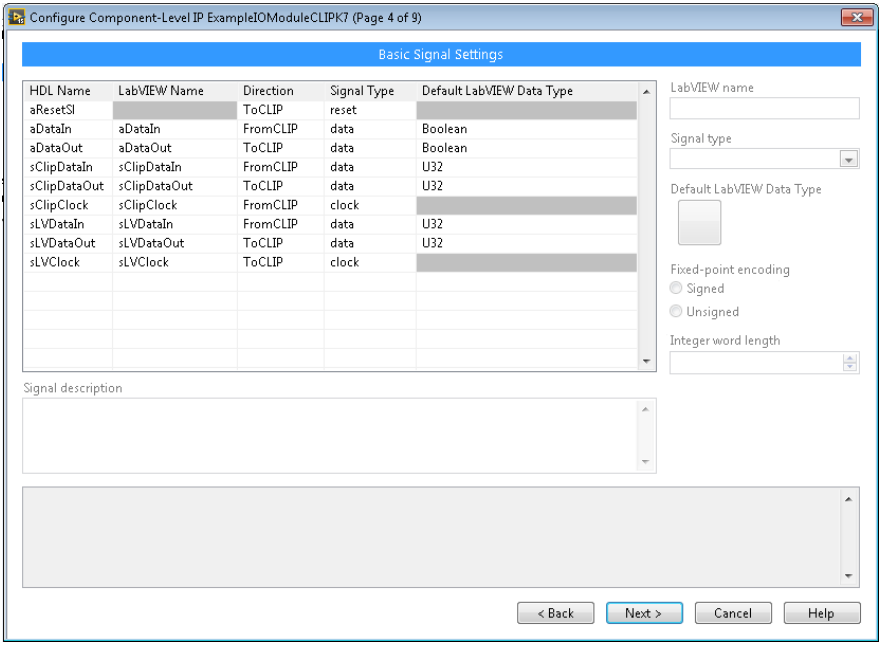
implement the interface. NI recommends using the CLIP Wizard to generate the XML. For more information about using the CLIP Wizard, refer to the *Using the CLIP Wizard* section of Chapter 11, *Creating Socketed Component-Level IP for Your Adapter Module and NI 795xR/796xR Modules* (NI 795*x*R and NI 796*x*R devices), or the *Using the CLIP Wizard* section of Chapter 12, *Creating Socketed Component-Level IP for Your Adapter Module and NI-793xR/NI 797xR Modules* (NI 797*x*R or NI-793*x*R devices), of this manual, or to the *Using the Configure Component-Level IP Wizard (FPGA Module)* topic of the *LabVIEW Help*.

In the CLIP Wizard, the VHDL inputs and outputs can be defined as a reset, data, or clock.

- Reset signals are connected to LabVIEW's global FPGA reset.
- Data signals are made into LabVIEW FPGA I/O Nodes.
- Clocks are connected to or made into LabVIEW FPGA clock resources in the LabVIEW project.

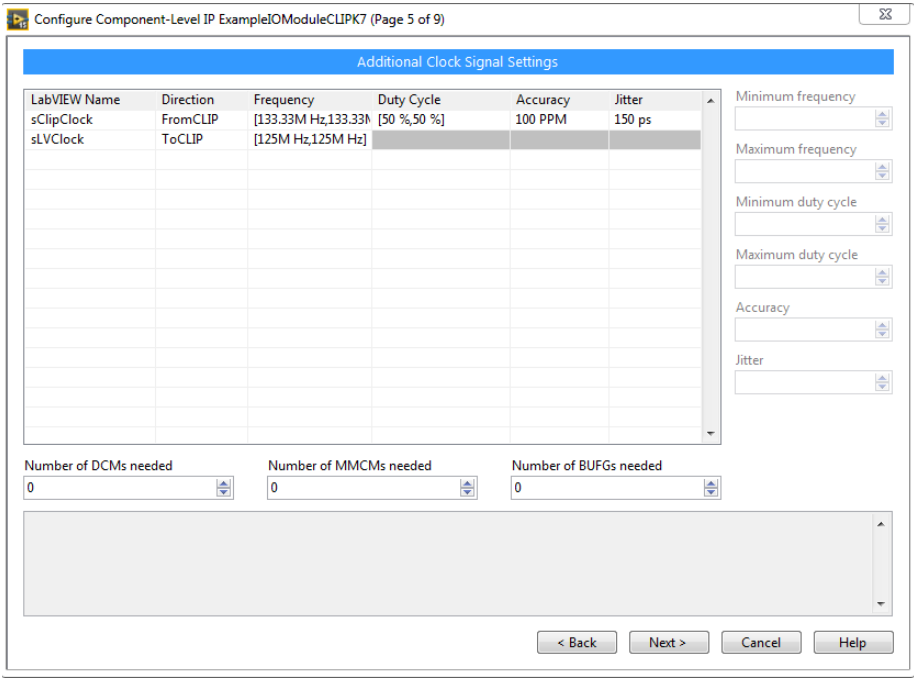
The signals are configured on Page 4 of the CLIP Wizard, as shown in Figure 13-4.

Figure 13-4. Signal Definition in the XML Wizard



The clocks can be further constrained with specific frequency ranges, duty cycle, accuracy, and jitter on Page 5 of the CLIP Wizard, as shown in Figure 13-5.

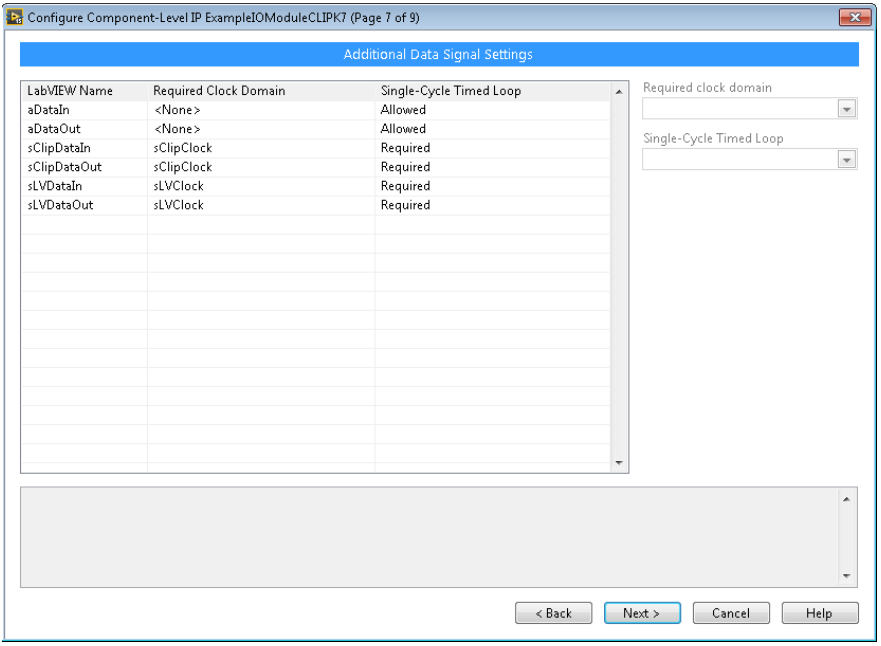
Figure 13-5. Clock Constraints in the XML Wizard



Finally, the data lines can be constrained on Page 7 of the CLIP Wizard to be placed inside single-cycle timed loops (SCTL) driven by specific clocks. Figure 13-6 shows that the sLVData and sClipData I/O Nodes must be placed in the sLVClock and sClipClock domains, respectively. This matches the above VHDL implementation.

Note The aData signals are driven by sClipClock in the VHDL and sLVClock in LabVIEW, making the interface asynchronous.

Figure 13-6. Signal Clock Domain Constraints in the XML Wizard



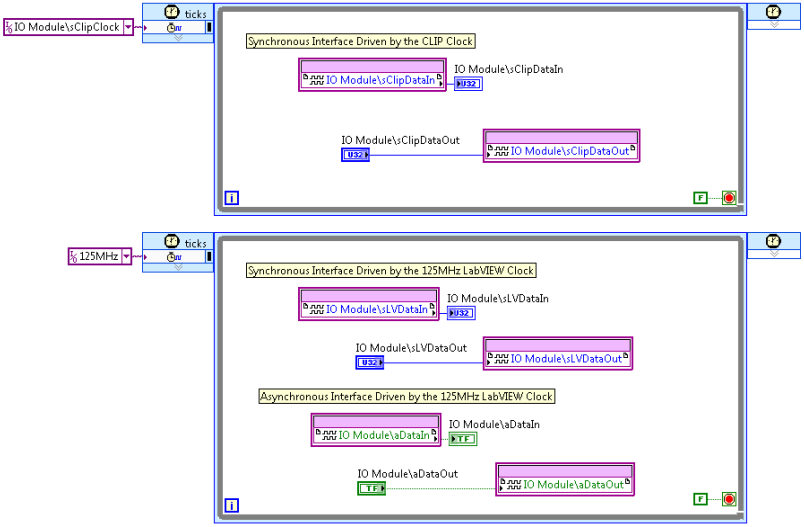
Integrating the CLIP into LabVIEW

In the LabVIEW project, you must configure the adapter module to use your CLIP and route any clocks from LabVIEW to the CLIP. For more information, refer to the *Configuring the FlexRIO Adapter Module in LabVIEW* sections in Chapter 11, *Creating Socketed Component-Level IP for Your Adapter Module and NI 795xR/796xR Modules*, (NI 795*x*R and NI 796*x*R devices) or Chapter 12, *Creating Socketed Component-Level IP for Your Adapter Module and NI-793xR/NI 797xR Modules* (NI 797*x*R or NI-793*x*R devices).

