NC-10 User Manual

NOVATOR
SOLUTIONS

NC-10 User Manual

© 2019 Novator Solutions AB

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Information contained herein may be subject to change without prior notice. For the latest version, refer to www.novatorsolutions.se. Novator Solutions reviews this document carefully for technical accuracy; however, NOVATOR SOLUTIONS MAKES NO EXPRESS OR IMPLIED WARRANTIES AS TO THE ACCURACY OF THE INFORMATION CONTAINED HEREIN AND SHALL NOT BE LIABLE FOR ANY ERRORS.

Please read the End User License Agreement (displayed before installing the software) before using this product.

National Instruments, NI, ni.com, NI FlexRIO, NI USRP and LabVIEW are trademarks of National Instruments. Microsoft and Windows are registered trademarks of Microsoft Corporation. VI Package Manager is a trademark of JKI (jki.net). ChannelCoreFlex is a registered trademark of RFEL, Ltd. Xilinx is a registered trademark of Xilinx, Inc.

This publication is independent of National Instruments and Microsoft, which are not affiliated with the publisher or the author, and do not authorize, sponsor, endorse or otherwise approve this publication.

Revision: 4.0.0
Printed: June 2019 in Sweden

Part I Introduction 4
1 What is NC-10? 5
Part II Getting Started 7
1 System Requirements 8
2 Evaluation 9
3 Installing NC-10 9
4 Licensing 12
5 Examples 20
Part III LabVIEW API 22
1 Programming Flow 26
2 Channelizer API.lvclass VIs 27
3 Target Specific VIs 40
4 NC-10 Error Codes 73
Part IV NC-10 Operation 76
1 RF Input 77
2 Channelizer 78
3 Timestamp 81
4 Synchronization 82
5 RF Output 82
6 Channel Output 83
7 Channel Buffering 83
8 Outputting VITA49 Data 89
9 Troubleshooting 96
Part V Deployment 97
1 Deployment Licensing 98
2 Building Executables 98
Part VI Contact Information 99

© 2019 Novator Solutions AB

Introduction

1 Introduction

This document contains information about NC-10 Toolkit and is divided into the following sections:

- The Introduction contains brief information about the product functionality.
- Getting started gives information about system requirements, evaluation, installation, getting up and running with the API as well as licensing the channelizer.
- LabVIEW API contains detailed information about NC-10 API functions. Read the Programming Flow section to get an overview of how the VIs in the API should be used.
- NC-10 Operation contains more in-depth details about how NC-10 works.

1.1 What is NC-10?

In a wideband signal source there can be many different signals of interest that must be processed further. Processing the entire wideband for every interesting signal is too CPU intensive. Instead, the signals of interest should be extracted from the wideband source to make it possible to process each signal individually.

Novator Solutions NC-10 (Novator Channelizer) is an NI LabVIEW™ Toolkit that lets LabVIEW users leverage powerful industry-leading channelization techniques (NC-10 is powered by ChannelCoreFlex Technology from RFEL LTD., UK) and channelize up to 512 or 1024 channels simultaneously, depending on the target. For most targets, NC-10 comprises two parts, namely a host API and an FPGA bitfile that is configured by the API. The USRP version includes also a server program that can receive TCP/IP commands for configuration. The Server functionality is described in a separate manual. No LabVIEW FPGA programming or licensing is required to use NC-10.

Each of the channels can individually be tuned on center frequency, sample rate, gain, filter selection and if the channel is enabled or not. It is possible to combine narrow and wide channels and have them overlap. The channels can be reconfigured in runtime and reconfiguring one channel does not affect the operation of the others.

NC-10 can run on multiple hardware targets that feature a Xilinx Kintex7 FPGA and it is the only ready-to-run Channelizer solution for NI LabVIEW. Please contact Novator Solutions regarding available targets. Different targets can have different ways to acquire Wideband Signal and also different ways of Configuring NC-10 and Outputting Channel Data.

1.1.1 Wideband Source Acquisition

A wideband source signal that is to be channelized with NC-10 can be acquired in different ways depending on the target.

FlexRIO Target

When NC-10 Channelizer is run on a FlexRIO target, the wideband signal comes via a P2P connection from a P2P capable signal source from NI. The source can be one of various PXIe based signal sources such as VSA, VST or digitizers, or it can be another FlexRIO device, or another FPGA based board. The modularity of the PXIe platform makes it easy to integrate NC-10 into a larger system together with other NI hardware. Furthermore, it is possible to acquire the wideband signal data on a host through a DMA FIFO.

© 2019 Novator Solutions AB

USRP Target

Unlike FlexRIO, USRP has its own RF Frontend, so it is capable of both acquiring wideband signals and performing channelization on the FPGA board. In fact, the USRP has a total of 4 RF inputs. NC-10 offers the possibility of selecting for each channel, which of the four RF inputs the signal should come from. The acquired RF data can be streamed out through UDP on SFP+ port 0 on the USRP unit.

1.1.2 Configuring NC-10

The different hardware targets have different ways of configuring NC-10 Channelizer.

FlexRIO Target

FlexRIO is a PXIe based target and in order to configure it, a PXIe controller is needed. Configuring NC-10 Channelizer requires that the toolkit is run on the controller computer.

USRP Target

A USRP unit can be connected to the PXIe/PCIe platform via a MXI link. This connection is necessary in order to load the bitfile, as explained in Loading USRP Bitfile section. However, once the bitfile is loaded, the USRP can be used as a standalone device that only needs to be connected either to a computer with SFP+ port or to a switch. The toolkit will then use the UDP packets to send the configuration to NC-10. All configuration and operational communication goes through SFP+ port 0 of USRP.

1.1.3 Channel Data Output

The way the channel data is output from the channelizer target FPGA depends on the hardware used.

FlexRIO Target

When NC-10 Channelizer is run on a FlexRIO target, the channelized data can be sent to a LabVIEW application either in Windows through DMA or to other P2P capable FPGAs.

USRP Target

When NC-10 Channelizer is used on a USRP device, the channelized data is streamed in UDP packets through SFP+ port 1. The channel data output format is VITA49.

© 2019 Novator Solutions AB

Getting Started

2 Getting Started

This section contains information about system requirements that need to be fulfilled in order to install the NC-10.

Furthermore, it explains how to evaluate and install the software.

Additional information about licensing needed in order to run the full software functionality is given.

Finally, the provided examples are described.

2.1 System Requirements

- NI LabVIEW™ 2016 or later
- Target drivers compatible with LabVIEW 2016 or later (FlexRIO driver or USRP driver depending on the intended hardware target)
- NI System Configuration
- NI FPGA Hardware (contact Novator Solutions for a list of current supported hardware and roadmap).

*NOTE: LabVIEW FPGA Module is **not** required in order to use NC-10.*

Operating System

NC-10 has been tested on Windows 7 but is expected to work on all Windows operating systems supported by LabVIEW 2016.

PXI Hardware

- NI 7975 or NI 7975
- PXIe Peer-to-Peer enabled Signal source
- PXIe-chassis and controller

USRP Hardware

- NI USRP-2955 or Ettus USRP X310
- 2x SFP+ 10Gb compatible cables
- Host with 2x SFP+ 10Gb connection ports or 10Gbit enabled switch with minimum 2x SFP+ 10Gbit ports. See notes below

Important Notes on Networking and Data Transfer

1. 1Gb/s SFP+ to RJ45 adapters will *not* transmit the 10Gb/s data sent from USRP even if the connector form factors allow the cables to be connected.
2. Network switches with both 10Gb SFP+ ports and 1Gb RJ45 ports are usually able to convert the data flows so the client system can be connected to the USRP via a switch instead of direct cabling.
3. With higher data flows (e.g. high amounts of channels, wideband channels, and multiple wide FFT spectra enabled) the probability of package losses will increase, if both the SFP0 and SFP1 data is managed by the same computer. A more robust solution is to let one computer manage the server application, another to manage

© 2019 Novator Solutions AB

channel data reads (for example with NC-Channel), and possibly a third to manage configuring and FFT display (for example with NC-Config).

2.2 Evaluation

Evaluate NC-10 With Hardware

To evaluate NC-10, a bitfile for the FPGA is required for both FlexRIO and USRP. If a hardware/software bundle is ordered directly from Novator Solutions, the bitfile will be delivered on a USB stick. Otherwise, bitfiles are not included in the delivered package and need to be obtained separately. If the hardware target is NI USRP, a PCIe link needs to be provided in order to load the bitfile. Furthermore, the USRP version requires additional steps such as network configuration before being ready to run.

Installing NC-10 explains in details all the steps required to run NC-10 in evaluation mode. When in evaluation mode, NC-10 will run for approximately ten minutes before it stops outputting data.

For the unlocking full NC-10 functionality refer to Licensing section.

2.3 Installing NC-10

NC-10 uses VI Package Manager™ (included with LabVIEW) to distribute the software API installation files. The package file is *NC-10_4.0.x.vipm*, and it is located either on the USB-stick if the toolkit has been purchased as a bundle, or it can be downloaded from LabVIEW Tools Network. Software API can be installed on any number of computers and is not bound to a specific computer or PXI system.

Depending on the target, follow the steps listed below in order to access the software in Evaluation mode. For unlocking the full functionality of NC-10 refer to Licensing after the installation is finished.

FlexRIO Target

Required steps

1. Install NC-10 from *NC-10_4.0.x.vipm* by following the installation instructions
2. Request the bitfile by e-mailing Novator Solutions at nc10support@novatorsolutions.se
3. Once Novator Solutions sends the information required for obtaining the bitfile, download it and place it in the default location: %Public%\Documents\Novator Solutions\NC-10\FPGA Bitfiles

USRP Target

If NC-10 is used with a USRP unit, additional steps are needed. The sections Update USRP Drivers and NI Register Bus v0 explain in more detail why. Moreover, if NC-10 was *not* purchased as bundle of software and hardware, a bitfile has to be loaded using PCIe link. Finally, network configuration is required.

Required steps

1. Make sure NI USRP driver is installed.

 © 2019 Novator Solutions AB

2. Install NC-10 toolkit from *NC-10_4.0.x.vipm* by following the installation instructions.
3. Request the bitfile by e-mailing Novator Solutions at nc10support@novatorsolutions.se
4. Once Novator Solutions sends the information required for obtaining the bitfile, download it and place it in the default location: %Public%\Documents\Novator Solutions\NC-10\FPGA Bitfiles
5. Connect the USRP through PCIe-link and update its firmware
6. Update the IP address of the USRP

2.3.1 Update USRP Drivers

The NC-10 version for USRP uses UDP packets for both configuring channelizer and acquiring data. In order to be able to support this feature, some VIs from the USRP Drivers installation must be changed.

If you installed the USRP driver before installing NC-10 package you should already have these VIs replaced as the NC-10 package installer takes care of this. However, if you were not the administrator on the computer you installed this toolkit on the installer may have failed to copy these files to the correct location. You can also do this manually as we have included these files in the package installer with instructions in this path: %Public%\Documents\Novator Solutions\NC-10\Include\USRP driver.

2.3.2 NI Register Bus v0

The NC-10 version for USRP uses UDP packets for both configuring channelizer and acquiring data. In order to support this feature, some VIs in NI Register Bus library installation must be changed. If you were not the administrator on the computer you installed this toolkit on the installer may have failed to copy these files to the correct location. You can also do this manually as we have included these files in the package installer with instructions in this path: %Public%\Documents\Novator Solutions\NC-10\Include\Registerbus.

2.3.3 Loading USRP Bitfile

In order to use NC-10 with the USRP, it is necessary to go through a procedure of loading a bitfile via a PCIe-link. If NC-10 was purchased as a software and hardware bundle, USRP comes already loaded with the correct bitfile for the current release.

After the bitfile is loaded, the PCI link is not needed any more and all communication with USRP, both configuration and acquisition of data goes over an Ethernet connection.

NOTE: Some computers might have limited bus enumeration in BIOS for the PCIe bus and this will limit the possibility to communicate with USRP. More and more computer manufacturers use this memory in BIOS for other important implementations and there is nothing in the BIOS or PCIe standard saying you have to have full enumeration. Check this with your computer provider before buying.

We have seen much better performance with Mojo DX to Thunderbolt 2 converter that can easily be converted to USB C or Thunderbolt 3 with a cable adapter. Sonnettech sells this adapter that have proved to work in most cases, but no guarantees are given by Novator Solutions that they work. Contact National Instruments and computer provider when having issues with connecting to your USRP over PCIe-link.

© 2019 Novator Solutions AB

Connection

1. Connect USRP through PCIe while the computer is turned off
2. First power the USRP on, then the computer
3. Open MAX (Measurement and Automation Explorer)
4. Find your USRP device under “Devices and Interfaces” category
5. Click the "Update firmware" button in MAX.

Reset to NI Default bitfile

1. Select file *usrp_x310_fpga_HG* from "%programfiles(x86)%\National Instruments\NI-USRP\images"
2. After the Firmware load has been completed you are prompted to reboot the device, this is a mandatory step in order to be able to run the new bitfile
3. Power off the computer
4. Power cycle the USRP
5. Power on the computer a few seconds after powering on the USRP.

Loading custom NC-10 bitfile

1. Repeat the same procedure as in Reset to NI Default Bitfile but this time select NC-10 bitfile provided by Novator Solutions instead

> *NOTE: The default IP Address of port 0(10G) is 192.168.30.2 and port 1(10G) is 192.168.30.3 when delivered. Use the Network Configurator to change these to match your network topology.*

2.3.4 Network Configuration

In order to run NC-10 with the USRP it is necessary to configure the Ethernet settings on the USRP device.

A separate program called "USRP Network Configurator" is used to configure the Ethernet settings of NC-10 USRP. This program allows you to configure the IP Address and the Subnet Mask for each port and the Default Gateway that applies to both ports. The program itself contains a detailed help explaining how to run it.

Set the new IP Addresses, Subnet Masks and Default Gateway for the SFP+ ports of the USRP by following the steps below:

1. Download the USRP Network Configurator from http://novatorsolutions.com/spectral-data-analysis/nc10gettingstarted/
2. Run the application and follow the instructions in the software.

© 2019 Novator Solutions AB

NOTE: Make sure that jumbo frames are enabled and set to 9014 Bytes.

See also important notes on network connectivity under "System Requirements"

2.4 Licensing

To run NC-10 in a fully operational mode, a license file is required. Without a valid license file, the FPGA will work for approximately 10 minutes before it stops outputting data (so-called *timebombed* execution).

Each NC-10 license is tied to a specific FPGA-target, and the license file can contain licenses for multiple targets. When the Open FPGA.vi is called, the API automatically searches for the serial number of the hardware resource wired to the target specific Create method. If the serial number is found in the license file, the corresponding NC-10 key is used to unlock the full functionality of NC-10. If the serial number is not found in the license file or the key is not valid, an error is returned when calling Open FPGA.vi.

Obtaining a License Key section contains step-by-step guide for obtaining the key.