Note The **Clock Selections IO Module Properties** page populates with any clock from LabVIEW to the CLIP, allowing you to route any derived FPGA clock to the adapter module.

The FPGA VI can be populated with SCTLs and I/O nodes. The compiler forces you to follow the constraints set in the XML file and the **IO Module Properties** page.

Figure 13-7. LabVIEW FPGA VI Implementing Synchronous and Asynchronous Interfaces



Refer to the *Updating the LabVIEW Project to Reflect Changes in the CLIP XML* section of this chapter for more information.

Considerations for Asynchronous Data Interfaces

This section provides helpful tips and requirements to consider when using asynchronous interfaces in your application.

LabVIEW FPGA automatically inserts synchronization registers into the signal path of any asynchronous CLIP I/O. The default number of synchronization registers inserted into the signal path is one for signals going into the CLIP, and two for signals coming from the CLIP. If the design can safely leave the synchronization registers out of the interface path, you can remove them by completing the following steps.

1. In the LabVIEW project, expand the list of CLIP I/O.
2. Right click the I/O signal and select **Properties**.
3. In the **Properties** window, change the synchronization registers to a non-default value.

Refer to the Passing Data between Component-Level IP and VIs topic in the *LabVIEW Help* for more information about synchronization registers.

Asynchronous data interfaces require additional constraints to compile successfully. Often, you must place a constraint on an interface that needs to perform a clock domain crossing, which causes the compiler to ignore timing on the identified interface path. Refer to the *How to Constrain Timing Failures in ISE* section or the *How to Constrain Timing Failures in Vivado* section of this chapter for instructions about how to apply constraints to interfaces with a destination or origin in LabVIEW FPGA.

Xilinx Integrated Synthesis Environment (ISE) and Xilinx Vivado Design Suite compilers may analyze asynchronous interfaces differently. For example, a design that uses an unconstrained asynchronous interface may compile without errors in ISE, but will fail timing in Vivado due to the increased scrutiny that Vivado places on interfaces that cross clock domains. You must account for cases like these when developing a CLIP intended for use on ISE and Vivado targets. When developing a CLIP for both compilers, explicity define the signal's intended behavior, rather than allowing implicit assumptions to define the behavior.

Best Practices for Designing Constraints

The following best practices can ensure that the constraints are properly and efficiently designed for your application. The recommendations in this section apply to CLIPs intended for use on FPGA targets that use a Vivado compiler. For more information about designing constraints, refer to the following Xilinx user guides.

- *Vivado Design Suite User Guide: Using Constraints*
- *7 Series FPGAs Clocking Resources: User Guide*
- *7 Series FPGAs Select IO Resources: User Guide*

Constraint File Organization

The organization of constraints within the `.xdc` file helps the compiler correctly analyze the constraints in the design. The compiler interprets constraints in a sequential manner, from top to bottom, until all constraints have been analyzed. NI recommends the following constraint organization structure:

```
## Timing Assertions Section
         # Primary clocks
         # Virtual clocks
         # Generated clocks
         # Clock Groups
         # Input and output delay constraints
## Timing Exceptions Section
         # False Paths
         # Max Delay / Min Delay
         # Multicycle Paths
         # Case Analysis
         # Disable Timing
```

Keep physical constraints in a separate `.xdc` file, such as the `.xdc` file that defines the I/O assignments of the GPIO lines.

Documenting Constraints

Include comments in your constraints file that clearly indicate the signal being acted upon, the purpose of the constraint, and why the signal is being constrained. The following code provides examples of comments.

```
##Primary Clocks
# 50 MHz DataClk - the DUT generates a free running and static 50MHz clock
to which the data is synchronous
create_clock -name DataClk -period 20 [get_ports aUserGpio[38]]
```

Clocks

Use `create_clock` to create all primary and virtual clocks in your CLIP design. For example, you may have a device under test (DUT) that provides a data clock that is synchronous to a data source. The constraint file defines that signal as a clock to the compiler. Failure to constrain clocks in your design could lead to issues in which your compilation passes timing, but works inconsistently during functional testing.

You do not need to create constraints for clocks generated in the LabVIEW FPGA design, such as a base clock source from the LabVIEW FPGA project. These constraints are automatically generated by your LabVIEW FPGA project.

Resets

Asynchronous resets that originate inside your CLIP should also be constrained to remove false timing violations that may occur. Identify all reset signals in your CLIP design and be sure to create constraints as required.

Note If you assign a top-level reset signal in LabVIEW FPGA's CLIP Wizard, you should not create a constraint for this signal in the `.xdc` file.

Max Delay and False Path

The .xdc file uses two important constraints called `set_max_delay` and `set_false_path`. You must understand the implications and the correct application of `set_max_delay` and `set_false_path` to successfully compile your design, and to analyze time-critical paths in your design. If you do not include constraints for these paths, the compile tools incorrectly analyze these paths' timing. For more information, refer to the *Vivado Design Suite User Guide: Using Constraints*.

NI recommends only using `set_max_delay` constraints rather than `set_false_path` constraints. Both commands can be used to create a successfully constrained project, but `set_max_delay` provides more timing control over path delay than `set_false_path`.

For example, a path delay of 20 ns would not be reported to the designer if a `set_false_path` command were used, yet the compiler could route your signal with any delay it needed based on the constraint. However, using `set_max_delay` at 10 ns would safely constrain the maximum path delay in the fabric to 10 ns. You can relax the maximum delay path value to find the optimal value for your design.

Any time you use `set_false_path`, you must understand where the constraint begins and ends. Using both `-to` and `-from` creates the most control over the path to be ignored. Be careful when using just one constraint or the other, as timing critical paths other than the one you intend to constrain in the design may be ignored.

Clock Groups

When setting two clock domains asynchronous of each other in your constraints file, you can use the `set_clock_groups` Tcl command, which ignores any timing analysis between two clock domains. Using this command is acceptable when you have a clock domain crossing between two clocks that exist entirely within the CLIP, and are not related to one another.

LabVIEW FPGA base clocks should not be set asynchronous from CLIP clocks when CLIP clocks are brought into LabVIEW FPGA and used on the block diagram. Doing so ignores important timing relationships in LabVIEW FPGA. If you set a base clock asynchronous from a CLIP clock, ensure that the CLIP clock is not used in LabVIEW FPGA.

The best practice for constraining a clock domain crossing is to mark any clock domain crossings (CDCs) and their related D flop in the design, and use the `set_max_delay` or `set_false_path` command for all registers or FIFOs crossing into a new clock domain. The following example constraints demonstrate this function.

```
#all D inputs of cdc synchronization registers are ignored
set_false_path -to [get_pins -hier *aurora_64b66b_cdc_to*/D]
#all D inputs of cdc synchronization registers have maximum delay of 10 ns
set_max_delay 10 -to [get_pins -hier *aurora_64b66b_cdc_to*/D]
```

Creating .xdc Constraints

Several different methods are available to create the `.xdc` constraint path to pins or cells using Vivado constraint syntax. Consider the following tips while building the path to your pin, port, or cell in your design. For more information about creating `.xdc` constraints, refer to the *Xilinx Vivado Design Suite User Guide: Using Constraints* (UG903).

- When you first begin creating your `.xdc` constraints, you can use the Tcl Console to validate the syntax of the `.xdc` commands before saving them in the `.xdc` files. You can do this by creating a project in Vivado and opening the elaborated or synthesized design, which then allows you to copy and paste constraints in the Tcl Console. If you enter the constraint and it does not return an error, then it is validated. Refer to the *Xilinx Vivado Design Suite User Guide: Using Constraints* (UG903) for more information.
- NI recommends avoiding using `-hierarchical` in your constraints. Instead, explicitly use the `/` character at all levels of hierarchy to avoid potential problems with finding the path in memory. Vivado synthesis can sometimes flatten hierarchy levels, which removes

some of the hierarchy path from memory. Creating a constraint that uses the `-hierarchical` option with partial paths, such as using wildcards across multiple levels of hierarchy, may return a warning that the object was not found. The following example is adapted from the *Xilinx Vivado Design Suite User Guide: Using Constraints* (UG903):

RTL Design Example: register in overall design

```
inst_A/inst_B/control_reg
```

If the hierarchy is flattened, the following constraints will be found in the resulting flattened design. However, if the hierarchy is not flattened, they will not be found. Note that a wildcard is used to traverse multiple levels in both instances. Because a wildcard is used to represent multiple levels of hierarchy with the hierarchical command, Vivado cannot find the path in the design.

```
% get_cells –hierarchical *inst_B/control_reg (path not
found/constraint not applied in design, missing inst_A level)
% get_cells inst_A*control_reg (path not found/constraint not applied
in design, missing inst_b level)
```

Explicitly writing all or some part of every level in the hierarchy ensures that constraints are properly found in both flattened and unflattened hierarchies, leading up to the object you wish to constrain. The following is a correction to the above constraints to be found in any synthesis/implementation of control_reg. Wildcards are still used within a level of hierarchy, but do not traverse any hierarchy level:

```
% get_cells inst_A/inst_B/*_reg (successfully finds all pins/cells
with *_reg at the control_reg level)
% get_cells inst_*/inst_B/control_reg (successfully finds all
pins/cells with inst_* at the top inst_A level)
```

Refer to the following Xilinx Answer Record for more information: *AR# 62136 Vivado Constraints - Understanding how hierarchy separator "/" works with wildcard * in XDC and UCF.*

- NI provides a `%ClipInstancePath%` token that appends the overall VHDL hierarchy used by the LabVIEW FPGA compiler to your existing external top-level VHDL path. For instance, if you have a constraint for your CLIP that works outside of LabVIEW FPGA, when you bring that constraint into LabVIEW FPGA, you must append the token to the beginning of the object path so that the compiler can still find it in memory, as shown in the following example:

```
% get_cells inst_A/inst_B/control_reg(Vivado constraint)
% get_cells %ClipInstancePath%/inst_A/inst_B/control_reg(LV FPGA
constraint that adds additional hierarchy levels)
```

Design Analysis and Closure Techniques

Timing violations and design issues are common when working in Vivado. For more information on using various Vivado tools for analysis and timing closure, refer to the *Vivado Design Suite User Guide: Design Analysis and Closure Techniques*, available at `xilinx.com`.

Common Issues and Troubleshooting

When creating constraints, you may find that you have some issues migrating constraints from the Vivado standalone environment into your LabVIEW FPGA project. Below are common issues that arise when migrating a Vivado design to LabVIEW, and troubleshooting techniques to correct them.

Port vs Pin

When generating constraints for your adapter module, you may migrate some existing `.xdc` constraints from a sample project into your LabVIEW FPGA project. In the LabVIEW FPGA project, your top-level VHDL file (the CLIP) is a few levels below the adapter module's top-level VHDL file within your LabVIEW FPGA project. It is common to see the following path appended to your top-level VHDL file by using the `%ClipInstancePath%` token (where `XXXX` is replaced by the model number of your adapter module):

```
DeviceWindow/theCLIPs/AdapterModuleCLIP
```

This scenario creates an issue when referring to a port vs a pin in your top-level VHDL file. Since the port must be on the top level to be referenced, you will see warnings that the port does not exist when you bring your design into the LabVIEW FPGA project. To correct this, you must replace `get_ports` with `get_pins` in your .xdc constraints file, along with adding the `%ClipInstancePath%` token to the path.

In the following example, the first constraint may work standalone in Vivado; the second constraint is updated for migrating to the LabVIEW FPGA project.

- `create_clock -name DataClk -period 6.400 [get_ports DataClk]`
- `create_clock -name DataClk -period 6.400 [get_pins %ClipInstancePath%/AuroraBlock.DataClkIBuf/O]`

Troubleshooting Xilinx Log Files

When troubleshooting Xilinx log files (`lvXilinxLog.txt`), the following tips can help you efficiently resolve problems in your design compilation. The `lvXilinxLog.txt` log file is located at `C:\NIFPGA\compilation\`*`CompilationProjectName`*. In addition to the `lvXilinxLog.txt` is an `output_files.zip` file that contains useful compiler logs.

- Resolve critical warnings in your log file. Search for "critical warnings" to locate all references inside the log file.
- Analyze user constraints after synthesis in the compilation flow. The *Processing XDC Constraints* section of the `lvXilinxlog.txt file` identifies issues in the constraints. You must resolve any warnings or critical warnings directly related to user-defined constraints.
  - Ignore warnings referencing constraints that you did not directly create. The `.xdc` file you created is appended to a larger `.xdc` file, which contains many automatically generated constraints for LabVIEW FPGA. These constraints do not have a negative impact on your design.

  - If a user-defined constraint shows up as a warning or a critical warning, then the constraint was not applied to the compiled bitstream. Any user-defined constraints not indicated in this section of the log file were successfully applied.
- Most constraint warnings provide helpful information about the source of the issue. Read this information for an explanation of why the warnings appeared.
- LabVIEW 2016 and later include an `unapplied_constraints.xdc` file in the `output_files.zip` file. The `output_files.zip` file also contains a `PumaK7Top.twr` that contains a timing report of the analyzed signals, which can be useful when investigating a timing failure.

Syntax Issues

Mistakes in the naming of a path can generate the following warning:

```
"WARNING: [Vivado 12-584] No ports matched"
"WARNING: [Vivado 12-508] No pins matched"
"WARNING: [Vivado 12-180] No cells matched"
"WARNING: [Vivado 12-627] No clocks matched"
```

To avoid these warnings, ensure the path you are searching for exists in your design.

The following is a list of common syntax issues that you may encounter in your constraints design:

- Missing brackets { } or [ ]
- The port or line number in a constraint is incorrect
- Misspelled port or line names
- Missing underscores

Debugging With the Timing Violation Analysis Window

As you compile in LabVIEW FPGA, you may encounter timing violations in the **Compilation Status** window. Refer to the Timing Violation Analysis Window topic of the *LabVIEW FPGA Help* for more information about the features of the window.

In order to effectively debug these violations, it helps to understand the two main types of errors and their common reasons for failure. The first failure is an unrealistic timing requirement in your design, as shown in the example below.

```
Path 5: Requirement 0.01ns missed by 3.95ns
    PXIe-7975R IO Socket v1
    PXIe-7975R IO Socket v1
```

Such timing violations may occur when clock domain crossings are unconstrained. The phase shift drift simulated by the compiler leads to the small interval required.

If the above error message appears in the **Timing Violation Analysis** window, you must resolve any unrealistic timing requirements first in your design. Other timing violations may be

occurring only because the compiler is trying to meet the unrealistic timing requirement. The compiler prioritizes the tight paths over other important timing critical paths, and having these requirements in your design may cause several paths to fail to meet timing simultaneously.

To resolve these tight timing violations, first ensure that the clock domain crossings are crossed properly. Next, check for errors in the path. You can see the path by clicking on the items under the violation in the **Timing Violation Analysis** window, as shown in Figure 13-8. The most common way to correct these issues is to call a false path constraint so that the compiler no longer analyzes timing for the given path. For more information about handling timing failures by ignoring the path for analysis, refer to the *How to Constrain Timing Failures in ISE* section or the *How to Constrain Timing Failures in Vivado* section of this chapter.

The following figure shows other common paths that may appear in the **Timing Violation Analysis** window:

Figure 13-8. Timing Violation Analysis Window

⊟ Path 2 : Requirement 4.00ns missed by 0.47ns	4.00	0.77	3.23	35
Non-diagram component	0.26	0.26	0.00	
Non-diagram component	0.55	0.05	0.50	35
Non-diagram component	0.96	0.14	0.82	13
Non-diagram component	0.12	0.12	0.00	1
Non-diagram component	0.88	0.12	0.76	3
Non-diagram component	0.79	0.04	0.75	2
Non-diagram component	0.45	0.04	0.40	8
Non-diagram component	0.00	0.00	0.00	1
⊟ Path 3 : Requirement 0.20ns missed by 2.17ns	3.64	0.30	3.34	1
PXIe-6592R IO Socket v1	0.26	0.26	0.00	
Non-diagram component	3.38	0.04	3.34	1
PXIe-6592R IO Socket v1	0.00	0.00	0.00	1

Requirements that look like a period interval to a real clock in your design typically do not need correction when a Non-diagram component is the only item in the path. This is usually remedied once the unrealistic timing requirements are corrected. When there are paths in your LabVIEW FPGA diagram or your CLIP, and the requirement is reasonable, you may have to use pipelining to ensure your logic can be performed within one clock cycle of the main clock driving the logic. For more information about pipelining, refer to the *Optimizing FPGA VIs Using Pipelining* topic of the *LabVIEW FPGA Module Help*.

How to Constrain Timing Failures in ISE

In order for LabVIEW to resolve timing failures on a path between a CLIP and the LabVIEW FPGA Module block diagram, you must specify a timing constraint for that path. Refer to the following KnowledgeBase article for more information: *How Do I Instruct LabVIEW to Ignore Timing Failures between Component-Level IP (CLIP) and the LabVIEW FPGA Module Block Diagram?*

How to Constrain Timing Failures in Vivado

In order for LabVIEW to ignore timing failures on a path between a CLIP and the LabVIEW FPGA Module block diagram, you must specify a timing constraint for that path.

Complete the following steps to create and configure a constraints file.

1. If your CLIP component does not currently include an `.xdc` constraints file in the LabVIEW project, create a blank text file with the `.xdc` extension.
2. Open the file for editing.
3. Locate the failing compilation in the **Compilation Status** window and click the **Investigate Timing Violation…** button to see a description of the failing path.
4. Note the internal names of the source and destination registers, located at the bottom of the **Timing Violation Analysis** window. You need one of these pieces of information in order to write the constraint.
5. The constraint being created ignores timing on the failing path. The syntax may vary, depending on the path's direction.
   - If the path is an input path from CLIP to the LabVIEW FPGA Module block diagram refer to the *Digital Input Case* section.
   - If the path is an output path from the LabVIEW FPGA Module block diagram to CLIP, refer to the *Digital Output Case* section.
6. After editing the `.xdc` file, save it and close it.
7. Edit the CLIP declaration and add the `.xdc` file to it if you created the file specifically for fixing these timing failures.
8. Rebuild your LabVIEW project. The timing failure on the path in this case should be handled by the new constraint file.

Digital Input Case

In the Xilinx Vivado digital input case, the constraint ignores timing on all the failing paths that have a certain destination, and is retrieved from the **Timing Violation Analysis** window. Refer to the following example syntax:

```
set_false_path -to [get_cells *<internal name of the destination register in the failing path>]
```

In the above constraint, `<internal name of the destination register in the failing path>` refers to the internal name of the destination register. The internal name can be found at the bottom of the **Timing Violation Analysis** window. For example, the internal name may look like the following:

```
/Component_dash_Level_IP_GPIO_In_0_din/cFirstRegister_ms_reg[0]
```

Add the above constraint to your `.xdc` file and populate the necessary fields with relevant information.