2.4.1 Obtaining a License Key

Steps required for license management are listed here:

1. Download NC-10 FlexRIO License Manager or NC-10 USRP License Manager from http://novatorsolutions.com/spectral-data-analysis/nc10gettingstarted/ depending on the target in use. Follow the instructions in the program to generate License Request File.
2. Email the obtained License Request File from step 1 to nc10support@novatorsolutions.se
3. An email with a license file will be emailed to you within 2 business days.
4. Use the same target specific License Manager program to verify the license and place it in the default location or move it manually to : %Public%\Documents\Novator Solutions\NC-10\Licenses

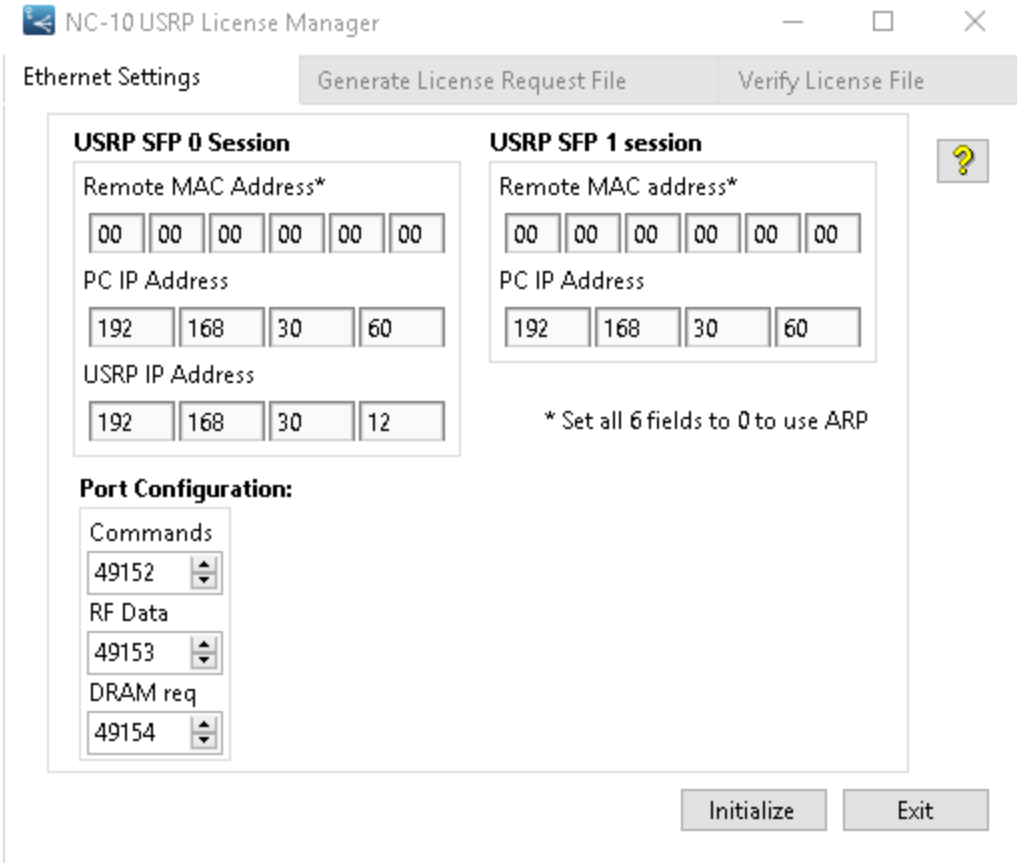
2.4.2 NC-10 USRP License Manager

NC-10 USRP License Manager is a utility for generating License Request File for a USRP unit. It can also be used to verify a License File and place it in the correct location after the license has been received from Novator Solutions. The utility can be downloaded from http://novatorsolutions.com/spectral-data-analysis/nc10gettingstarted/

Generating License Request File

© 2019 Novator Solutions AB

1. Start the program. Verify that the PC IP Address and USRP IP Address match your system configuration

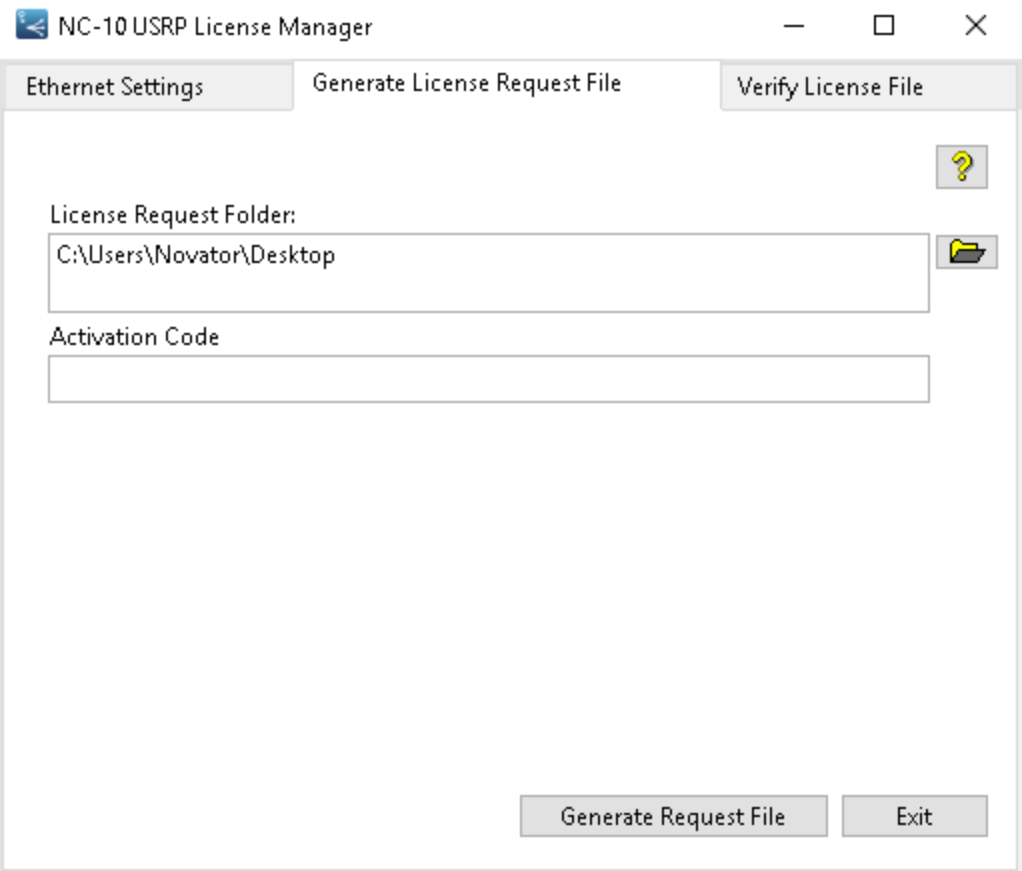


   a. PC IP Address should match the IP address of your computer's network adapter that is connected to USRP port SFP0.
   b. USRP IP Address should match the IP address of the SFP0 port in the USRP.

> *NOTE: This utility cannot be used to change the network settings. Use USRP Network Configurator for that purpose*

2. Press "Initialize". If the PC IP Address is correct, the program will switch to the next screen (see Figure below). Make sure to enter the Activation Code if NC-10 was purchased through LabVIEW Tools Network.

© 2019 Novator Solutions AB



The License Request File will be created in the indicated folder.

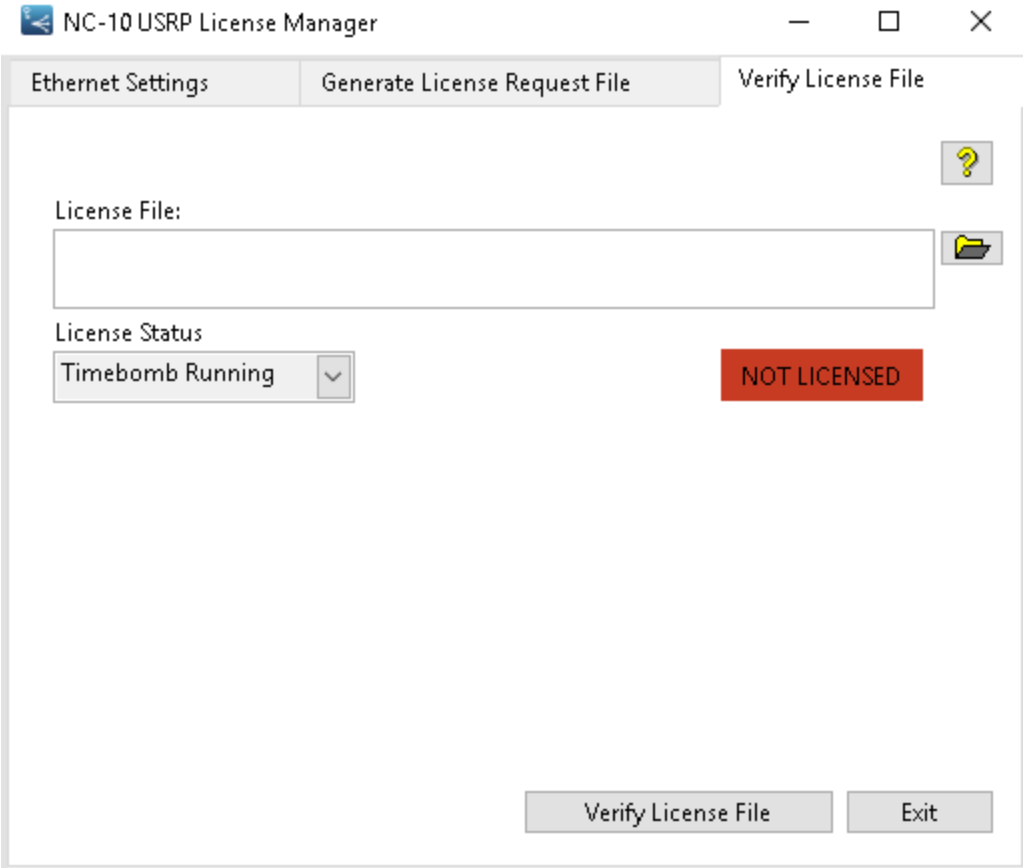
3. Click "Generate Request File". If successful, a text file is going to be created in the License Request Folder, for example "20180413103444 NC-10 LicenseRequest.txt" where the initial part indicates the request file creation date and time.

4. Refer back to the section on "Obtaining a License Key" for next steps.

License File Verification

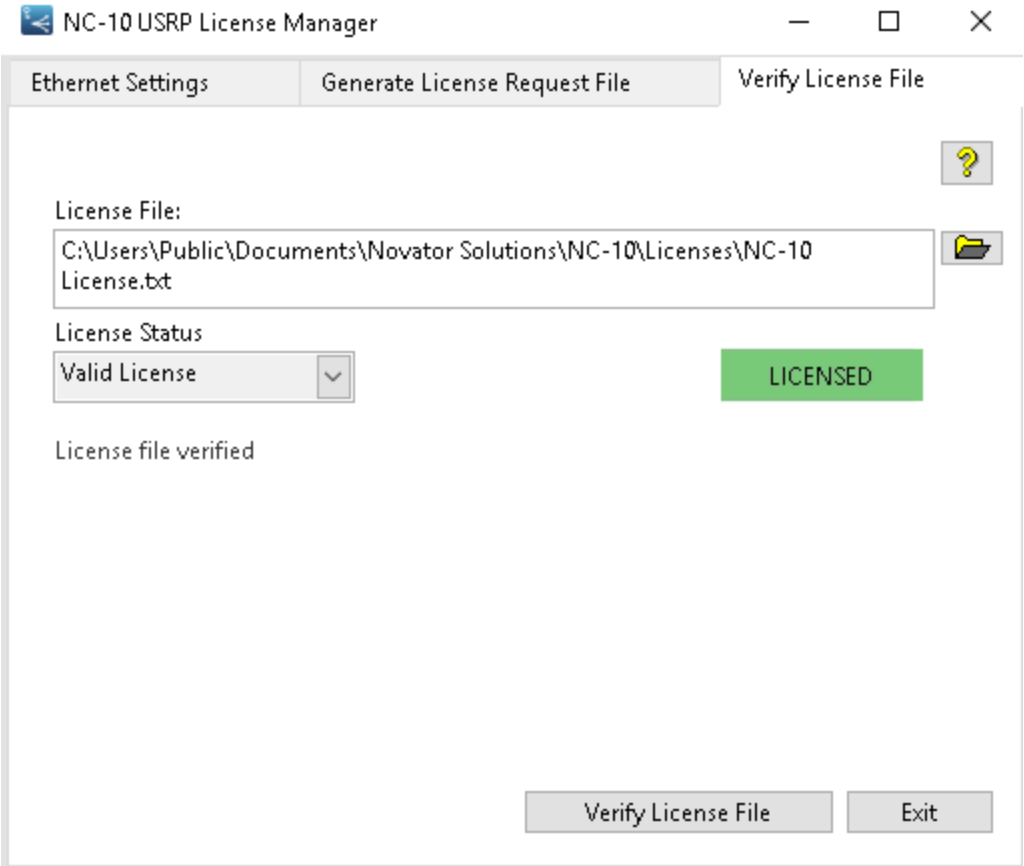
© 2019 Novator Solutions AB

1. If you have a license file installed, you can choose the "Verify License File" tab. Fill in the path to the license file you have obtained from Novator Solutions and click "Verify License File".



© 2019 Novator Solutions AB

2. If the license status is Valid as in the figure below, the program will copy the license file to the default location %Public%\Documents\Novator Solutions\NC-10\Licenses.



2.4.3 NC-10 FlexRIO License Manager

NC-10 FlexRIO License Manager is a utility for generating License Request File for a FlexRIO unit. It can also be used to verify a License File and place it in the correct location after the license has been received from Novator Solutions. The utility can be downloaded from http://novatorsolutions.com/spectral-data-analysis/nc10gettingstarted/

Generating License Request File

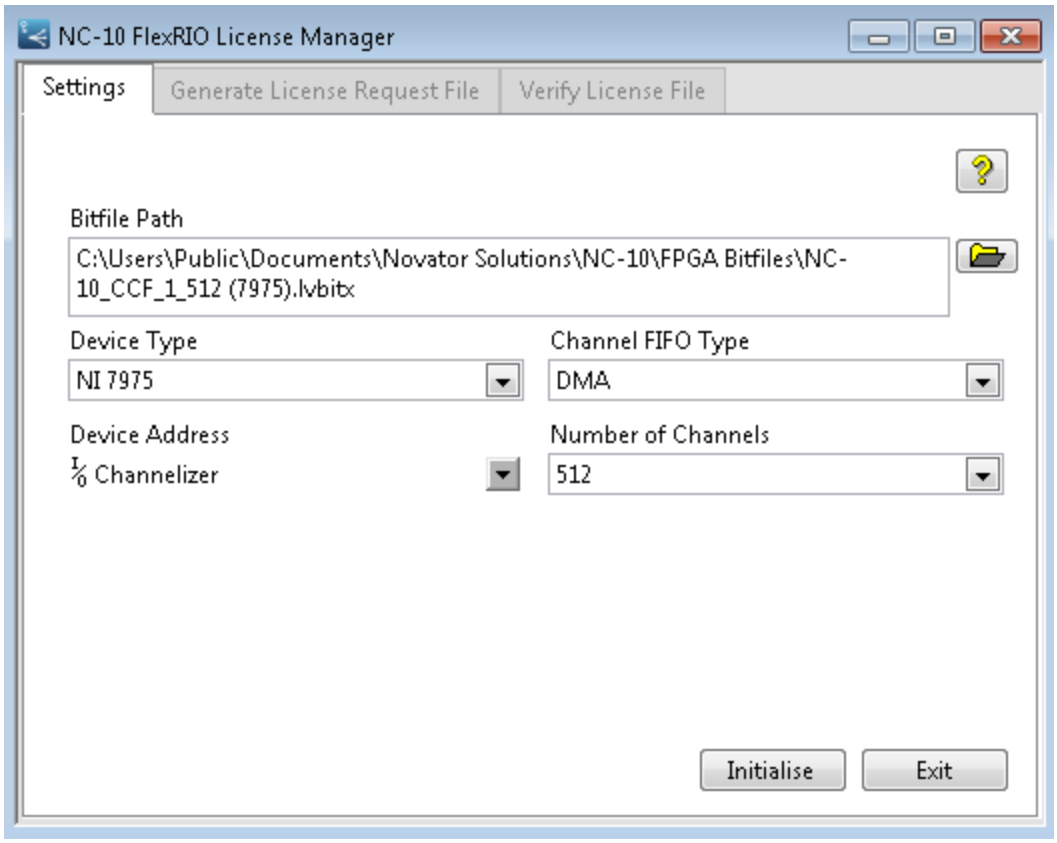
1. Start the program. Make sure that Device Type, Channel FIFO Type and Number of Channels settings correspond to the ones that specified bitfile is expecting:

Bitfile	Device Type	Channel FIFO Type	Number of Channels
NC-10_CCF_1_512 (7975).lvbitx	NI 7975	Any	512
NC-10_CCF_1_1024 (7975) P2P.lvbitx	NI 7975	P2P	1024
NC-10_CCF_1_1024 (7975) DMA.lvbitx	NI 7975	DMA	1024

© 2019 Novator Solutions AB

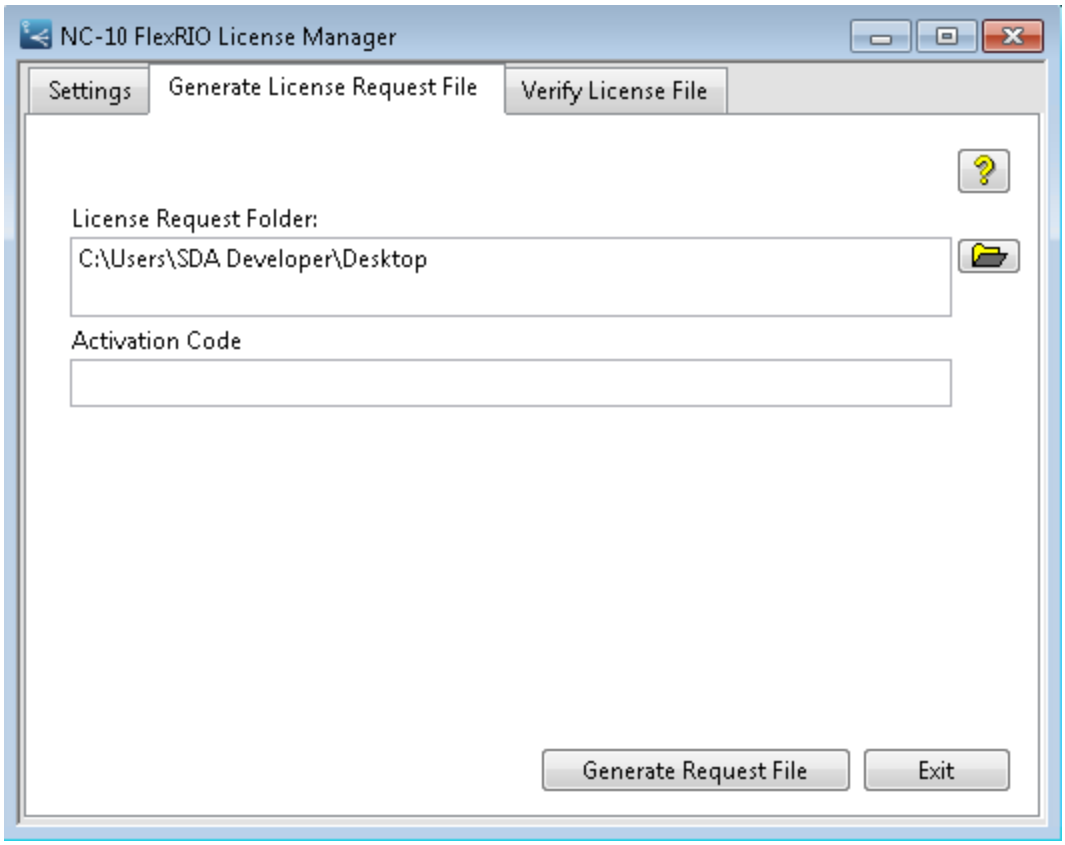
NC-10_CCF_1_512 (7976).lvbitx	NI 7976	Any	512

The Device Address parameter must match the device identifier configured in MAX.



2. Press "Initialize". The program will switch to the next screen (see Figure below). Make sure to enter Activation Code if NC-10 was purchased through LabVIEW Tools Network.

© 2019 Novator Solutions AB



The License Request File will be created in the indicated folder.

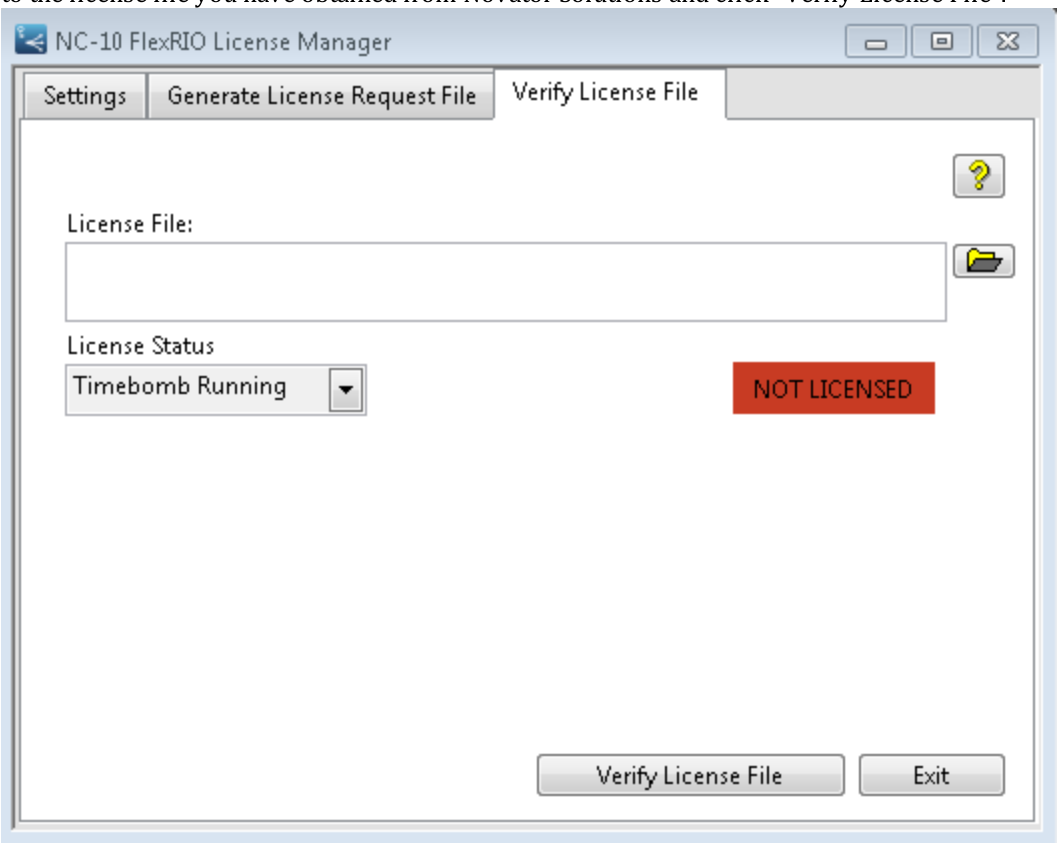
3. Click "Generate Request File". If successful, a text file is going to be created in the License Request Folder, for example "20180413103444 NC-10 LicenseRequest.txt" where the initial part indicates the request file creation date and time.

4. Refer back to the section on "Obtaining a License Key" for next steps.

License File Verification

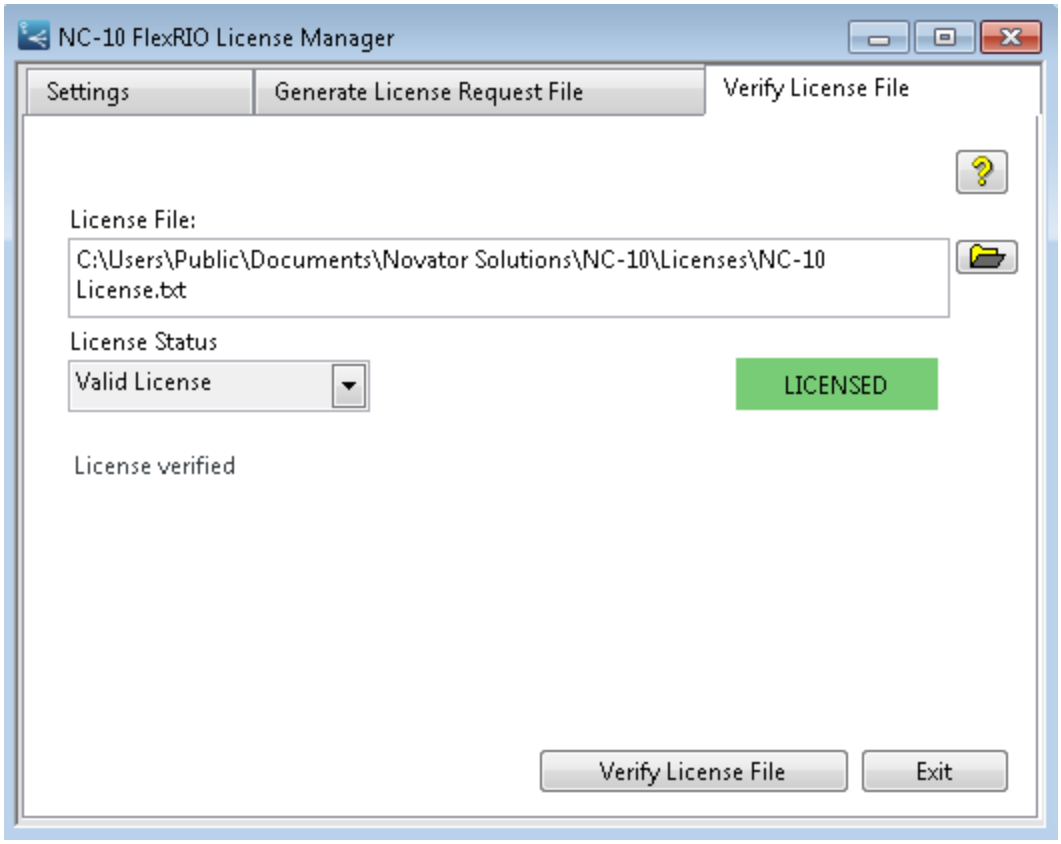
© 2019 Novator Solutions AB

1. If you have a license file installed, you can choose the "Verify License File" tab. Fill in the path to the license file you have obtained from Novator Solutions and click "Verify License File".



2. If the license status is Valid as in the figure below, the program will copy the license file to the default location %Public%\Documents\Novator Solutions\NC-10\Licenses.

© 2019 Novator Solutions AB



2.5 Examples

The examples can be found in the LabVIEW examples folder, subfolder Novator Solutions. The examples are divided into two categories, FlexRIO and USRP. Note that the examples require that the drivers for the respective target are installed. Additionally, the USRP examples require that the drivers are updated with the Novator Solutions code as described in Update USRP Drivers and NI Register Bus v0.

Below is a summary of the shipping examples. For more information refer to the comments in the individual example programs.

FlexRIO Examples

File Name	Comments
Read Channel Data with Custom Headers.vi	This example shows several methods for inserting custom headers into channel data packages and how the headers are extracted at reception.
Read Channel Data with Old Samples.vi	Channel data are received in buffers and this example shows how samples from the

© 2019 Novator Solutions AB

	previous buffer can be added to the current one.
Read Channel Data.vi	A simple example for reading a single channel.
NC-10 Application Example.vi	A complete application tying together several features described in the three examples above.
Read RF Data.vi	A simple example for streaming the RF data up to the host.
Read RF Data with FFT Analysis.vi	An example showing how the host can perform analysis on the raw RF data, for example in order to locate channels of interest from the RF stream.

The functionality of NC-10 is dependent on a signal source and the examples are written to run on a NI-VSA using the NI-RFSA driver. The NI-VSA Targets used to verify examples were NI 5667 and NI 5665. The P2P signal source can also be replaced to use other signal sources. Some targets such as NI 5668 may require additional changes in the examples code.

To minimize the necessary hardware required to run the code, all examples stream channel data to the host and not peer-to-peer.

USRP Examples

File Name	Comments
USRP Read Channel Data.vi	A simple example for reading a single channel.
USRP Application Example.vi	A complete application tying together several features.
USRP Read RF Data.vi	A simple example showing how to read RF data from the UDP stream
USRP Timestamp.vi	An example reading timestamped channel and RF data and showing how timestamping can be done relative to GPS time and also relative to the computer time.

Unlike FlexRIO, the USRP target has its own RF Frontend. Therefore, the examples include RF configuration to acquire data. All data and communication is handled through UDP and NI Register Bus library.

NOTE: To run any USRP example load them in a LabVIEW project and set a project property for Conditional Disable Symbol to ETHERNET=TRUE. Read the readme file in example folder for more detailed explanation.

© 2019 Novator Solutions AB

LabVIEW API

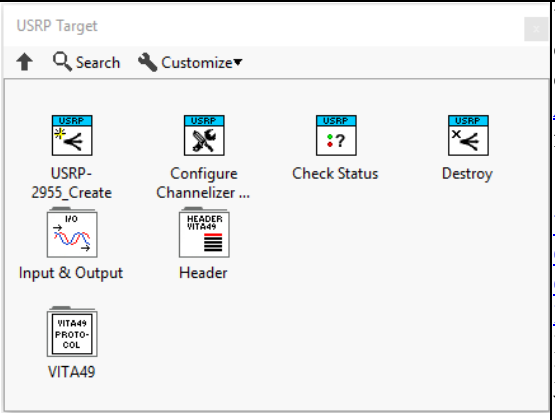
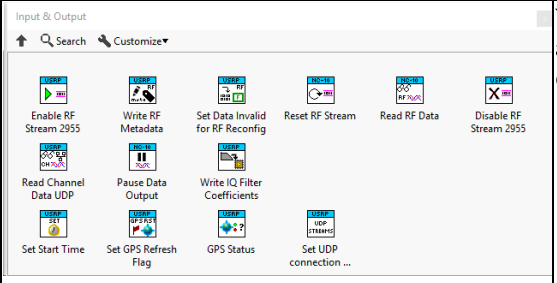
3 LabVIEW API

The API is divided into two main categories of VIs: NC-10 Core VIs and NC-10 Target Specific VIs. NC-10 Core are VIs common to all targets while NC-10 Target Specific VIs are different for each FPGA target system. An example is configuring the center frequency of a channel which is the same for all targets, while configuring the channel output destination is dependent on the target.

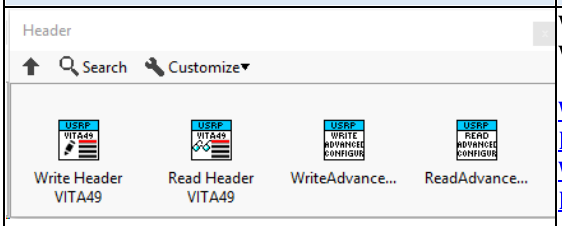
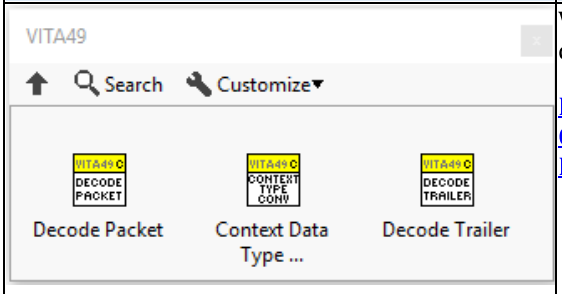
The NC-10 API is reference-based, which means that you can fork the wire connected to the upper left and upper right terminals and still access the same data (as in a queue reference).