The following code is an example of a complete constraint, based on the **Timing Violation Analysis** window.

```
set_false_path -to [get_cells
%ClipInstancePath%cFirstRegister_ms_reg[0]]
```

Note FlexRIO CLIPs are not instantiated as part of the top-level design. NI recommends using `%ClipInstancePath%` in your constraints information to ensure that the Xilinx compiler can locate the signal name. If you do not direct the compiler to the signal using `%ClipInstancePath%`, the compiler may return an error.

Digital Output Case

In the Xilinx Vivado digital output case, the constraint ignores timing on all the failing paths that have a certain source, and is retrieved from the **Timing Violation Analysis** window. Refer to the following example syntax:

```
set_false_path -from [get_cells *<internal name of the source register
in the failing path>]
```

In the above constraint, `<internal name of the source register in the failingpath>` refers to the internal name of the source register. The internal name can be found at the bottom of the **Timing Violation Analysis** window. For example, the internal name may look like the following:

```
/Component_dash_Level_IP_GPIO_Out_0_dout/DoRegister.
SyncRegisterVector[0].SyncRegisterRising.cSyncRegister
```

Add the above constraint to your .xdc file and populate the necessary fields with relevant information.

Refer to the following example of a complete constraint, based on the **Timing Violation Analysis** window shown in the figure above:

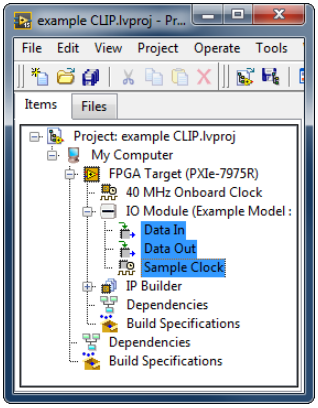
```
set_false_path -from [get_cells
%ClipInstancePath%DoRegister.SyncRegisterVector[0].
SyncRegisterRising.cSyncRegister]
```

Note FlexRIO CLIPs are not instantiated as part of the top-level design. NI recommends using `%ClipInstancePath%` in your constraints information to ensure that the Xilinx compiler can locate the signal name. If you do not direct the compiler to the signal using `%ClipInstancePath%`, the compiler may return an error.

Updating the LabVIEW Project to Reflect Changes in the CLIP XML

The CLIP XML file defines the signals from the top-level design file that are imported into LabVIEW FPGA.The following figure shows the CLIP I/O exposed by the `ExampleIOModuleCLIPK7.xml` used by the MDK example CLIP for NI PXIe-797*x*R targets.

Figure 13-9. Example CLIP I/O



When the XML file is first loaded into a project, a snapshot of the CLIP I/O is cached in the `.lvproj` file. This provides certain performance advantages; however, it can lead to issues during development if the XML file is frequently updated, since those updates on disk will not be propagated to the LabVIEW project.

For example, the adapter module CLIP in Figure 13-9 contains CLIP input with an interface name of **Data In**. If the interface name of that signal is updated to **Data In U32** in the `ClipAdder.xml` file, those changes are not automatically reflected in the LabVIEW project by saving the XML file, nor by opening and closing the LabVIEW Project.

The safest way to propagate updated XML changes to the LabVIEW project is to unload the CLIP from the project, save the project, then load the CLIP back into the project. Performing these steps removes the cached I/O from the `.lvproj` file and guarantees that the LabVIEW project reflects the most recent contents of the XML file. To unload a CLIP, open the **IO Module Properties** dialog page and uncheck the **Enable IO Module** box, then click **OK**. You must update the LabVIEW project for every target in the project that has a CLIP with an XML file that has changed.

You do not need to unload and reload a CLIP when changing `.vhd` files or any other design files. Only changes to the XML require this step.

A

Signal Suggestions

NI 795xR, NI 796xR, NI-793*x*R, and NI 797xR Pinout Locations

Table A-1 describes the Xilinx FPGA pin location for each FPGA signal on the card edge interface.

The tables listed in this chapter are also available as a searchable Excel document in Knowledge Base article *6NND5NOA* and on the FlexRIO Community page.

Table A-1. NI 795xR, NI 796xR, NI-793*x*R, and NI 797xR Pinout Locations

Sxxx	795xR/ 796xR Name	793x/797x Name	795x FPGA Pin	796x FPGA Pin	797x/793x Pin
S1	GPIO_48_n	GPIO_57	F20	A33	J11
S2	GPIO_48	GPIO_57_n	G20	B32	J12
S3	GPIO_47_n	GPIO_56	E20	C33	K14
S4	GPIO_47	GPIO_56_n	E21	B33	J14
S5	GPIO_46_n	GPIO_55	E23	D32	L11
S6	GPIO_46	GPIO_55_n	E22	C32	K11
S7	GPIO_45_n	GPIO_54	F23	D34	K13
S8	GPIO_45	GPIO_54_n	F22	C34	J13
S9	GPIO_44_n	GPIO_53	G22	H32	C12
S10	GPIO_44	GPIO_53_n	G21	G32	B12
S11	GPIO_43_n	GPIO_52	H22	E34	A11
S12	GPIO_43	GPIO_52_n	H21	F33	A12
S13	GPIO_42_n	GPIO_51	H19	E33	J16
S14	GPIO_42	GPIO_51_n	J19	E32	H16
S15	GPIO_41_n	GPIO_50	J23	F34	G13
S16	GPIO_41	GPIO_50_n	H23	G33	F13

Table A-1. NI 795xR, NI 796xR, NI-793*x*R, and NI 797xR Pinout Locations (Continued)

Sxxx	795xR/ 796xR Name	793x/797x Name	795x FPGA Pin	796x FPGA Pin	797x/793x Pin
S17	GPIO_40_n	GPIO_49	J20	H33	E14
S18	GPIO_40	GPIO_49_n	J21	J32	E15
S19	GPIO_39_n	GPIO_48	K20	J34	B13
S20	GPIO_39	GPIO_48_n	K21	H34	A13
S21	GPIO_38_n	GPIO_47	K22	K34	H21
S22	GPIO_38	GPIO_47_n	K23	L34	H22
S23	GPIO_37_n	GPIO_46	L19	K32	J17
S24	GPIO_37	GPIO_46_n	L20	K33	H17
S25	GPIO_36_n	GPIO_45	M21	M32	C17
S26	GPIO_36	GPIO_45_n	M22	L33	B17
S27	GPIO_35_n	GPIO_44	N19	N34	D16
S28	GPIO_35	GPIO_44_n	P19	P34	C16
S29	GPIO_34_n	GPIO_43	M19	N32	B18
S30	GPIO_34	GPIO_43_n	M20	P32	A18
S31	GPIO_33_n	GPIO_42	N23	R34	A16
S32	GPIO_33	GPIO_42_n	P23	T33	A17
S33	GPIO_32_n	GPIO_41	H26	AA33	C19
S34	GPIO_32	GPIO_41_n	G26	Y33	B19
S35	GPIO_31_n	GPIO_40	G25	Y34	G17
S36	GPIO_31	GPIO_40_n	G24	AA34	F17
S37	GPIO_30_n	GPIO_39	J26	W32	G18
S38	GPIO_30	GPIO_39_n	J25	Y32	F18
S39	GCLK_LVDS	GPIO_38	F14	H17	D17
S40	GCLK_LVDS_n	GPIO_38_n	E13	H18	D18
S41	GPIO_29_n	GPIO_37	J24	AD34	D21

Table A-1. NI 795xR, NI 796xR, NI-793*x*R, and NI 797xR Pinout Locations (Continued)

Sxxx	795xR/ 796xR Name	793x/797x Name	795x FPGA Pin	796x FPGA Pin	797x/793x Pin
S42	GPIO_29	GPIO_37_n	H24	AC34	C21
S43	GPIO_28_n	GPIO_36	L25	AB32	A20
S44	GPIO_28	GPIO_36_n	L24	AC32	A21
S45	GPIO_27_n	GPIO_35	K26	AB33	C20
S46	GPIO_27	GPIO_35_n	K25	AC33	B20
S47	GPIO_26_n	GPIO_34	M26	AE33	F21
S48	GPIO_26	GPIO_34_n	M25	AF33	E21
S49	GPIO_25_n	GPIO_33	N24	AE34	B22
S50	GPIO_25	GPIO_33_n	M24	AF34	A22
S51	GPIO_24_n	GPIO_32	N26	AJ34	D22
S52	GPIO_24	GPIO_32_n	P26	AH34	C22
S53	GPIO_23_n	GPIO_31	P24	AE32	E19
S54	GPIO_23	GPIO_31_n	P25	AD32	D19
S55	GPIO_22_n	GPIO_30	T25	AK33	H20
S56	GPIO_22	GPIO_30_n	T24	AK34	G20
S57	GPIO_21_n	GPIO_29	U26	AH32	J19
S58	GPIO_21	GPIO_29_n	V26	AG32	H19
S59	GPIO_20_n	GPIO_28	U25	AK32	K19
S60	GPIO_20	GPIO_28_n	U24	AJ32	K20
S61	GPIO_19_n	GPIO_27	W26	AL33	L17
S62	GPIO_19	GPIO_27_n	W25	AL34	L18
S63	GPIO_18_n	GPIO_26	Y26	AM32	F20
S64	GPIO_18	GPIO_26_n	Y25	AM33	E20
S65	GPIO_17_n	GPIO_25	AA25	AN33	G22
S66	GPIO_17	GPIO_25_n	AB25	AN34	F22

Table A-1. NI 795xR, NI 796xR, NI-793*x*R, and NI 797xR Pinout Locations (Continued)

Sxxx	795xR/ 796xR Name	793x/797x Name	795x FPGA Pin	796x FPGA Pin	797x/793x Pin
S67	GPIO_16_n	GPIO_24	AB26	AP32	K18
S68	GPIO_16	GPIO_24_n	AC26	AN32	J18
S69	RSVD_A1	TDC_Assert_Clk	N/A	N/A	N/A
S70	RSVD_A2	TDC_Assert_ Clk_n	N/A	N/A	N/A
S71	Veeprom	Veeprom	N/A	N/A	N/A
S72	$V_{cco}B$	V_{cco}	N/A	N/A	N/A
S73	TB_Power_Good	TB_Power_Good	N/A	N/A	N/A
S74	SDA	SDA	N/A	N/A	N/A
S75	GPIO_65_n	GPIO_67	B16	J29	H11
S76	GPIO_65	GPIO_67_n	B15	H29	H12
S77	GPIO_64_n	GPIO_66	C16	E31	H15
S78	GPIO_64	GPIO_66_n	D16	F31	G15
S79	GPIO_63_n	GPIO_65	C17	K29	D11
S80	GPIO_63	GPIO_65_n	D18	L29	C11
S81	GPIO_62_n	GPIO_64	A17	G31	B14
S82	GPIO_62	GPIO_64_n	B17	H30	A15
S83	GPIO_61_n	GPIO_63	A19	J31	F11
S84	GPIO_61	GPIO_63_n	A18	J30	E11
S85	GPIO_60_n	GPIO_62	C18	M30	C15
S86	GPIO_60	GPIO_62_n	B19	L30	B15
S87	GPIO_59_n	GPIO_61	B20	P29	D14
S88	GPIO_59	GPIO_61_n	A20	N29	C14
S89	GPIO_58_n	GPIO_60	D19	L31	D12
S90	GPIO_58	GPIO_60_n	C19	K31	D13

Table A-1. NI 795xR, NI 796xR, NI-793*x*R, and NI 797xR Pinout Locations (Continued)

Sxxx	795xR/ 796xR Name	793x/797x Name	795x FPGA Pin	796x FPGA Pin	797x/793x Pin
S91	GPIO_57_n	GPIO_59	D20	P30	F15
S92	GPIO_57	GPIO_59_n	D21	P31	E16
S93	GPIO_56_n	GPIO_58	B21	N30	F12
S94	GPIO_56	GPIO_58_n	C21	M31	E13
S95	GPIO_55_n	GPIO_23	A22	R31	H24
S96	GPIO_55	GPIO_23_n	B22	T31	H25
S97	GPIO_54_n	GPIO_22	A24	T30	E23
S98	GPIO_54	GPIO_22_n	A23	U30	D23
S99	GPIO_53_n	GPIO_21	C23	T29	G23
S100	GPIO_53	GPIO_21_n	B24	T28	G24
S101	GPIO_52_n	GPIO_20	C24	U28	F25
S102	GPIO_52	GPIO_20_n	D24	U27	E25
S103	GPIO_51_n	GPIO_19	A25	R27	E24
S104	GPIO_51	GPIO_19_n	B25	R26	D24
S105	GPIO_50_n	GPIO_18	C26	T26	B23
S106	GPIO_50	GPIO_18_n	B26	U26	A23
S107	GPIO_49_n	GPIO_17	D25	T25	H26
S108	GPIO_49	GPIO_17_n	D26	U25	H27
S109	GPIO_15_n	GPIO_16	T22	V24	C24
S110	GPIO_15	GPIO_16_n	T23	W24	B24
S111	GPIO_14_n	GPIO_15	R20	W26	F26
S112	GPIO_14	GPIO_15_n	R21	Y26	E26
S113	GCLK_SE	GPIO_14	AD18	AG21	C25
S114	GND	GPIO_14_n	N/A	N/A	B25
S115	GPIO_13_n	GPIO_13	T19	W25	B27

Table A-1. NI 795xR, NI 796xR, NI-793*x*R, and NI 797xR Pinout Locations (Continued)

Sxxx	795xR/ 796xR Name	793x/797x Name	795x FPGA Pin	796x FPGA Pin	797x/793x Pin
S116	GPIO_13	GPIO_13_n	T20	V25	A27
S117	GPIO_12_n	GPIO_12	U21	W27	A25
S118	GPIO_12	GPIO_12_n	U22	Y27	A26
S119	GPIO_11_n	GPIO_11	W23	W30	B28
S120	GPIO_11	GPIO_11_n	W24	V30	A28
S121	GPIO_10_n	GPIO_10	V23	V27	D26
S122	GPIO_10	GPIO_10_n	V24	V28	C26
S123	GPIO_9_n	GPIO_9	AA24	Y31	C29
S124	GPIO_9	GPIO_9_n	AA23	W31	B29
S125	GPIO_8_n	GPIO_8	Y22	V29	G27
S126	GPIO_8	GPIO_8_n	Y23	W29	F27
S127	GPIO_7_n	GPIO_7	AC22	Y29	D29
S128	GPIO_7	GPIO_7_n	AC23	Y28	C30
S129	GPIO_6_n	GPIO_6	AC24	AA31	E28
S130	GPIO_6	GPIO_6_n	AB24	AB31	D28
S131	GPIO_5_n	GPIO_5	AA22	AC30	B30
S132	GPIO_5	GPIO_5_n	AB22	AB30	A30
S133	GPIO_4_n	GPIO_4	AB21	AA30	E29
S134	GPIO_4	GPIO_4_n	AC21	AA29	E30
S135	GPIO_3_n	GPIO_3	W20	AC29	G28
S136	GPIO_3	GPIO_3_n	W21	AD30	F28
S137	GPIO_2_n	GPIO_2	W19	AD29	D27
S138	GPIO_2	GPIO_2_n	V19	AE29	C27
S139	GPIO_1_n	GPIO_1	Y20	AK31	G29
S140	GPIO_1	GPIO_1_n	Y21	AJ31	F30

Table A-1. NI 795xR, NI 796xR, NI-793*x*R, and NI 797xR Pinout Locations (Continued)

Sxxx	795xR/ 796xR Name	793x/797x Name	795x FPGA Pin	796x FPGA Pin	797x/793x Pin
S141	GPIO_0_n	GPIO_0	AC19	AF30	H30
S142	GPIO_0	GPIO_0_n	AD19	AF29	G30
S143	IOModSyncClk*	IOModSyncClk	N/A	N/A	N/A
S144	IOModSyncClk_n†	IOModSyncClk_n	N/A	N/A	N/A
S145	RSVD	RSVD	N/A	N/A	N/A
S146	V_{cco}A	V_{cco}	N/A	N/A	N/A
S147	TB_Present_n	TB_Present_n	N/A	N/A	N/A
S148	SCL	SCL	N/A	N/A	N/A

* RSVD_B1 on NI 795xR devices

† RSVD_B2 on NI 795xR devices

NI 795xR, NI 796xR, NI-793xR, and NI 797xR Pinout Capabilities

Table A-2 describes the functionality of each pin to aid in the design of your adapter module gold finger and schematic.

Table A-2. NI 795xR, NI 796xR, NI-793xR, and NI 797xR Pinout Capabilities

S*xxx*	V5 Clock Capable?	K7 Clock Capable?	V5 GPIO?	K7 GPIO?	V5 Bank	K7 Bank
S1	No	No	TRUE	TRUE	Bank 11	Bank 18
S2	No	No	TRUE	TRUE	Bank 11	Bank 18
S3	No	No	TRUE	TRUE	Bank 11	Bank 18
S4	No	No	TRUE	TRUE	Bank 11	Bank 18
S5	No	No	TRUE	TRUE	Bank 11	Bank 18
S6	No	No	TRUE	TRUE	Bank 11	Bank 18
S7	No	No	TRUE	TRUE	Bank 11	Bank 18
S8	No	No	TRUE	TRUE	Bank 11	Bank 18
S9	No	No	TRUE	TRUE	Bank 11	Bank 18

Table A-2. NI 795xR, NI 796xR, NI-793xR, and NI 797xR Pinout Capabilities (Continued)