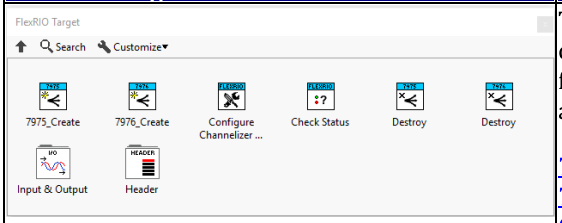
The API consists of the following palettes:

Root Palette	
	USRP Target (subpalette) FlexRIO Target (subpalette) General (subpalette) Channel Config (subpalette)

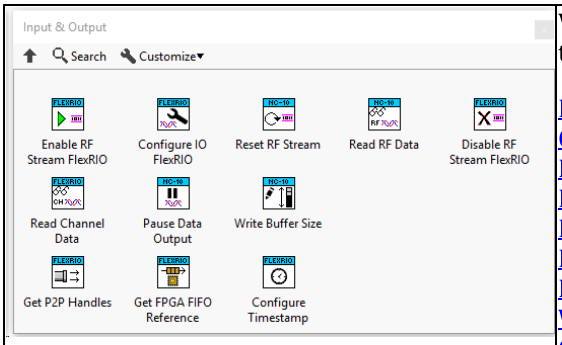
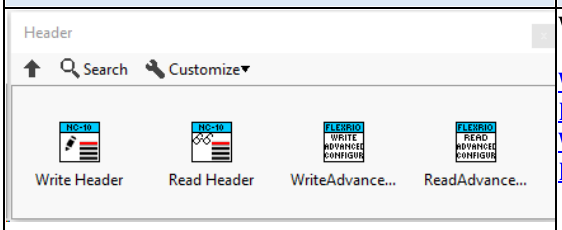
USRP Target	
	USRP-2955 target specific functions to configure, control, and run channelizer. This class is a child class of Channelizer API.lvclass and can call on their parents methods as well. USRP-2955_Create.vi Configure Channelizer 2955.vi Check Status.vi Destroy.vi Input & Output (subpalette) Header (subpalette) VITA49 (subpalette)
Input & Output	
	VI's to control and read IO data from tuners and channels as well as reading status of different parts of the NC-10 USRP.

© 2019 Novator Solutions AB

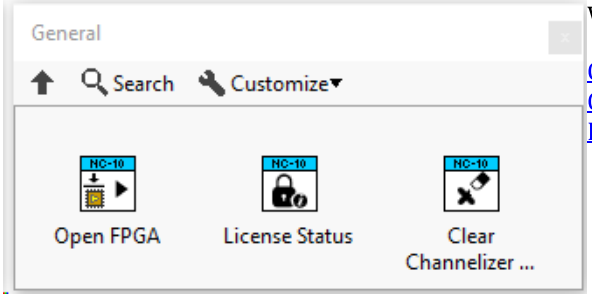
	Enable RF Stream 2955.vi Write RF Metadata.vi Set Data Invalid for RF Reconfig.vi Reset RF Stream.vi Read RF Data.vi Disable RF Stream 2955.vi Read Channel Data UDP.vi Pause Data Output.vi Write IQ Filter Coefficients.vi Set Start Time.vi Set GPS Refresh Flag.vi GPS Status.vi Set UDP Connection Streams.vi
Header	
	VI's to write and read header information in VITA49 standard. Write Header VITA49.vi Read Header VITA49.vi WriteAdvancedConfiguration.vi ReadAdvancedConfiguration.vi
VITA49	
	VI's to pack and unpack and convert VITA49 data from streams. Decode Packet.vi Context Data Type Conversion.vi Decode Trailer.vi

FlexRIO Target	
	Target specific VIs for FlexRIO. This palette contains configuration and control functions for two different FlexRIO targets 7975.lvclass and 7976.lvclass. 7975_Create.vi 7976_Create.vi Configure Channelizer FlexRIO.vi Check Status.vi Destroy.vi Destroy.vi Input & Output (subpalette) Header (subpalette)
Input & Output	

© 2019 Novator Solutions AB

	VIs for handling input and output to and from the FPGA. Enable RF Stream FlexRIO.vi Configure IO FlexRIO.vi Reset RF Stream.vi Read RF Data.vi Disable RF Stream FlexRIO.vi Read Channel Data.vi Pause Data Output.vi Write Buffer Size.vi Get P2P Handles.vi Get FPGA FIFO Reference.vi Configure Timestamp.vi
Header	
	VIs for writing/reading headers Write Header.vi Read Header.vi WriteAdvancedConfiguration.vi ReadAdvancedConfiguration.vi

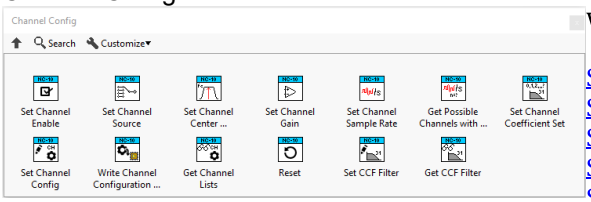
General



VIs for initialization, cleaning up, status etc.

Open FPGA.vi
Clear Channelizer Error.vi
License Status.vi

Channel Configuration



VIs to configure the channelization process

Set Channel Enable.vi
Set Channel Source.vi
Set Channel Center Frequency.vi
Set Channel Gain.vi
Set Channel Sample Rate.vi
Get Possible Channels with Sample Rate.vi
Set Channel Coefficient Set.vi
Set Channel Config.vi
Write Channel Configuration to FPGA.vi
Get Channel Lists.vi
Reset.vi
Set CCF Filter.vi
Get CCF Filter.vi

© 2019 Novator Solutions AB

3.1 Programming Flow

For the correct behavior, NC-10 must be initialized before applying channel configurations etc. Refer to the examples for the correct order of operations.

Initialization/Open

The VIs used for initialization are found in the General Palette as well as in the target specific palettes, and are called once per session.

These VIs must be called before any other VI and in the following order:

- 7975_Create.vi (Target specific, see 7975.lvclass VIs), 7976_Create.vi (Target specific, see 7976.lvclass VIs) or USRP-2955_Create.vi (Target specific, see USRP-2955.lvlclass VIs)
- Configure Channelizer FlexRIO.vi (Target specific, this VI is dynamically dispatched to child implementation depending on the Reference Input) or Configure Channelizer 2955.vi

When Open FPGA has been called, other VIs in the API can be called.

If FlexRIO is used and data should be streamed via peer-to-peer, the link should have been set up by getting peer-to-peer handles from Get P2P Handles.vi before the channelization begins.

For USRP, UDP connections should be open before running any of the USRP-2955.lvclass VIs. USRP UDP connection is saved in an FGV and used in both USRP RIO driver and NC-10 API. The FGV needs to be reset and written between opening the connections to UDP and USRP-2955_Create.vi as it is required for proper communication.

Main Application

After the initialization the rest of the VIs in the API can be called. Channels are configured and committed to the FPGA using the VIs in the Channel Configuration palette. No channelization settings are committed to the FPGA until Write Channel Configuration to FPGA.vi is called. The channel settings can be reconfigured and committed during application execution.

Depending on the target, channelized data can be streamed either to host, to another FPGA or both for FlexRIO. If USRP is used, the data is always sent over UDP. NI 7976 supports sending channel data to both DMA and P2P FIFO whereas on NI 7975, *Channel FIFO Type* input of 7975_Create.vi is used to determine the destination of the channel data. If the data is streamed to the host, the data can be read with VIs in the IO palette for the specific target. If the data is streamed to another FPGA, that FPGA application must be programmed separately.

If the incoming RF data should be sent to host, the VI Enable RF Stream FlexRIO.vi must be called. If the data is sent to host, VIs to read the data are found in the IO palette. When using USRP you always need to call Enable RF Stream 2955.vi for the RF data to be streamed out on SFP+ port 0.

Clean Up/Close

If RF data flow to the host has been enabled, call Disable RF Stream FlexRIO.vi or Disable RF Stream 2955.vi to stop the data flow. Then call target specific Destroy.vi last of all VIs to close down references.

© 2019 Novator Solutions AB

3.2 Channelizer API.lvclass VIs

The Reference of this class is obtained by calling the "Create" function of the target specific child class, either 7975 Create.vi, 7976 Create.vi or USRP-2955_Create.vi.

The following common input and output parameters are not described under the individual API function descriptions:

Common Input Parameters

Reference in is the obligatory class reference variable.

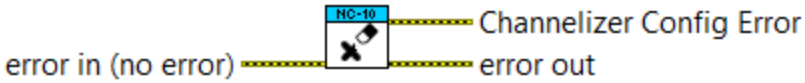
Error in is the standard LabVIEW error cluster.

Common Output Parameters

Reference out is a copy of the “Reference in” parameter.

Error out is the standard LabVIEW error cluster updated with errors raised by the current VI, if any.

3.2.1 Clear Channelizer Error.vi

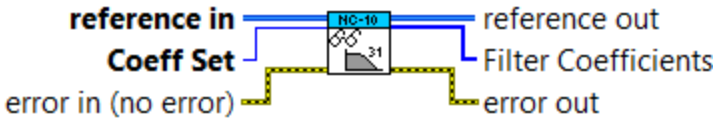


Clears error codes that are in the NC-10 error code range (403200-403999).

Output Parameters

Channelizer Config Error contains the cleared error.

3.2.2 Get CCF Filter.vi



This VI retrieves the coefficients of a symmetric filter from specified Coefficient Set. See Output Shaping Filter for further details about the filter.

Input Parameters

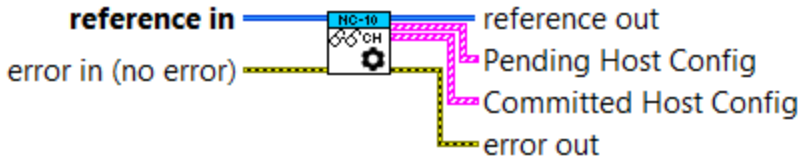
Coeff Set is the coefficient set to retrieve.

Output Parameters

Filter Coefficients is an array of first 16 elements of a 31-tap symmetric filter.

© 2019 Novator Solutions AB

3.2.3 Get Channel Lists.vi



This VI returns all channels currently pending for update and already committed channels. The pending channel list indicates how the channel configuration will look like after the next commit to the FPGA and includes all channels, also channels not modified since the last commit.

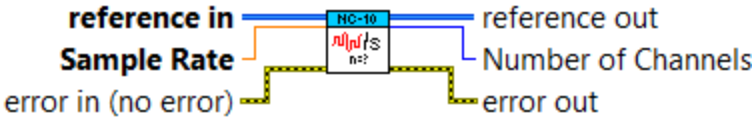
Output Parameters

Pending Host Config, an array of clusters with the following elements:

- **Ch ID** is the unique channel identifier (0-511 or 1023).
- **Enable** enables/disables the channel. If TRUE, the channel is enabled, otherwise disabled.
- **Source** is the RF input source. Except for USRP targets, this should be 0.
- **Fine Gain** is the fine gain in dB.
- **Coarse Gain** is the coarse gain in dB.
- **Center Freq** is the center frequency of the extracted channel, given in Hz. The valid range is $\pm f_{in}/2$.
- **Sample Rate** is the output sample rate of the channel.
- **Coarse BW** is the coarse bandwidth of the channel.
- **Coeff Set** is the coefficient set to be used for the channel with Channel ID.

Commited Host Config, array of clusters with the same data structure as Pending Host Config.

3.2.4 Get Possible Channels with Sample Rate.vi



The VI calculates the number of channels with the input sample rate that can be added to the current configuration without exceeding the aggregate output rate limit. This VI does not allocate the channels. See Channel Settings for further details.

Input Parameters

Sample Rate is the desired sample rate for the channels.

Output Parameters

© 2019 Novator Solutions AB

Number of Channels is the number of channels that can be allocated with the sample rate.

3.2.5 License Status.vi

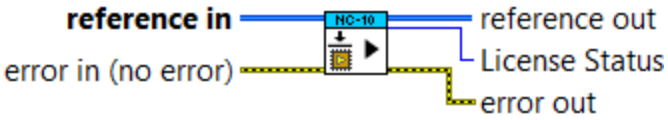
When running the time limited version of NC-10, this VI is used to determine if the FPGA code is still running (status: Timebomb Running) or not (status: Core Locked). If NC-10 has already been licensed, calling this VI is not necessary other than to verify that the license is valid.

Output Parameters

License Status displays the current license status (Valid License, Valid Device Specific, Timebomb Running, Core Locked).

© 2019 Novator Solutions AB

3.2.6 Open FPGA.vi



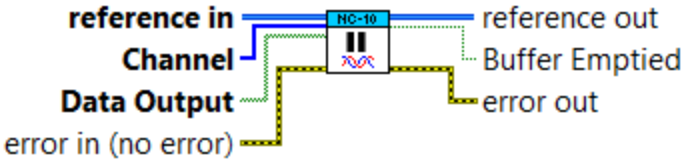
This VI opens a reference to the FPGA for FlexRIO target. Some targets such as USRP do not open a connection to FPGA directly but they send important initialization steps to FPGA through UDP. This VI should be called one time per session.

If there is no license file in the default license file location, NC-10 will run in timebombed execution mode.

Output Parameters

License Status displays the current license status (Valid License, Valid Device Specific, Timebomb Running, Core Locked).

3.2.7 Pause Data Output.vi



The VI allows the user to pause data output for a specific channel without stopping the channelization process. This means that the channel count in the dynamic header will keep counting up. When paused, all data will be lost. Overflow indicators will also be true for all channels paused. Clear them when channels are resumed.

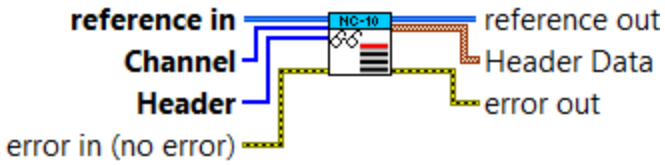
Input Parameters

Channel specifies which channels to configure.
Data Output specifies if the data output should be paused or not for that specific channel.

Output Parameters

Buffer Emptied indicates that that the data in the buffers before calling this VI have been sent out.

3.2.8 Read Header.vi



© 2019 Novator Solutions AB

This VI reads user-configurable headers from the FPGA. The headers are output together with the channel data and can be used for metadata, etc. See Data Header section for further details. This VI can be used in conjunction with Write Header.vi to make sure that data actually was written to headers.

To optimize performance if verification of written headers is needed, the recommended practice is to only read the first and last header from the latest write operation.

Even though this VI is accessible for USRP target as well, it is not recommended to use this as it is used internally within Read Header VITA49.vi with fixed configuration.

Input Parameters

Channel specifies which channels should be read (0-511 or 1023). The channel index corresponds to the header and header data index.

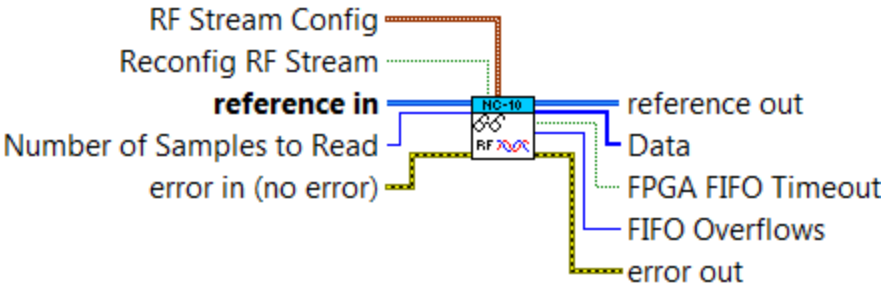
Header specifies which of the available headers should be read (ID Header, User defined, Footer and Dynamic Header).