S*xxx*	V5 Clock Capable?	K7 Clock Capable?	V5 GPIO?	K7 GPIO?	V5 Bank	K7 Bank
S10	No	No	TRUE	TRUE	Bank 11	Bank 18
S11	No	No	TRUE	TRUE	Bank 11	Bank 18
S12	No	No	TRUE	TRUE	Bank 11	Bank 18
S13	No	No	TRUE	TRUE	Bank 11	Bank 18
S14	No	No	TRUE	TRUE	Bank 11	Bank 18
S15	No	MRCC	TRUE	TRUE	Bank 11	Bank 18
S16	No	MRCC	TRUE	TRUE	Bank 11	Bank 18
S17	Regional	No	TRUE	TRUE	Bank 11	Bank 18
S18	Regional	No	TRUE	TRUE	Bank 11	Bank 18
S19	Regional	No	TRUE	TRUE	Bank 11	Bank 18
S20	Regional	No	TRUE	TRUE	Bank 11	Bank 18
S21	Regional	No	TRUE	TRUE	Bank 11	Bank 17
S22	Regional	No	TRUE	TRUE	Bank 11	Bank 17
S23	Regional	No	TRUE	TRUE	Bank 11	Bank 17
S24	Regional	No	TRUE	TRUE	Bank 11	Bank 17
S25	No	No	TRUE	TRUE	Bank 11	Bank 17
S26	No	No	TRUE	TRUE	Bank 11	Bank 17
S27	No	No	TRUE	TRUE	Bank 11	Bank 17
S28	No	No	TRUE	TRUE	Bank 11	Bank 17
S29	No	No	TRUE	TRUE	Bank 11	Bank 17
S30	No	No	TRUE	TRUE	Bank 11	Bank 17
S31	No	No	TRUE	TRUE	Bank 11	Bank 17
S32	No	No	TRUE	TRUE	Bank 11	Bank 17
S33	No	No	TRUE	TRUE	Bank 13	Bank 17
S34	No	No	TRUE	TRUE	Bank 13	Bank 17

Table A-2. NI 795xR, NI 796xR, NI-793xR, and NI 797xR Pinout Capabilities (Continued)

S*xxx*	V5 Clock Capable?	K7 Clock Capable?	V5 GPIO?	K7 GPIO?	V5 Bank	K7 Bank
S35	No	No	TRUE	TRUE	Bank 13	Bank 17
S36	No	No	TRUE	TRUE	Bank 13	Bank 17
S37	No	No	TRUE	TRUE	Bank 13	Bank 17
S38	No	No	TRUE	TRUE	Bank 13	Bank 17
S39	Global	MRCC	FALSE	TRUE	Bank 3	Bank 17
S40	Global	MRCC	FALSE	TRUE	Bank 3	Bank 17
S41	No	No	TRUE	TRUE	Bank 13	Bank 17
S42	No	No	TRUE	TRUE	Bank 13	Bank 17
S43	No	No	TRUE	TRUE	Bank 13	Bank 17
S44	No	No	TRUE	TRUE	Bank 13	Bank 17
S45	No	No	TRUE	TRUE	Bank 13	Bank 17
S46	No	No	TRUE	TRUE	Bank 13	Bank 17
S47	Regional	SRCC	TRUE	TRUE	Bank 13	Bank 17
S48	Regional	SRCC	TRUE	TRUE	Bank 13	Bank 17
S49	Regional	No	TRUE	TRUE	Bank 13	Bank 17
S50	Regional	No	TRUE	TRUE	Bank 13	Bank 17
S51	Regional	No	TRUE	TRUE	Bank 13	Bank 17
S52	Regional	No	TRUE	TRUE	Bank 13	Bank 17
S53	Regional	SRCC	TRUE	TRUE	Bank 13	Bank 17
S54	Regional	SRCC	TRUE	TRUE	Bank 13	Bank 17
S55	No	No	TRUE	TRUE	Bank 13	Bank 17
S56	No	No	TRUE	TRUE	Bank 13	Bank 17
S57	No	No	TRUE	TRUE	Bank 13	Bank 17
S58	No	No	TRUE	TRUE	Bank 13	Bank 17
S59	No	No	TRUE	TRUE	Bank 13	Bank 17

Table A-2. NI 795xR, NI 796xR, NI-793xR, and NI 797xR Pinout Capabilities (Continued)

S*xxx*	V5 Clock Capable?	K7 Clock Capable?	V5 GPIO?	K7 GPIO?	V5 Bank	K7 Bank
S60	No	No	TRUE	TRUE	Bank 13	Bank 17
S61	No	No	TRUE	TRUE	Bank 13	Bank 17
S62	No	No	TRUE	TRUE	Bank 13	Bank 17
S63	No	MRCC	TRUE	TRUE	Bank 13	Bank 17
S64	No	MRCC	TRUE	TRUE	Bank 13	Bank 17
S65	No	No	TRUE	TRUE	Bank 13	Bank 17
S66	No	No	TRUE	TRUE	Bank 13	Bank 17
S67	No	No	TRUE	TRUE	Bank 13	Bank 17
S68	No	No	TRUE	TRUE	Bank 13	Bank 17
S69	No	No	FALSE	FALSE	N/A	N/A
S70	No	No	FALSE	FALSE	N/A	N/A
S71	No	No	FALSE	FALSE	N/A	N/A
S72	No	No	FALSE	FALSE	N/A	N/A
S73	No	No	FALSE	FALSE	N/A	N/A
S74	No	No	FALSE	FALSE	N/A	N/A
S75	No	No	TRUE	TRUE	Bank 15	Bank 18
S76	No	No	TRUE	TRUE	Bank 15	Bank 18
S77	No	No	TRUE	TRUE	Bank 15	Bank 18
S78	No	No	TRUE	TRUE	Bank 15	Bank 18
S79	No	No	TRUE	TRUE	Bank 15	Bank 18
S80	No	No	TRUE	TRUE	Bank 15	Bank 18
S81	No	No	TRUE	TRUE	Bank 15	Bank 18
S82	No	No	TRUE	TRUE	Bank 15	Bank 18
S83	No	No	TRUE	TRUE	Bank 15	Bank 18
S84	No	No	TRUE	TRUE	Bank 15	Bank 18

Table A-2. NI 795xR, NI 796xR, NI-793xR, and NI 797xR Pinout Capabilities (Continued)

S*xxx*	V5 Clock Capable?	K7 Clock Capable?	V5 GPIO?	K7 GPIO?	V5 Bank	K7 Bank
S85	No	No	TRUE	TRUE	Bank 15	Bank 18
S86	No	No	TRUE	TRUE	Bank 15	Bank 18
S87	Regional	No	TRUE	TRUE	Bank 15	Bank 18
S88	Regional	No	TRUE	TRUE	Bank 15	Bank 18
S89	Regional	MRCC	TRUE	TRUE	Bank 15	Bank 18
S90	Regional	MRCC	TRUE	TRUE	Bank 15	Bank 18
S91	Regional	No	TRUE	TRUE	Bank 15	Bank 18
S92	Regional	No	TRUE	TRUE	Bank 15	Bank 18
S93	Regional	SRCC	TRUE	TRUE	Bank 15	Bank 18
S94	Regional	SRCC	TRUE	TRUE	Bank 15	Bank 18
S95	No	No	TRUE	TRUE	Bank 15	Bank 16
S96	No	No	TRUE	TRUE	Bank 15	Bank 16
S97	No	No	TRUE	TRUE	Bank 15	Bank 16
S98	No	No	TRUE	TRUE	Bank 15	Bank 16
S99	No	No	TRUE	TRUE	Bank 15	Bank 16
S100	No	No	TRUE	TRUE	Bank 15	Bank 16
S101	No	No	TRUE	TRUE	Bank 15	Bank 16
S102	No	No	TRUE	TRUE	Bank 15	Bank 16
S103	No	No	TRUE	TRUE	Bank 15	Bank 16
S104	No	No	TRUE	TRUE	Bank 15	Bank 16
S105	No	No	TRUE	TRUE	Bank 15	Bank 16
S106	No	No	TRUE	TRUE	Bank 15	Bank 16
S107	No	No	TRUE	TRUE	Bank 15	Bank 16
S108	No	No	TRUE	TRUE	Bank 15	Bank 16
S109	No	No	TRUE	TRUE	Bank 17	Bank 16

Table A-2. NI 795xR, NI 796xR, NI-793xR, and NI 797xR Pinout Capabilities (Continued)

S*xxx*	V5 Clock Capable?	K7 Clock Capable?	V5 GPIO?	K7 GPIO?	V5 Bank	K7 Bank
S110	No	No	TRUE	TRUE	Bank 17	Bank 16
S111	No	No	TRUE	TRUE	Bank 17	Bank 16
S112	No	No	TRUE	TRUE	Bank 17	Bank 16
S113	Global	MRCC	FALSE	TRUE	Bank 4	Bank 16
S114	No	MRCC	FALSE	TRUE	Bank 17	Bank 16
S115	No	No	TRUE	TRUE	Bank 17	Bank 16
S116	No	No	TRUE	TRUE	Bank 17	Bank 16
S117	No	No	TRUE	TRUE	Bank 17	Bank 16
S118	No	No	TRUE	TRUE	Bank 17	Bank 16
S119	No	No	TRUE	TRUE	Bank 17	Bank 16
S120	No	No	TRUE	TRUE	Bank 17	Bank 16
S121	No	SRCC	TRUE	TRUE	Bank 17	Bank 16
S122	No	SRCC	TRUE	TRUE	Bank 17	Bank 16
S123	No	No	TRUE	TRUE	Bank 17	Bank 16
S124	No	No	TRUE	TRUE	Bank 17	Bank 16
S125	No	No	TRUE	TRUE	Bank 17	Bank 16
S126	No	No	TRUE	TRUE	Bank 17	Bank 16
S127	Regional	No	TRUE	TRUE	Bank 17	Bank 16
S128	Regional	No	TRUE	TRUE	Bank 17	Bank 16
S129	Regional	SRCC	TRUE	TRUE	Bank 17	Bank 16
S130	Regional	SRCC	TRUE	TRUE	Bank 17	Bank 16
S131	Regional	No	TRUE	TRUE	Bank 17	Bank 16
S132	Regional	No	TRUE	TRUE	Bank 17	Bank 16
S133	Regional	No	TRUE	TRUE	Bank 17	Bank 16
S134	Regional	No	TRUE	TRUE	Bank 17	Bank 16

Table A-2. NI 795xR, NI 796xR, NI-793xR, and NI 797xR Pinout Capabilities (Continued)

S*xxx*	V5 Clock Capable?	K7 Clock Capable?	V5 GPIO?	K7 GPIO?	V5 Bank	K7 Bank
S135	No	No	TRUE	TRUE	Bank 17	Bank 16
S136	No	No	TRUE	TRUE	Bank 17	Bank 16
S137	No	MRCC	TRUE	TRUE	Bank 17	Bank 16
S138	No	MRCC	TRUE	TRUE	Bank 17	Bank 16
S139	No	No	TRUE	TRUE	Bank 17	Bank 16
S140	No	No	TRUE	TRUE	Bank 17	Bank 16
S141	No	No	TRUE	TRUE	Bank 17	Bank 16
S142	No	No	TRUE	TRUE	Bank 17	Bank 16
S143	No	No	FALSE	FALSE	N/A	N/A
S144	No	No	FALSE	FALSE	N/A	N/A
S145	No	No	FALSE	FALSE	N/A	N/A
S146	No	No	FALSE	FALSE	N/A	N/A
S147	No	No	FALSE	FALSE	N/A	N/A
S148	No	No	FALSE	FALSE	N/A	N/A

B

Troubleshooting

LabVIEW FPGA Project Troubleshooting

IO Modules»Properties»General Dialog Box

My adapter module is not displayed in the IO Modules list.