Output Parameters

Header Data sets the data of the configured header. Header data is a cluster with 16 elements, all I32:

Pos 0 … **Pos 15**

3.2.9 Read RF Data.vi



This VI is used for reading RF data in an application. With FlexRIO target the data is read from DMA FIFO whereas with a USRP the data is read from a UDP stream.

If Reconfig RF Stream is set to true, the RF Stream parameters are updated in this VI. Data can either be sent from FPGA in bursts of fixed length or continuously. In order to create continuous streaming, set Number of Samples to Skip in RF Stream Config to 0.

It is important to read the DMA FIFO/UDP connection often enough to avoid buffer overflows. The higher the RF rate, the more frequently the stream must be read.

To optimize speed the **RF Stream Configuration** cluster is only updated when **Reconfig RF Stream** is TRUE.

FlexRIO Targets

© 2019 Novator Solutions AB

This VI reads data from the RF DMA FIFO buffer and makes it available in the application. The size of the DMA buffer is configured in Configure Channelizer FlexRIO.vi.

If buffer overflow is occuring and more optimized performance is required, it is possible to use a reference to the DMA FIFO transferring the channel data to the host. This can be done by calling Get FPGA FIFO Reference.vi

By reading 0 samples only the FPGA FIFO Timeout is read. This is useful when using the DMA FIFO reference to read the data but still want to keep track of buffer overflows. The timeout is cleared with Check Status.vi.

The reference to the channelizer FPGA is updated at first call and when **Reconfig RF Stream** is TRUE. Thererfore, if the FPGA reference is closed explicitly at some stage, **Reconfig RF Stream** needs to be set to true for one call of this VI to avoid run-time errors.

USRP Target

This VI reads data from UDP connection. Data is sent out using VITA49 protocol. Some of the inputs are not used when using the VI with USRP, these are specified under input and output section.

The timeout and overflow outputs are updated every 1 second.

Example:

An FFT of the incoming data is often needed to get an overview of the signal frequency content. However, continuously streaming the full RF bandwidth could strain the overall system bandwidth. Instead a fixed number of samples at fixed time interval can be sent from the FPGA, e.g. if 1024 samples should be sent and then not send any data the next 32768 samples. In that case, set Number of Samples to 1024 and Number of Samples to Skip in RF Stream Config to 32768.

Input Parameters

Number of Samples
If it is equal to -1, then all available samples will be read from FPGA. Note that if Number of Samples to Skip value in RF Stream Config is greater than 0, then not all RF data will be stored in this FIFO.
If it is 0, then zero data points and current timeout status will be read
If it is greater than 0, then set amount of points will be read with a timeout of 5s.
For USRP this input is not used at all, data packets always have 2048 samples if the packet is valid.

Reconfig RF Stream sends the RF Stream Config parameter values to FPGA and updates the FPGA reference for FlexRIO for the read functionality.

RF Stream Config sets the parameters for a DMA or UDP transfer of the RF data from the channelizer instrument to the host. The contents of this parameter are the same as in Configure Channelizer FlexRIO.vi or Configure Channelizer 2955.vi . For USRP Host Buffer Size is ignored.

Output Parameters

Data contains the RF data stream read from the FPGA. I32 data type represents concatenated I and Q data and in order to get their values, **Data** should be split into

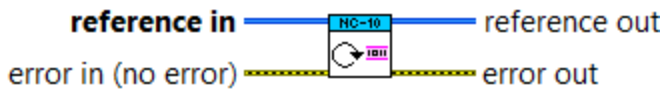
© 2019 Novator Solutions AB

two 16-bits numbers. Most significant 16 bits correspond to Q and least significant to I data. Since splitting function gives U16 data type, I and Q should finally be converted to I16.
For USRP VITA49 decoding is required.

FPGA FIFO Timeout indicates if there is an overflow when sending data to UDP for USRP or output destination for FlexRIO.

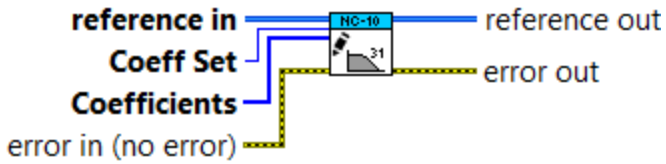
FIFO Overflows indicates the number of times FIFO has overflown.

3.2.10 Reset RF Stream.vi



The VI resets the RF acquisition by stopping the transfer, clearing the internal buffers and then restarting. It also clears the FPGA FIFO Timeout Boolean (see Read RF Data.vi)

3.2.11 Set CCF Filter.vi



This VI configures and applies the filter coefficients of one of the 31 tap filters to the FPGA. No extra commit call is necessary when updating the filters. If a filter of an enabled channel is updated, transitional effects can be seen in the data. See Output Shaping Filter for further details about the filter.

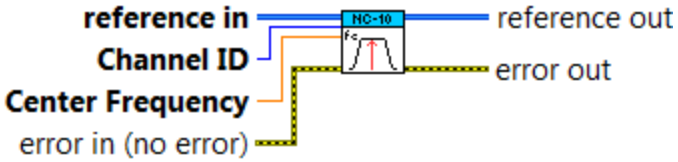
Filter set 0 are predefined to zero out any channel using this set.

Input parameters

Coeff Set sets which coefficient set that should be configured.

Coefficients is an array of first 16 elements of a 31-tap symmetric filter.

3.2.12 Set Channel Center Frequency.vi



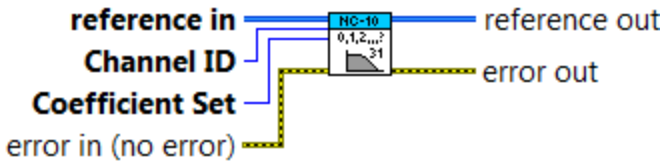
This VI configures the relative center frequency of the channel ($\pm f_{in}/2$).

Input Parameters

© 2019 Novator Solutions AB

Channel ID is the unique channel identifier (0…511 or 0…1023).
Center Frequency is the center frequency of the extracted channel, given in Hz. The valid range is $\pm f_{in}/2$.

3.2.13 Set Channel Coefficient Set.vi



Each channel is run through one of 32 filters and this VI configures which of the 32 coefficient sets to use for the 31-tap filter.

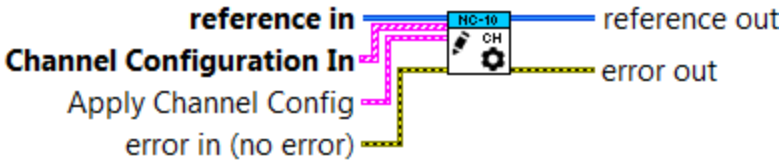
The coefficients of the filters are configured with Set CCF Filter.vi

Input Parameters

Channel ID is the unique channel identifier (0…511 or 0…1023).
Coefficient Set is the coefficient set to be used for the channel with Channel ID.

© 2019 Novator Solutions AB

3.2.14 Set Channel Config.vi



This VI writes all channel parameters at once. The Channel Configuration can be a subset of the channels (e.g. channel 34-78 and 564-675 etc.) and by using Apply Channel Config it is possible to configure which parameters that should be applied. If an error occurs in any channel, none of the channel configurations will be applied.

Input Parameters

Channel Configuration In, an array of clusters with the following elements:

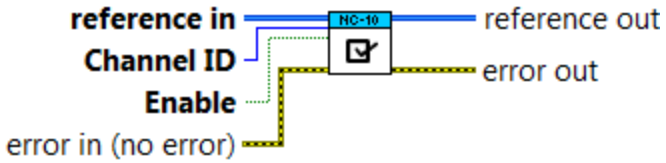
- **Ch ID** is the unique channel identifier (0…511 or 0…1023).
- **Source** is the RF input source. Except for USRP targets, this should be 0.
- **Enable** enables/disables the channel. If TRUE, the channel is enabled, otherwise disabled.
- **Gain** is the fine gain in dB.
- **Coarse Gain** is the coarse gain in dB.
- **Center Freq** is the center frequency of the extracted channel, given in Hz. The valid range is $\pm f_{in}/2$.
- **Sample Rate** is the output sample rate of the channel.
- **Coarse BW** is the coarse bandwidth of the channel.
- **Coeff Set** is the coefficient set to be used for the channel with Channel ID.

Apply Channel Config is a boolean cluster defining which parameters should have changes applied.

- **Enable** controls if Enable should be applied.
- **Source** controls if Source should be applied.
- **Gain** controls if Fine Gain and Coarse Gain should be applied.
- **Center Frequency** controls if Center Frequency should be applied.
- **Sample Rate** controls if Sample Rate and Coarse BW should be applied.
- **Coeff Set** controls if Coeff Set should be applied.

© 2019 Novator Solutions AB

3.2.15 Set Channel Enable.vi

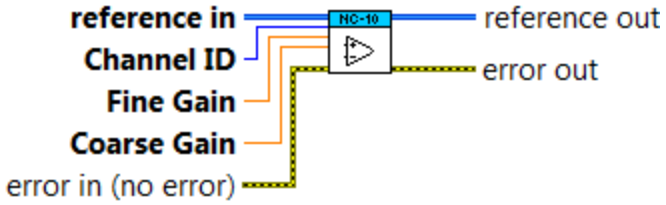


This VI enables or disables the channel.

Input Parameters

Channel ID is the unique identifier (0…511 or 0…1023)..
Enable enables/disables the channel. If TRUE, the channel is enabled, otherwise disabled.

3.2.16 Set Channel Gain.vi



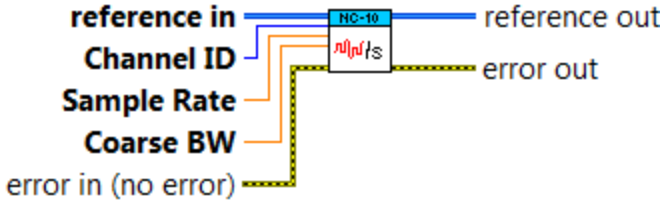
This VI configures the gain settings in the channelizer. When no additional gain is applied, the natural gain of each channel is normalized so that a full-range narrow-band carrier input with a frequency aligned with the centre of the channel pass-band will occupy half of the full range of the complex output. This is to avoid overshoots and ripple effects. Configuring the gain to 6dB will give the original signal amplitude.

Two different gain settings exist for each channel, coarse and fine gain. To optimize dynamic performance the coarse gain should be as high as possible without causing any overflows. An overflow will lead to corrupted data which most likely will be unusable. Coarse gain overflow will result in wrapped data while fine gain overflow results in a clipped signal.

Input Parameters

Channel ID is the unique channel identifier (0…511 or 0…1023).
Fine Gain is gain in dB, 0-102.3 dB in 0.1 dB increment
Coarse Gain is gain in dB, 0-42.14 dB in 6.02 dB steps. Values are coerced down to the nearest multiple of 6.02.

3.2.17 Set Channel Sample Rate.vi



© 2019 Novator Solutions AB

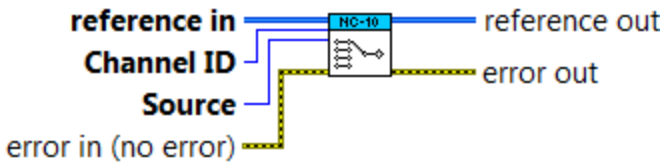
This VI sets the sample rate (IQ-rate) of the specified channel, given in samples per second. This is the only channelization setting that is dependent on the other channels since the aggregate output rate must be within the limits.

The coarse bandwidth, configured in Hz, is a filter in the early stages of the channelization process and allows you to filter undesired signals and noise. The coarse bandwidth must be less than or equal to the sample rate. When applied to the FPGA the bandwidth value will be coerced and the actual value can be read back in the "Commited Channels" list by using Get Channel Lists.vi.

Input Parameters

Channel ID is the unique channel identifier (0…511 or 0…1023).
Sample Rate is the output sample rate of the channel.
Coarse BW is the coarse bandwidth of the channel in Hz.

3.2.18 Set Channel Source.vi

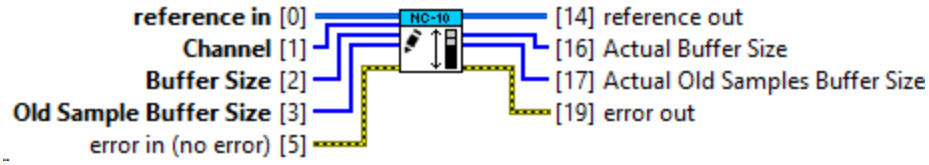


This VI selects the source of wideband signal that channels are extracted from, if the RF Front End that has multiple RF Inputs.

Input Parameters

Channel ID is the unique channel identifier (0…511 or 0…1023).
Source is the RF input source. For the FlexRIO target systems the value should be 0.

3.2.19 Write Buffer Size.vi



The VI sets how many samples at a time that should be output for each channel.

Number of samples=Buffer size*16 or Buffer size*8 for USRP targets

E.g. Buffer size=500 gives 16*500 samples every time that channel is output.

How often the channel buffer is emptied depends on the channel sample rate.

See channel buffering section for more information.

© 2019 Novator Solutions AB

Note: This VI should be used for FlexRIO targets only. Use with USRP will result in channel data not updating correctly.

Input Parameters

Channel is the unique channel identifier (0…511 or 0…1023).

Buffer Size specifies the size of each data buffer that will be output from the channelizer.
Number of samples=Buffer size*16 for FlexRIO and Buffer size*8 for USRP

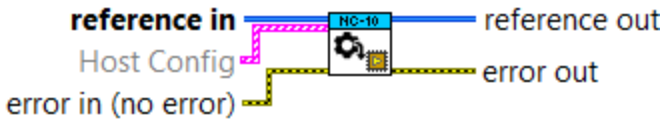
Old Sample Buffer Size specifies the size of each old data buffer that will be output from the channelizer.
Number of samples=Buffer size*16 for FlexRIO and Buffer size*8 for USRP

Output Parameters

Actual Buffer Size buffer size must be a multiple of 16 and if not is will be truncated.

Actual Old Sample Buffer Size buffer size must be a multiple of 16 and if not is will be truncated.

3.2.20 Write Channel Configuration to FPGA.vi



This VI commits the channel configuration changes to the FPGA. Before this VI has been called, no channelization changes have been applied.

When committing, the changes are first written and then read back to verify that the correct channel configuration has been written.

The optional input can be used to apply a channel configuration directly to the FPGA, but without verifying if the configuration is correct and without recalculating internal parameters for the new configuration. This is not recommended practice but can be useful if the programmer is certain of the channel configuration.

Input Parameters

Host Config, an array of clusters with the following elements:

Ch ID is the unique channel identifier (0…511 or 0…1023).

Source is the RF input source. Except for USRP targets, this should be 0.

Enable enables/disables the channel. If TRUE, the channel is enabled, otherwise disabled.

Gain is the fine gain in dB.

Coarse Gain is the coarse gain in dB.