Possible solutions:

- Ensure that the user `IO Modules` directory on disk contains a valid adapter module configuration file (`.fam` or `.tbc`) for your custom adapter module:
  - The user `IO Modules` directory is in the following location:
    - **Windows XP**

      `C:\Documents and Settings\All Users\Shared Documents\National Instruments\FlexRIO\IO Modules`
    - **Windows 7/Vista**

      `C:\Users\Public\Documents\National Instruments\FlexRIO\IO Modules`
    - **Windows 8**

      `C:\Users\Public\Documents\National Instruments\FlexRIO\IO Modules`

    For more information about adapter module configuration file creation, refer to Chapter 9, *Configuring Your Adapter Module for Use with NI 795xR/796xR Modules and LabVIEW FPGA* or Chapter 10, *Configuring Your Adapter Module for Use with NI-793xR/797xR Modules and LabVIEW FPGA*.
- Ensure that the manufacturer and model information in your `.fam` or `.tbc` file is correct. The IO Modules list is automatically populated using these values from all discovered `.fam` and `.tbc` files.

My socketed component-level IP is not listed in the Component Level IP list box when I select my adapter module.

Possible solutions:

- Ensure that your component-level IP is in one of the following two locations:
  - The **FPGA Target Properties Component Level IP** dialog box. To view this dialog box, right-click your FPGA target in the **Project Explorer** window and select **Properties** from the shortcut menu. If your CLIP does not appear in the Component Level IP window, click the **Add** button and navigate to your declaration file location to manually add your CLIP to the project.
  - The user `IO Modules` directory on disk, which is in the following location:
    - **Windows XP**

      `C:\Documents and Settings\All Users\Shared Documents\National Instruments\FlexRIO\IO Modules`
    - **Windows 7/Vista**

      `C:\Users\Public\Documents\National Instruments\FlexRIO\IO Modules`
    - **Windows 8**

      `C:\Users\Public\Documents\National Instruments\FlexRIO\IO Modules`
- Ensure that your component-level IP XML file has a `<CompatibleCLIPSocketList>` tag that lists `FlexRIO-IOModule` or `FlexRIO-K7IOModule` as a compatible socket.

  Examples include the following tags:

  ```
  <CompatibleCLIPSocketList>
  <Socket>FlexRIO-IOModule</Socket>
  </CompatibleCLIPSocketList>
  ```
- Ensure that your component-level IP XML file lists your adapter module in the `<CompatibleIOModuleList>` tag. An adapter module is identified either by EEPROM IO Module ID or by name.

  Examples include the following tags:

  ```
  <CompatibleIOModuleList>
  <IOModule>IOModuleID:0xFFFF0001</IOModule>
  <IOModule>Name:Example Manufacturer::Example Module
  </IOModule>
  </CompatibleIOModuleList>
  ```

IO Modules»Properties»Status Dialog Box

The inserted adapter module name is not displayed correctly.

Possible solutions:

- Ensure that your adapter module has an EEPROM. An adapter module is only identifiable if it contains an EEPROM.
- Ensure that you have a proper RIO resource name configured for your FlexRIO FPGA module.
- Ensure that your adapter module is properly inserted into your FlexRIO FPGA module.
- Ensure that the EEPROM on your adapter module is configured to use the correct I^2C device address. Refer to Chapter 3, *Interfacing Adapter Modules with NI 795xR and NI 796xR Modules*, or Chapter 4, *Interfacing Adapter Modules with NI-793xR and NI 797xR Devices*, for more information about configuring the I^2C device address of your adapter module EEPROM.
- Ensure that the EEPROM on your adapter module is programmed with a proper IO Module ID. Refer to Chapter 3, *Interfacing Adapter Modules with NI 795xR and NI 796xR Modules*, or Chapter 4, *Interfacing Adapter Modules with NI-793xR and NI 797xR Devices*, for more information about programming the IO Module ID on your adapter module.
- Ensure that you have a valid adapter module configuration (`.fam` or `.tbc`) file for your adapter module. Refer to Chapter 9, *Configuring Your Adapter Module for Use with NI 795xR/796xR Modules and LabVIEW FPGA* or Chapter 10, *Configuring Your Adapter Module for Use with NI-793xR/797xR Modules and LabVIEW FPGA*, for more information about creating adapter module configuration files.

The Expected IO Module does not match the Inserted IO Module.

Possible solutions:

- In the **IO Module Properties»General** category, select the adapter module that is currently inserted. Recompile your application and download it to the FPGA. At this point, the FlexRIO FPGA module should be updated to expect the correct adapter module.
- If the inserted adapter module contains an EEPROM, ensure that the `IOModuleID` parameter in the adapter module configuration (`.fam` or `.tbc`) file for the inserted adapter module is correct. This information is used at compilation time to determine which adapter module the FPGA firmware discovers. Refer to Chapter 9, *Configuring Your Adapter Module for Use with NI 795xR/796xR Modules and LabVIEW FPGA* or Chapter 10, *Configuring Your Adapter Module for Use with NI-793xR/797xR Modules and LabVIEW FPGA*, for more information about creating adapter module configuration files.
- If the inserted adapter module does not contain an EEPROM, ensure that the adapter module configuration (`.fam` or `.tbc`) file for the inserted adapter module does not contain a `IOModuleID` parameter. The absence of this parameter is used at compilation time to tell the FPGA firmware that it should expect to discover an adapter module with no EEPROM. Refer to Chapter 9, *Configuring Your Adapter Module for Use with NI 795xR/796xR Modules and LabVIEW FPGA* or Chapter 10, *Configuring Your Adapter Module for Use*

with NI-793xR/797xR Modules and LabVIEW FPGA, for more information about creating adapter module configuration files.

Device Does Not Appear in MAX

Follow the steps for verifying that the device appears in MAX in your device's getting started guide.

1. In the MAX Configuration pane, click **Devices and Interfaces**.
2. Expand the **Chassis** tree to see the list of installed devices, and press <F5> to refresh the list.
3. If the module is still not listed, power off the system, ensure that all hardware is correctly installed, and restart the system.
4. Navigate to the Device Manager.

Table B-1. Device Manager Options

Option	Description
Windows 8	Right-click the Start screen, and select **All apps»Control Panel» Hardware and Sound»Device Manager**.
Windows 7	Select **Start»Control Panel»Device Manager**.
Windows Vista	Select **Start»Control Panel»System and Maintenance»Device Manager**.
Windows XP	Select **Start»Control Panel»System»Hardware»Device Manager**.

5. If you are using a PXI controller, verify that a **National Instruments** entry appears in the system device list.
6. Restart your computer.

If the device still fails to appear in MAX, contact NI technical support or visit `ni.com/support`.

Frequently Asked Questions

Do I have to use an EEPROM on my adapter module?

NI strongly recommends adding an EEPROM to your adapter module for identification purposes. Adding EEPROM provides better electrical protection for both the adapter module and the Controller for FlexRIO/FlexRIO FPGA module. It also improves the software configuration experience within LabVIEW FPGA. Refer to Chapter 3, *Interfacing Adapter Modules with NI 795xR and NI 796xR Modules*, or Chapter 4, *Interfacing Adapter Modules with NI-793xR and NI 797xR Devices*, for more information about adding EEPROM to your adapter module.

However, you can power on an adapter module without an EEPROM and an IO Module ID. Perform the following steps to set up the CLIP to ignore the IO Module ID on the EEPROM.

1. Delete the IOModuleID line from the `.tbc` or `.fam` file, depending on which device you are using.
2. Set the IOModule tag to use the manufacturer name and model name. This links the adapter module CLIP XML file to the `.tbc` or `.fam` file. Use the following syntax:
   ```
   <CompatibleIOModuleList>
   <IOModule>Name:Manufacturer Name::Model Name</IOModule>
   </CompatibleIOModuleList>
   ```
3. Refresh the CLIP from the IO Module Properties window and compile your FPGA VI using the new CLIP XML.
4. Remove the EEPROM.

Can I build a "passive" or "pass through" adapter module to interface to an external hardware device?