© 2019 Novator Solutions AB

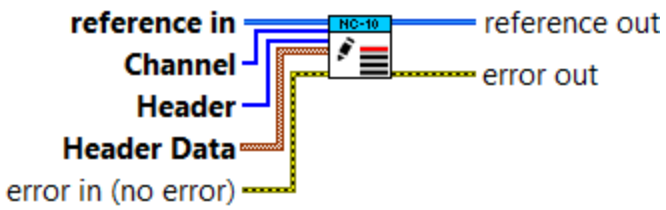
Center Freq is the center frequency of the extracted channel, given in Hz. The valid range is $\pm f_{in}/2$.

Sample Rate is the output sample rate of the channel.

Coarse BW is the coarse bandwidth of the channel.

Coeff Set is the coefficient set to be used for the channel with Channel ID.

3.2.21 Write Header.vi



This VI writes user-configurable headers to the FPGA. The headers are output together with the channel data and can be used for metadata, etc. See Header section for further details. Use Read Header.vi to confirm what actually was written to headers.

Even though this VI is accessible for USRP target as well, it is not recommended to use this as it is used internally within Write Header VITA49.vi with a fixed configuration.

Input Parameters

Channel specifies which channels should be configured. The channel index corresponds to the header and header data index.

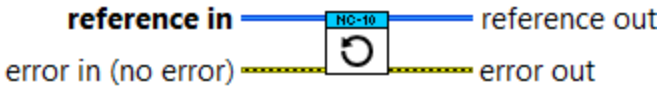
Header specifies which of the available headers should be configured (ID Header, User defined, Footer and Dynamic Header).

Header Data sets the data of the configured header. Header data is a cluster with 16 elements, all I32:

Pos 0 … **Pos 15**

© 2019 Novator Solutions AB

3.2.22 Reset.vi



The VI resets the channelizer operation, meaning that all channels are set to default values (disabled) and all buffers are reset. The reset VI does not reset headers or DRAM.

3.3 Target Specific VIs

It is possible to load NC-10 to an FPGA in a number of target systems. Due to the differences with regards to the target system FPGA size and available system resources, some Channelizer API functions must be replaced with the target specific versions.

The target specific classes inherit from Channelizer API.lvclass, which means that after creating a target specific class it is possible to call methods of the Channelizer API class as well. However, the initialisation of a target specific class must always be done with its specific functions listed in this section.

3.3.1 FlexRIO.lvclass

This class is a child class of Channelizer API.lvclass. It contains mutual VIs used by different NI FlexRIO targets such as NI 7975 and NI 7976. 7975.lvclass and 7976.lvclass are children of the FlexRIO class. The Reference of this class is obtained by calling the "Create" function of the target specific child class, either 7975 Create.vi or 7976 Create.vi

The following common input and output parameters are not described under the individual API function descriptions:

Common Input Parameters

Reference in is the obligatory class reference variable.

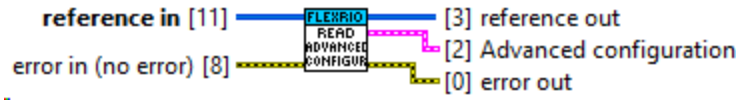
Error in is the standard LabVIEW error cluster.

Common Output Parameters

Reference out is a copy of the “Reference in” parameter.

Error out is the standard LabVIEW error cluster updated with errors raised by the current VI, if any.

3.3.1.1 ReadAdvancedConfiguration.vi



Used to read out the current settings for headers shape and size, old samples and CCF Core Parameters.

© 2019 Novator Solutions AB

Output Parameters

Advanced Configuration

DRAM Active Headers defines which headers should be included in the channel data stream and how the dynamic header should be generated. See Header section for further details.

idHeader is an internal header.

numOfHdrs is the amount of headers including dynamic header to be used, ID and footer are excluded to this number

dynamicHeaderIdx define on which index the dynamic header will be inserted, zeroindexed and must be less than numOfHdrs. -1 to disable dynamic header. Read more in dynamic header.

footer is an internal header.

Dynamic Header Settings, see Dynamic Headers for further details.

Timestamp Index configures the location of the timestamp.

Count Index configures the location of the counter.

Byte Enable configures how the header should be written.

Old Data use these settings to specify how many and if they should be active. One user case scenario are when the signal needs to be filtered.

sendOldData enables or disabled the usage of old samples

nrOldSampleBuffers sets the amount of old samples buffers should be in use, each buffer are 16 samples.

CCF Core Parameters are used to configure the CCF core, depending on bitfile the values might need changes

Number of Filters should be set to 32 or 0 depending on bitfile

Number of Filter Taps if filter exist they should be 31 taps

Limits for the CCF core

Sample Rate Lower

Sample Rate Upper

Fine Gain Lower

Fine Gain Upper

Coarse Gain Lower

Coarse Gain Upper

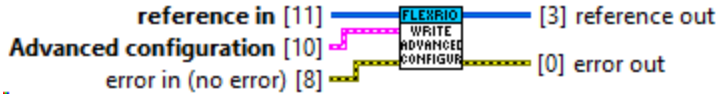
Number of Sources depending on the bitfile this number can be 1 or 4.

Number of Channels depending on bitfile this number can vary from 32-1024

maxIpRate depending on bitfile this value might change

© 2019 Novator Solutions AB

3.3.1.2 WriteAdvancedConfiguration.vi



Can be used to modify the headers shape and size, enable and edit old samples needed for filtering for example. CCF Core Parameters can be left as is if. Novator Solutions will notify upon bitfile delivery if any changes are needed.

Input Parameters

Advanced Configuration

- **DRAM Active Headers** defines which headers should be included in the channel data stream and how the dynamic header should be generated. See Header section for further details.
  - **idHeader** is an internal header.
  - **numOfHdrs** is the amount of headers including dynamic header to be used, ID and footer are excluded to this number
  - **dynamicHeaderIdx** define on which index the dynamic header will be inserted, zeroindexed and must be less than numOfHdrs. -1 to disable dynamic header. Read more in dynamic header.
  - **footer** is an internal header.
- **Dynamic Header Settings**, see Dynamic Headers for further details.
  - **Timestamp Index** configures the location of the timestamp.
  - **Count Index** configures the location of the counter.
  - **Byte Enable** configures how the header should be written.
- **Old Data** use these settings to specify how many and if they should be active. One user case scenario are when the signal needs to be filtered.
  - **sendOldData** enables or disabled the usage of old samples
  - **nrOldSampleBuffers** sets the amount of old samples buffers should be in use, each buffer are 16 samples.
- **CCF Core Parameters** are used to configure the CCF core, depending on bitfile the values might need changes
  - **Number of Filters** should be set to 32 or 0 depending on bitfile
  - **Number of Filter Taps** if filter exist they should be 31 taps
  - **Limits** for the CCF core
  - **Sample Rate Lower**
  - **Sample Rate Upper**
  - **Fine Gain Lower**
  - **Fine Gain Upper**

© 2019 Novator Solutions AB

Coarse Gain Lower

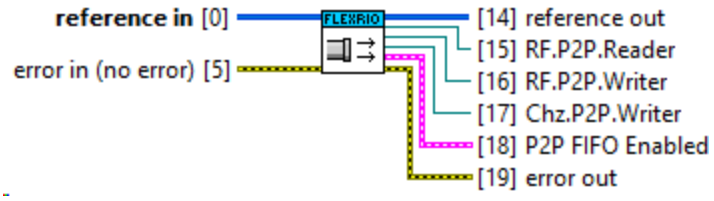
Coarse Gain Upper

Number of Sources depending on the bitfile this number can be 1 or 4.

Number of Channels depending on bitfile this number can vary from 32-1024

maxIpRate depending on bitfile this value might change

3.3.1.3 Get P2P Handles.vi



The peer-to-peer handles are used to connect NC-10 to a signal source and/or a signal sink. This function retrieves the P2P handles that are to be used in the application calling NC-10.

Use the target specific Create function to enable Channel Data P2P FIFO. If some of the P2P FIFOs is not present in FPGA, the P2P references will be invalid.

Output Parameters

RF.P2P.Reader is the handle for the P2P stream that sends data to NC-10. It should be connected to the P2P writer handle of the data source (such as a NI Digitizer or FlexRIO board).

RF.P2P.Writer can be used to forward the incoming data stream to another FPGA board. The data is an identical copy of the data sent to the RF Reader.

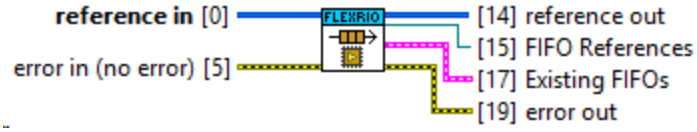
Chz.P2P.Writer is a writer handle for the channel data stream and should be connected to another FPGA board.

P2P FIFO Enabled output shows status of P2P FIFOs read from FPGA

RF.p2p.out.exist shows whether RF P2P FIFO exists on FPGA

chz.p2p.out.exist shows whether Channel Data P2P FIFO exists on FPGA

3.3.1.4 Get FPGA FIFO Reference.vi



Depending on the transfer speed, reading data from the DMA FIFOs can require a more optimized performance and tighter integration into the application. Any wrapping VI around the function call to read the DMA makes the process slower and thus more prone to buffer overflows.

© 2019 Novator Solutions AB

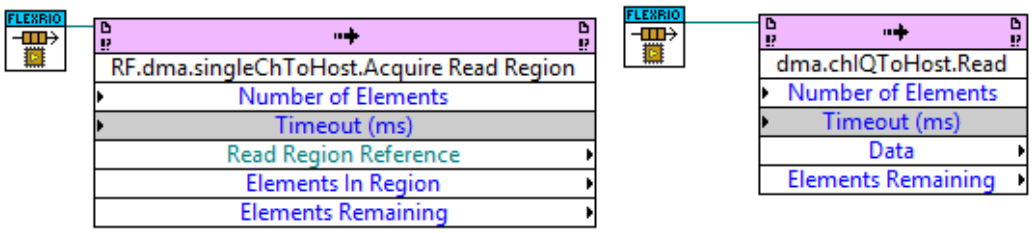
Because of this, in certain cases, using Read RF Data.vi or Read Channel Data.vi could become too slow.

The FPGA FIFO references enable the user to use FPGA Invoke Methods to read from the DMA FIFOs directly.

Note that the RF DMA FIFOs, RF.dma.singleChToHost or RF.dma.multiChToHost still need to be configured to stream continuously or skip samples to create bursts of RF data. This is done in Configure Channelizer FlexRIO.vi.

Design Advice (advanced)

To avoid unnecessary data copies when using the LabVIEW FPGA interface, use **Acquire Read Region** FPGA method instead of the **Read Data** FPGA method.



Output Parameters

FIFO References contains references to FPGA FIFOs that can be used in the application.

Existing FIFOs contains information of the FPGA internal FIFOs

- **DRAM.chIQtoHost.exist** is TRUE if the selected configuration of target system supports DMA transfer of channel data from FPGA to host. For NI 7976 it is always TRUE, and for NI 7975 it depends on selected Channel FIFO Type control value in 7975_Create.vi
- **RF.multiCh.fifo.exist** is TRUE if the selected target system supports multiple RF acquisition devices
- **RF.singleCh.dma.exist** is TRUE if the selected target system supports single RF acquisition devices.

3.3.1.5 Configure Timestamp.vi

This VI configures start time (t0) and triggering for resetting the internal timer. Initial time t0 is only updated if Update t0 is set to true. Reset can either be manually performed from the host by setting Reset Counter to true when calling this VI, or sent via the PXI backplane with one of the PXI triggers.

See Timestamp for more details.

Input Parameters

© 2019 Novator Solutions AB

Timestamp contains a cluster that configures the timestamp settings. Timestamp PXI trigger configures which trigger should be used to reset the counter.

- **t0** start time in ns.
- **Update t0** if TRUE, the initial time t0 is updated.
- **Reset Counter** resets the internal counter.
- **Timestamp PXI Trigger** defines which (if any) PXI trigger line should be used to reset the counter.

3.3.1.6 Check Status.vi

This VI retrieves status information about the NC-10 Channelizer and resets counters and timeout indicators.

See Channel Settings for information about the channel configuration overflow status.

The counters are useful to check if data is flowing in to and out from the NC-10 FPGA.
The buffer overflow indicator indicates whether the internal buffers have overflown and data have been lost, which can happen if the channel data is not read fast enough.

Input Parameters

Reset Status clears the status information.

- **Reset Data IO** contains controls to reset counters and timeouts.
  - **Reset RF In Count** resets the counter RF In Count.
  - **Reset Channel Out Count** resets the counter Channel Out Count.
  - **Reset RF Timeout** resets the RF DMA Timeout and RF P2P Timeout indicators.
- **Reset Buffer Overflow** resets the channel buffer overflow.

Output Parameters

Channelizer Status contains information about the current status of NC-10.

- **Data IO Status** contains information about data in to and out from the FPGA.
  - **RF In Count** is the number of input samples. During operation this should always count up, otherwise the channelizer is not receiving RF data.
  - **Channel Out Count** is the number of channel output samples (including headers). When any channel is enabled, this should count up.
  - **RF DMA Timeout** indicates a timeout on the FPGA when writing the RF data to the host buffer. RF data have been lost if this indicator is

© 2019 Novator Solutions AB

True.

RF P2P Timeout indicates a timeout on the FPGA when writing the RF data to the Peer to peer buffer. RF data have been lost if this is indicator is True.

Channel Cfg Overflow contains information about channel configuration overflow.

Overflow Channel indicates the first channel index that was discarded. All higher channel indices will also be discarded.

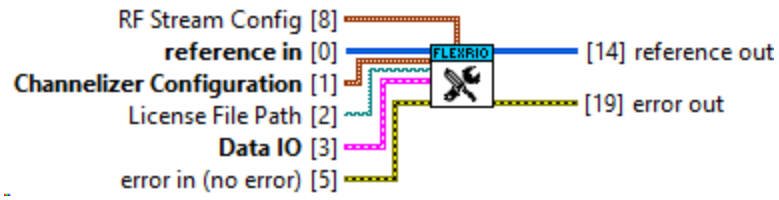
Overflow True indicates if a channel configuration overflow has occurred.

Channel Buffering Overflow contains information on the FPGA internal DRAM status

FIFO Overflow is True if one of the FPGA internal data transfer FIFOs has overflown causing data loss.

Channel Buffer Overflow is a vector of Booleans showing internal buffer overflows for each of the configured channel. If any of these are true, data in that channel buffer have been overwritten with new data and channel data have been lost.

3.3.1.7 Configure Channelizer FlexRIO.vi



This VI must be called to configure important parameters on the FPGA and should be called before Open FPGA.vi.

This VI is dynamically dispatched to the child class version, which means in practice that for example, if the **reference in** wire has been created with a call to 7975_Create.vi , LabVIEW will automatically replace a call to Configure Channelizer FlexRIO.vi with a 7975 specific version of Configure Channelizer FlexRIO.vi.

Some of the Data IO input can also be changed during runtime to control where the data should be sent.

Common Input Parameters

Channelizer Configuration

f_in is the data input rate. See RF Input about details regarding data rate. The default value is 62.5 MS/s

License File Path if unwired or empty, uses default path %public% \documents\Novator Solutions\NC-10\Licenses\License.txt. If the file does not exist or no license key is found on this file, the execution will be timebombed.

Data IO configures of the Data IO modes.

rfInput can be either Real or Complex and defines the input data format. When Real, the quadrature part is set to 0. See RF Input for further details.

rfOutput configures if the RF input should be fed through to an RF output. See RF Output for further details.

chOutput control is not in use and is placed here for future releases. To determine destination of channel data refer to help of the target specific implementation of

© 2019 Novator Solutions AB

this VI.

rfScaleOffset see RF Input for further details.

Scale factor configures how the input sample should be scaled. Default is 1.

Offset configures how the input sample should be offset. Default is 0.

chOutputHostBufferSize is the size of the host buffer of the Channel DMA FIFO. If this is set to 0, the LabVIEW default value will be used.

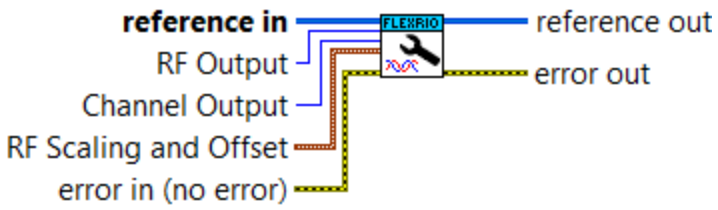
RF Stream Config sets the parameters for a DMA transfer of the RF data from the channelizer instrument to the host. The parameter **RF Output** in the **Data IO** cluster (see above) has to be set to "Host" for this parameter to have effect. NOTE: the values are applied in the FPGA code when Open FPGA.vi is called.

Host Buffer Size sets the amount of data in bytes in the host computer RAM to allocate for the RF streaming data. The value is limited by the available RAM memory of the host computer. This parameter needs to be greater or equal to **Number of Samples** parameter in Read RF Data.vi (if Number of Samples is greater than zero).

Number of Samples To Read sets the number of consecutive samples to pass from an RF source to the Channelizer.

Number of Samples To Skip sets the number of samples to omit after **Number of Samples to Read** has been reached. If Number of Samples to Skip =0, all samples are passed from the RF source to the RF DMA FIFO.

3.3.1.8 Configure IO FlexRIO.vi



The VI gives the user the option to reconfigure the output of NC-10. The settings of this VI are also configured in Configure Channelizer FlexRIO.vi and should only be called if a setting needs to be changed in runtime, for example turning RF-output streaming on or off.

If the Channel Output is changed it is important to disable all channels and let the buffers empty before changing. Otherwise header information could be lost and sent to one destination while the data is sent to another.

Read sections RF Input, RF Output and Channel Output for more information about how the IO configuration affects the application.

Input Parameters

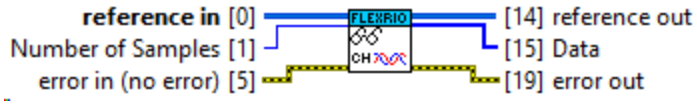
RF Output is a feedthrough of the RF input. See RF Output for further details.
Channel Output control is not in use and is placed here for future releases. To determine destination of channel data in NI 7975 use Channel FIFO Type input of 7975_Create.vi. With NI 7976, both DMA and P2P stream are supported at the same time.

© 2019 Novator Solutions AB

RF Scaling and Offset see RF Input for further details.

- **Scale factor** configures how the input sample should be scaled. Default is 1.
- **Offset** configures how the input sample should be offset. Default is 0.

3.3.1.9 Read Channel Data.vi



This VI reads channel IQ data from a DMA FIFO in the FPGA.

It is important to read the DMA FIFO often enough to avoid buffer overflows. The higher the aggregate output sample rate, the more often the DMA FIFO must be read. Depending on the CPU load and other parameters, using Read Channel Data might be too slow, thereby causing the buffers to overflow. If more optimized performance is required, it is possible to use a reference to the DMA FIFO transferring the channel data to the host. This can be done by calling Get FPGA FIFO Reference.vi

To see if a buffer overflow condition have occurred, use Check Status.vi.

Input Parameters

Number of Samples sets how many samples that should be read, -1 reads all samples.

Output Parameters

Data contains the IQ channel data stream read from the FPGA. I32 data type represents concatenated I and Q data and in order to get their values, **Data** should be split into two 16-bits numbers. Most significant 16 bits correspond to Q and least significant to I data. Since splitting function gives U16 data type, I and Q should finally be converted to I16.

3.3.1.10 Enable RF Stream FlexRIO.vi

Call this VI to enable an RF stream from the FPGA interface module to host via DMA once at the program configuration steps, after the FPGA has been initiated with Open FPGA.vi. This VI clears the DMA buffer prior to enabling the DMA stream to host. This VI is not required to start streaming RF data to the channelizer, unless the incoming RF data should be read on the host.

3.3.1.11 Disable RF Stream FlexRIO.vi

© 2019 Novator Solutions AB

Call this VI to stop an RF stream started by Enable RF Stream FlexRIO.vi prior to ending your application.

3.3.2 7975.lvclass VIs

This class is a child class of FlexRIO.lvclass. Call 7975_Create.vi to create the NC-10 instance, if NI PXIe-7975 is the target system in use.

The following common input and output parameters are not described under the individual API function descriptions:

Common Input Parameters

Reference in is the obligatory class reference variable. The value to this reference is created by calling the function **7975_Create.vi**.

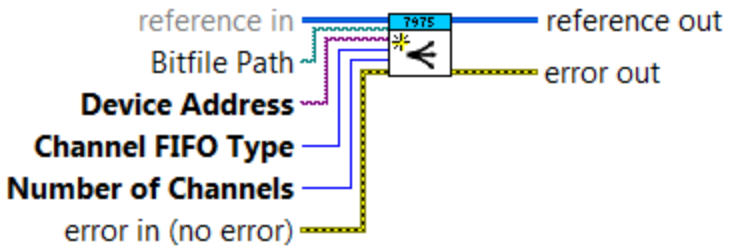
Error in is the standard LabVIEW error cluster.

Common Output Parameters

Reference out is a copy of the "Reference in" parameter.

Error out is the standard LabVIEW error cluster updated with errors raised by the current VI, if any.

3.3.2.1 7975_Create.vi



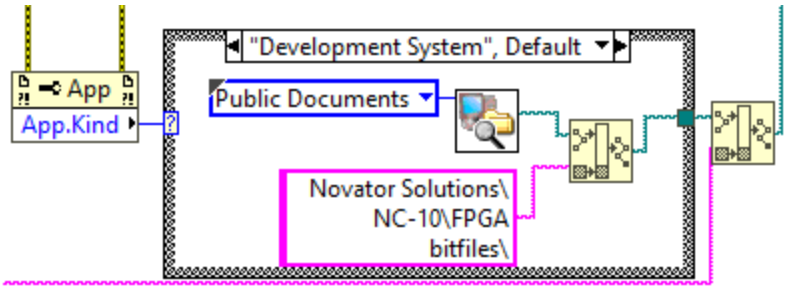
Call this VI to initialize a Channelizer session.

The value of the Bitfile Path input controls some of the functionality on how NC-10 will run. The different alternatives are empty, path to a TDMS file, or path to a bitfile.

Non empty path to bitfile: The API will look for the bitfile at the specified path. Note that Channel FIFO Type and Number of Channels input values have to correspond to the build configuration of the bitfile for the core to function properly.
Empty path: The API will look for the bitfile in the default locations. However, if the NC-10 package is downloaded from LabVIEW Tools Network, bitfiles will not be found in default locations and user should contact Novator Solutions. Once a bitfile is obtained, it can be placed in a location where the API expects it as described below.

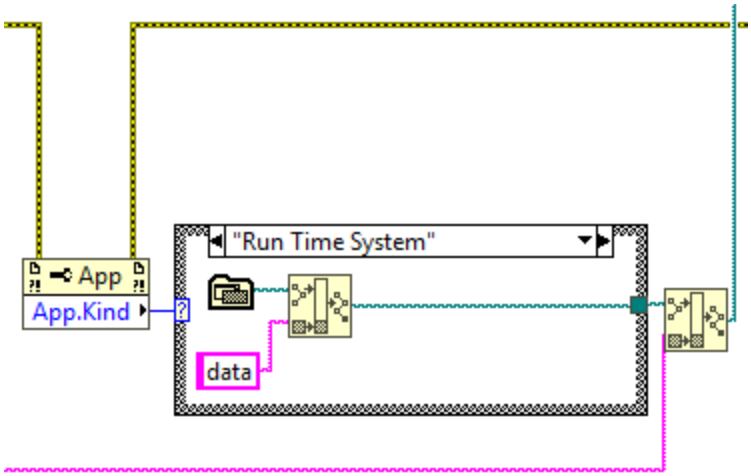
When running in LabVIEW, the API will look for the bitfile in %public%\documents\Novator Solutions\NC-10\FPGA bitfiles\ folder. If it is not found in this location, NC-10 will run the timebombed version.

 © 2019 Novator Solutions AB



NC-10 Bitfile Location in Development System

In Run Time Systems the API expects the bitfile in the data folder (application directory/data folder). This is important when building an executable.



NC-10 Bitfile Location in a Built Executable.

The default bitfile name depends on the selected Channel FIFO Type and Number of Channels:

Nr Channels	FIFO	Bitfile to use
512	P2P or DMA	*NC-10_CCF_1_512 (7975).lvbitx*
1024	P2P	*NC-10_CCF_1_1024 (7975) P2P.lvbitx*
1024	DMA	*NC-10_CCF_1_1024 (7975) DMA.lvbitx*

TDMS file path: The API is run in simulated mode and the TDMS file is used as data source.

Building executable
This VI dynamically calls three classes that needs to be added to build specifications if built into an executable.

DRAM IO_Class
CCF IO_Class
SrcIO_Class

Input Parameters

 © 2019 Novator Solutions AB

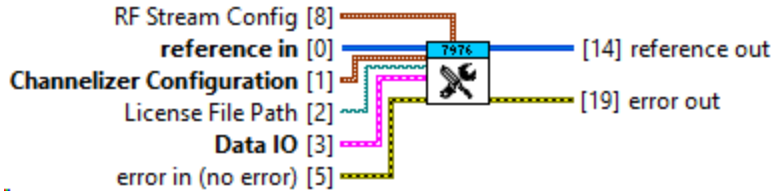
Bitfile Path is the path to either bitfile or TDMS-file for simulation. use empty path for default location.

Device Address is the identifier of the FlexRIO device you want to use defined in MAX.

Channel FIFO Type determines whether channel IQ data is output to DMA or P2P FIFO. This setting matters only if Number of Channels is 1024. When Number of Channels is 512, both DMA and P2P FIFOs are available, regardless of the selection. This limitiation with 1024 channels is due to resource utilization on the FPGA.

Number of Channels can be either 512 or 1024.

3.3.2.2 Configure Channelizer FlexRIO.vi



This VI must be called to configure important parameters on the FPGA and should be called before Open FPGA.vi.

This VI will be called when Configure Channelizer FlexRIO.vi is called with a reference obtained using 7975 Create.vi

Some of the Data IO input can also be changed during runtime to control where data should be sent.

Common Input Parameters

Channelizer Configuration

f_in is the data input rate. See RF Input about details regarding data rate. The default value is 62.5 MS/s

License File Path if unwired or empty, uses default path %public% \documents\Novator Solutions\NC-10\Licenses\License.txt. If the file does not exist or no license key is found on this file, the execution will be timebombed.

Data IO configures of the Data IO modes.

rfInput can be either Real or Complex and defines the input data format. When Real, the quadrature part is set to 0. See RF Input for further details.

rfOutput configures if the RF input should be fed through to an RF output. See RF Output for further details.

chOutput control is not in use and is placed here for future releases. To determine destination of channel data refer to help of the target specific implementation of this VI.

rfScaleOffset see RF Input for further details.

© 2019 Novator Solutions AB

Scale factor configures how the input sample should be scaled. Default is 1.

Offset configures how the input sample should be offset. Default is 0.

chOutputHostBufferSize is the size of the host buffer of the Channel DMA FIFO. If this is set to 0, the LabVIEW default value will be used.

RF Stream Config sets the parameters for a DMA transfer of the RF data from the channelizer instrument to the host. The parameter **RF Output** in the **Data IO** cluster (see above) has to be set to "Host" for this parameter to have effect. NOTE: the values are applied in the FPGA code when Open FPGA.vi is called.

Host Buffer Size sets the amount of data in bytes in the host computer RAM to allocate for the RF streaming data. The value is limited by the available RAM memory of the host computer. This parameter needs to be greater or equal to **Number of Samples** parameter in Read RF Data.vi (if Number of Samples is greater than zero).

Number of Samples To Read sets the number of consecutive samples to pass from an RF source to the Channelizer.

Number of Samples To Skip sets the number of samples to omit after **Number of Samples to Read** has been reached. If Number of Samples to Skip =0, all samples are passed from the RF source to the RF DMA FIFO.

3.3.2.3 Destroy.vi

Call this VI to close the NC-10 session.

3.3.3 7976.lvclass VIs

This class is a child class of FlexRIO.lvclass. Call 7976_Create.vi to create the NC-10 instance, if NI PXIe-7976 is the target system in use.

The following common input and output parameters are not described under the individual API function descriptions:

Common Input Parameters

Reference in is the obligatory class reference variable. The value to this reference is created by calling the function **7976_Create.vi**.

Error in is the standard LabVIEW error cluster.

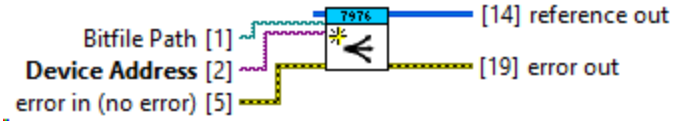
Common Output Parameters

Reference out is a copy of the “Reference in” parameter.

Error out is the standard LabVIEW error cluster updated with errors raised by the current VI, if any.

© 2019 Novator Solutions AB

3.3.3.1 7976_Create.vi

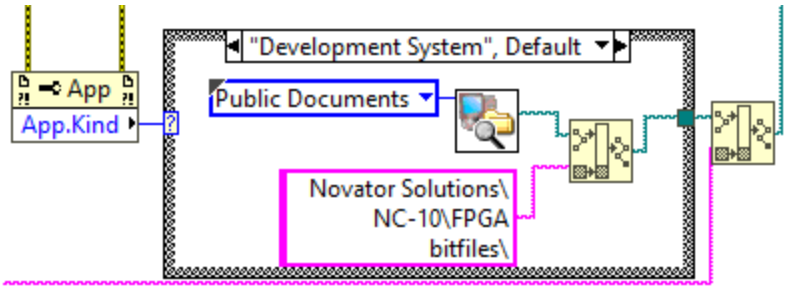


Call this VI to initialize a Channelizer session.

The value of the Bitfile Path input controls some of the functionality on how NC-10 will run. The different alternatives are empty, path to a TDMS file, or path to a bitfile.

Non empty path to bitfile: The API will look for the bitfile at the specified path.
Empty path: The API will look for the bitfile in the default locations. However, if the NC-10 package is downloaded from LabVIEW Tools Network, bitfiles will not be found in default locations and user should contact Novator Solutions. Once a bitfile is obtained, it can be placed in a location where the API expects it as described below.

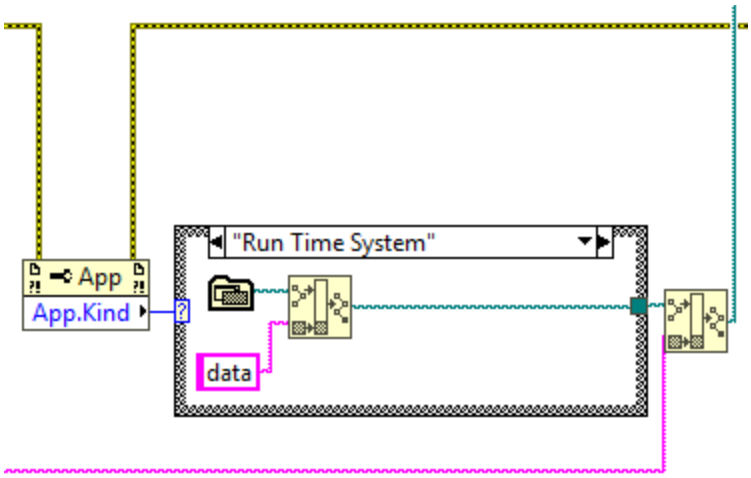
When running in LabVIEW, the API will look for the bitfile in %public%\documents\Novator Solutions\NC-10\FPGA bitfiles\ folder. If it is not found in this location, NC-10 will run the timebombed version.