NI recommends that you *never* directly expose signals from the front panel connector of the Controller for FlexRIO or FlexRIO FPGA module to the external connectivity of the adapter module. Some form of buffering is required when routing signals from the Controller for FlexRIO or FlexRIO FPGA module to the circuits external to the adapter module. Most of these parts are high speed and offer electrostatic discharge (ESD) protection as well as voltage tolerance protection. It is not necessary to consider this buffer requirement when using the Controller for FlexRIO or FlexRIO FPGA module in applications such as interfacing to an ADC, as the digital signals are not directly exposed. Chapter 3, *Interfacing Adapter Modules with NI 795xR and NI 796xR Modules*, or Chapter 4, *Interfacing Adapter Modules with NI-793xR and NI 797xR Devices*, for more information about electrical considerations and other information about designing your adapter module.

Adding these components to your design prevents damage to the FPGAs.

Caution Using adapter modules which are designed with FPGA pins directly accessible to the front panel (pass through) voids the Controller for FlexRIO and FlexRIO FPGA module warranty. NI is not liable for damage caused from this misuse.

What design templates are available to aid in adapter module design?

Design files and an example IO CLIP are available at `C:\Users\Public\Documents\National Instruments\FlexRIO\Module Development Kit`.

Install FlexRIO Adapter Module Support to view CLIPs as well as example projects containing Host VIs and FPGA code for the adapter modules. You can find the example projects through the NI Example Finder by selecting **Help»Find Examples»Hardware Input and Output» FlexRIO»IO Modules** and selecting your adapter module.

To download FlexRIO Adapter Module Support, visit `ni.com/downloads/drivers` and search for *FlexRIO Adapter Module Support*.

You can find the adapter module CLIPs in the following location:

`C:\Program Files\National Instruments\Shared\FlexRIO\IO Modules`

Can I design a custom module enclosure to provide additional space and/or power to the module?

NI strongly recommends using the NI enclosure for proper heat dissipation, structural strength, and EMI performance. If you design a custom enclosure, refer to the following cautions:

Caution Use a properly sealed interface between the adapter module and the Controller for FlexRIO or FlexRIO FPGA module to achieve regulatory emission compliance. NI's adapter module enclosure is designed to work with the Controller for FlexRIO and FlexRIO FPGA module front panel for a complete emissions solution. This enclosure uses gaskets to form an electrical seal when the enclosure is mated with a Controller for FlexRIO or FlexRIO FPGA module. Custom adapter module enclosures should use a similar gasket solution. Test your custom adapter module enclosure for emissions and immunity compliance prior to deployment.

Caution If the size of the enclosure increases, you must provide additional mechanical support external to the FAM to ensure that the Controller for FlexRIO or FlexRIO FPGA module and PXI/PXI Express chassis are not damaged.

Caution Do not exceed the maximum specified current limit on any power rail provided by the Controller for FlexRIO or FlexRIO FPGA module. Exceeding any of these limits disables the adapter module power. The adapter module must use less than 6 W of power internally, and total power dissipation in the adapter module must remain below 6 W due to the thermal capabilities of the enclosure. NI does not recommend using external power supplies for the adapter module. The Controller for FlexRIO and FlexRIO FPGA module warranties are voided when adapter modules use external power. If you must use external power supplies, use circuitry to properly sequence the external supply relative to other FlexRIO power supplies provided by the Controller for FlexRIO or FlexRIO FPGA module. You should also use fuses and reverse polarity protection inside the module.

Caution Drawing more than 6 W of power causes significant heat within the adapter module enclosure. Your custom enclosure must be able to properly dissipate the heat. If it is not able to dissipate the heat, you must monitor the operating temperatures of the module and possible safety issues related to the enclosure. If a large portion of the total power is on a few ICs, then adding a thermal slug with thermal interface material to connect the component to the inside of the housing is the most effective way reduce the internal temperate in the housing. If the power is more distributed throughout the enclosure, then thermal slugs are not an effective method for reducing internal temperature, and the product must be de-rated from the normal 55 °C. In general, when cooling an adapter module at power levels above 6 W, adding holes without forcing air through the enclosure produces only a few

degrees of cooling. Vent hole size for EMI considerations should be no larger than 1/20th of the wavelength that you intend to attenuate. For example, 1/20th of a wavelength of 100 MHz is 15 cm. 1/20th of a wavelength of 5 GHz is 3 mm. Smaller holes are generally better. Holes or slots should not exceed .125 in. Any adapter module exceeding the 6 W limit without forced air cooling will exceed the touchable surface temperature of 55 °C and must be de-rated to lower than the 55 °C operating temperature.

C

Xilinx Documentation References

Xilinx FPGA documentation provides information required for the successful development of your FlexRIO adapter module. The following table provides a list of specific Xilinx documentation resources.

All Xilinx documentation can be found at `www.xilinx.com`.

Table C-1. Xilinx 7-Series FPGA Documentation

Document	Location	Document Part Number	Description
7 Series FGPAs Overview	`http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf`	DS180	Outlines the features and product selection of the Xilinx 7 series FPGAs: Artix-7, Kintex-7, and Virtex-7 devices.
Kintex-7 FPGAs Data Sheet: DC and AC Switching Characteristics	`http://www.xilinx.com/support/documentation/data_sheets/ds182_Kintex_7_Data_Sheet.pdf`	DS182	Contains the DC and AC switching characteristic specifications for the Kintex-7 FPGAs.
Virtex-5 User Guide	`www.xilinx.com/support/documentation/user_guides/ug190.pdf`	UG190	Includes information about clocking resources, clock management technology, phase-locked loops, block RAM, Configurable Logic Blocks (CLBs), SelectIO™ resources, and SelectIO logic resources.

Table C-1. Xilinx 7-Series FPGA Documentation (Continued)

Document	Location	Document Part Number	Description
Virtex-5 Data Sheet	www.xilinx.com/support/documentation/data_sheets/ds202.pdf	DS202	Specifies the electrical characteristics of the Virtex®-5 Platforms, including absolute maximum ratings, recommended operating conditions, supply requirements, and switching characteristics.
Constraints Guide (for Xilinx 9.2.4)	http://toolbox.xilinx.com/docsan/xilinx92/books/docs/cgd/cgd.pdf	—	Describes each Xilinx constraint, including supported architectures, applicable elements, propagation rules, and syntax examples. This manual also describes constraint types and constraint entry methods, provides strategies for using timing constraints, and describes supported third-party constraints.
Virtex-5 FPGA SSO Calculator	https://secure.xilinx.com/webreg/clickthrough.do?cid=30154	—	Functions as a spreadsheet-based tool that automates all the Prolog Functional Data Model (PFDM) and SSO calculations for all I/O standards.

D

NI Services

NI provides global services and support as part of our commitment to your success. Take advantage of product services in addition to training and certification programs that meet your needs during each phase of the application life cycle; from planning and development through deployment and ongoing maintenance.

To get started, register your product at ni.com/myproducts.

As a registered NI product user, you are entitled to the following benefits:

- Access to applicable product services.
- Easier product management with an online account.
- Receive critical part notifications, software updates, and service expirations.

Log in to your NI ni.com User Profile to get personalized access to your services.

Services and Resources

- **Maintenance and Hardware Services**—NI helps you identify your systems' accuracy and reliability requirements and provides warranty, sparing, and calibration services to help you maintain accuracy and minimize downtime over the life of your system. Visit ni.com/services for more information.
  - **Warranty and Repair**—All NI hardware features a one-year standard warranty that is extendable up to five years. NI offers repair services performed in a timely manner by highly trained factory technicians using only original parts at a National Instruments service center.
  - **Calibration**—Through regular calibration, you can quantify and improve the measurement performance of an instrument. NI provides state-of-the-art calibration services. If your product supports calibration, you can obtain the calibration certificate for your product at ni.com/calibration.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. To learn more, call your local NI office or visit ni.com/alliance.

- **Training and Certification**—The NI training and certification program is the most effective way to increase application development proficiency and productivity. Visit ni.com/training for more information.
  - The Skills Guide assists you in identifying the proficiency requirements of your current application and gives you options for obtaining those skills consistent with your time and budget constraints and personal learning preferences. Visit ni.com/skills-guide to see these custom paths.
  - NI offers courses in several languages and formats including instructor-led classes at facilities worldwide, courses on-site at your facility, and online courses to serve your individual needs.
- **Technical Support**—Support at ni.com/support includes the following resources:
  - **Self-Help Technical Resources**—Visit ni.com/support for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on. Registered users also receive access to the NI Discussion Forums at ni.com/forums. NI Applications Engineers make sure every question submitted online receives an answer.
  - **Software Support Service Membership**—The Standard Service Program (SSP) is a renewable one-year subscription included with almost every NI software product, including NI Developer Suite. This program entitles members to direct access to NI Applications Engineers through phone and email for one-to-one technical support, as well as exclusive access to online training modules at ni.com/self-paced-training. NI also offers flexible extended contract options that guarantee your SSP benefits are available without interruption for as long as you need them. Visit ni.com/ssp for more information.
- **Declaration of Conformity (DoC)**—A DoC is our claim of compliance with the Council of the European Communities using the manufacturer's declaration of conformity. This system affords the user protection for electromagnetic compatibility (EMC) and product safety. You can obtain the DoC for your product by visiting ni.com/certification.

For information about other technical support options in your area, visit ni.com/services, or contact your local office at ni.com/contact.

You also can visit the Worldwide Offices section of ni.com/niglobal to access the branch office websites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

Index

G

I

L

M

N

P

R

S

T

U

V

X

Figures

Tables