NC-10 Bitfile Location in Development System

In Run Time Systems the API expects the bitfile in the data-folder (application directory/data folder). This is important when building an executable.

© 2019 Novator Solutions AB



NC-10 Bitfile Location in a Built Executable.

The default bitfile name is *NC-10_CCF_1_512 (7976).lvbitx*.

TDMS file path: The API is run in simulated mode and the TDMS file is used as data source.

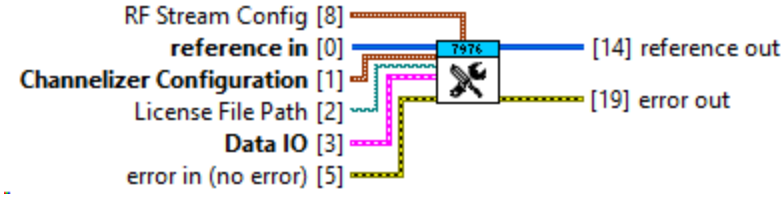
Building executable
This VI dynamically calls three classes that needs to be added to build specifications if built into an executable.

DRAM IO_Class
CCF IO_Class
SrcIO_Class

Input Parameters

Bitfile Path is the path to either bitfile or TDMS-file for simulation. Use empty path for default location.
Device Address is the name of the FlexRIO device you want to use.

3.3.3.2 Configure Channelizer FlexRIO.vi



This VI must be called to configure important parameters on the FPGA and should be called before Open FPGA.vi.

This VI will be called when Configure Channelizer FlexRIO.vi is called with a reference obtained using 7976 Create.vi

Some of the Data IO input can also be changed during runtime to control where data should be sent.

 © 2019 Novator Solutions AB

Common Input Parameters

Channelizer Configuration

f_in is the data input rate. See RF Input about details regarding data rate. The default value is 62.5 MS/s

License File Path if unwired or empty, uses default path %public% \documents\Novator Solutions\NC-10\Licenses\License.txt. If the file does not exist or no license key is found on this file, the execution will be timebombed.

Data IO configures of the Data IO modes.

rfInput can be either Real or Complex and defines the input data format. When Real, the quadrature part is set to 0. See RF Input for further details.

rfOutput configures if the RF input should be fed through to an RF output. See RF Output for further details.

chOutput control is not in use and is placed here for future releases. To determine destination of channel data refer to help of the target specific implementation of this VI.

rfScaleOffset see RF Input for further details.

Scale factor configures how the input sample should be scaled. Default is 1.

Offset configures how the input sample should be offset. Default is 0.

chOutputHostBufferSize is the size of the host buffer of the Channel DMA FIFO. If this is set to 0, the LabVIEW default value will be used.

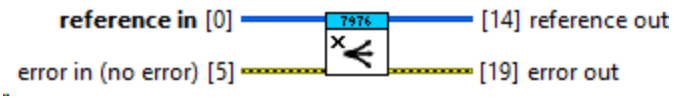
RF Stream Config sets the parameters for a DMA transfer of the RF data from the channelizer instrument to the host. The parameter **RF Output** in the **Data IO** cluster (see above) has to be set to "Host" for this parameter to have effect. NOTE: the values are applied in the FPGA code when Open FPGA.vi is called.

Host Buffer Size sets the amount of data in bytes in the host computer RAM to allocate for the RF streaming data. The value is limited by the available RAM memory of the host computer. This parameter needs to be greater or equal to **Number of Samples** parameter in Read RF Data.vi (if Number of Samples is greater than zero).

Number of Samples To Read sets the number of consecutive samples to pass from an RF source to the Channelizer.

Number of Samples To Skip sets the number of samples to omit after **Number of Samples to Read** has been reached. If Number of Samples to Skip =0, all samples are passed from the RF source to the RF DMA FIFO.

3.3.3.3 Destroy.vi



Call this VI to close the NC-10 session.

© 2019 Novator Solutions AB

3.3.4 USRP-2955.lvclass VIs

Call USRP-2955_Create.vi to create the NC-10 instance. This class contains all target specific methods for NC-10.

The following common input and output parameters are not described under the individual API function descriptions:

Common Input Parameters

Reference in is the obligatory class reference variable. The value to this reference is created by calling the function **USRP-2955_Create.vi**.

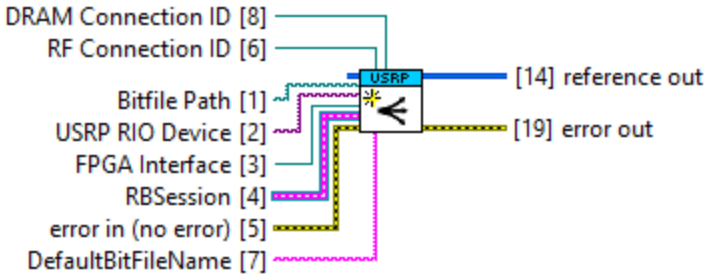
Error in is the standard LabVIEW error cluster.

Common Output Parameters

Reference out is a copy of the "Reference in" parameter.

Error out is the standard LabVIEW error cluster updated with errors raised by the current VI, if any.

3.3.4.1 USRP-2955_Create.vi



Call this VI to initialize a Channelizer (NC-10) session. Depending on the use case some inputs can be unused as described below.

Building executable
This VI dynamically calls three classes that needs to be added to build specifications if built into an executable.

DRAM IO RB_Class
CCF IO RB_Class
SrcIO RB_Class

Input Parameters

Connection over UDP and NI Register Bus

DRAM Connection ID	Connection needed to read out data from DRAM, this is a UDP unicast connection.
RF Connection ID	Connection where to send RF stream data, this can be both multicast or unicast.

© 2019 Novator Solutions AB

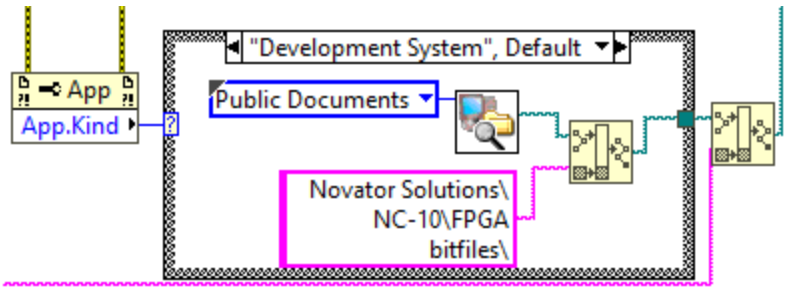
Bitfile Path	Not used in this mode.
USRP RIO Device	Not used in this mode.
FPGA Interface	Not used in this mode.
RBSession	Session to NI Register bus opened with Open session.vi from NI register bus v0 library.
DefaultBitFileName	Must be an empty string input, otherwise MXI connection is assumed.

Connection over MXI-Link

DRAM Connection ID	Not used in this mode.
RF Connection ID	Not used in this mode.
Bitfile Path	Can be specified but not needed if you placed the correct bitfile in the correct location described in paragraph **Empty Path.**
USRP RIO Device	Required to know which USRP to connect to. The value is set in MAX.
FPGA Interface	Only needed if you already opened and downloaded a session to a NC-10 bitfile.
RBSession	Not used in this mode.
DefaultBitFileName	Leave unwired for the default bitfile name. Not used if bitfile path is wired.

Non-empty path to bitfile: The API will look for the bitfile at the specified path.
Empty path: The API will look for the bitfile in the default locations. However, if the NC-10 package is downloaded from LabVIEW Tools Network, bitfiles will not be found in default locations and user should contact Novator Solutions. Once a bitfile is obtained, it can be placed in a location where the API expects it as described below.

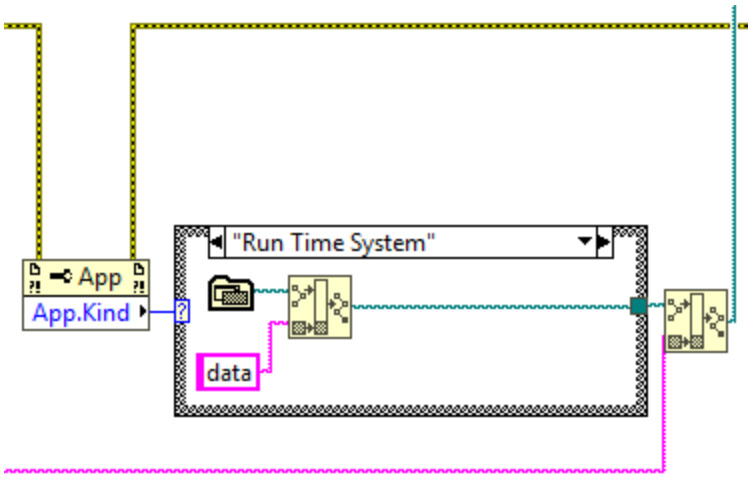
When running in LabVIEW, the API will look for the bitfile in %public%\documents\Novator Solutions\NC-10\FPGA bitfiles\ folder. If it is not found in this location, NC-10 will run the timebombed version.



NC-10 Bitfile Location in Development System

In Run Time Systems the API expects the bitfile in the data-folder (application directory/data folder). This is important when building an executable.

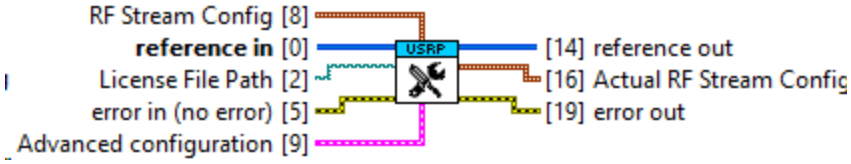
© 2019 Novator Solutions AB



NC-10 Bitfile Location in a Built Executable.

The default bitfile name for USRP connected over MXI-link is *NC-10_USRP_2955_4RF_512Ch.lvbitx*.

3.3.4.2 Configure Channelizer 2955.vi



This VI must be called to configure important parameters on the FPGA and should be called before Open FPGA.vi.

RF Stream Config will setup the number of samples to read in each burst from each tuner. Figure 1 below shows how the bursting mode works in more detail, R = samples to read and S = samples to skip. If only one tuner is enabled every sequence would burst data from that tuner. If multiple tuners are enabled (Figure 2) data from all tuners would be aggregated and interleaved.

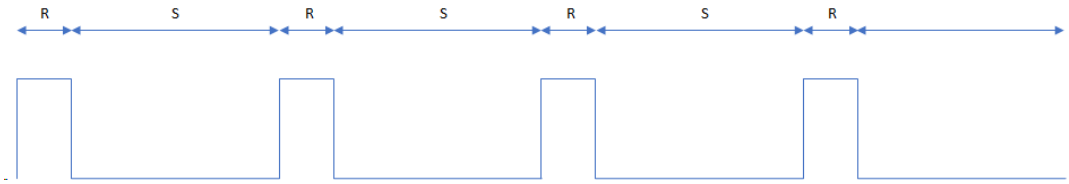


Figure 1: RF Stream, 1 tuner.

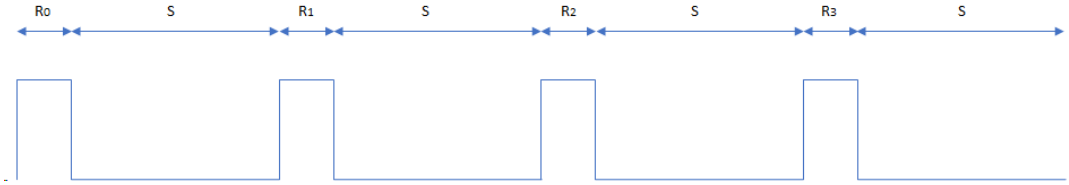


Figure 2: RF Stream, 4 tuners.

© 2019 Novator Solutions AB

Input Parameters

RF Stream Config is used for read and skip size settings.

- **Host Buffer Size** not used when UDP mode is selected, for MXI-link this will be the buffer size in samples on host.
- **Number of Samples To Acquire** the number of samples produced in each burst.
- **Number of Samples To Skip** the number of samples to skip between each read.

License File Path if unwired or empty, uses default path %public% \documents\Novator Solutions\NC-10\Licenses\License.txt. If the file does not exist or no license key is found on this file, the execution will be timebombed.

Advanced Configuration will not be wired for VITA49 usecase. Custom bitfiles might need changes in this input.

- **DRAM Active Headers** defines which headers should be included in the channel data stream and how the dynamic header should be generated. See Header section for further details.
  - **idHeader** is an internal header.
  - **numOfHdrs** is the amount of headers including dynamic header to be used, ID and footer are excluded to this number
  - **dynamicHeaderIdx** define on which index the dynamic header will be inserted, zeroindexed and must be less than numOfHdrs. -1 to disable dynamic header. Read more in dynamic header.
  - **footer** is an internal header.
- **Dynamic Header Settings**, see Dynamic Headers for further details.
  - **Timestamp Index** configures the location of the timestamp.
  - **Count Index** configures the location of the counter.
  - **Byte Enable** configures how the header should be written.
- **Old Data** use these settings to specify how many and if they should be active. One user case scenario are when the signal needs to be filtered.
  - **sendOldData** enables or disabled the usage of old samples
  - **nrOldSampleBuffers** sets the amount of old samples buffers should be in use, each buffer are 16 samples.
- **CCF Core Parameters** are used to configure the CCF core, depending on bitfile the values might need changes
  - **Number of Filters** should be set to 32 or 0 depending on bitfile
  - **Number of Filter Taps** if filter exist they should be 31 taps
  - **Limits** for the CCF core
  - **Sample Rate Lower**
  - **Sample Rate Upper**

© 2019 Novator Solutions AB

Fine Gain Lower

Fine Gain Upper

Coarse Gain Lower

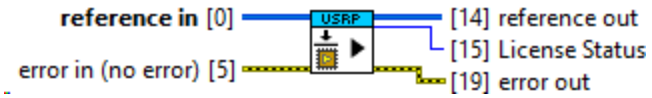
Coarse Gain Upper

Number of Sources depending on the bitfile this number can be 1 or 4.

Number of Channels depending on bitfile this number can vary from 32-1024

maxIpRate depending on bitfile this value might change

3.3.4.3 Open FPGA.vi



This VI writes all initialization steps to FPGA of USRP-2955.

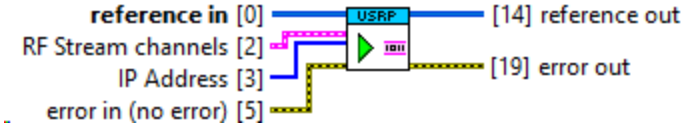
Output Parameters

License Status displays the current license status (Valid License, Valid Device Specific, Timebomb Running, Core Locked).

3.3.4.4 Destroy.vi

Closes the session to USRP and makes sure to stop any ongoing channel or RF streams.

3.3.4.5 Enable RF Stream 2955.vi



This VI turns on the stream of RF data for all enabled tuners. IP Address is used to determine the destination for stream data and it can be either unicast or Multicast address.

Input Parameters

RF Stream channels determines which tuners should be enabled.

Channel 0 tuner 0

Channel 1 tuner 1

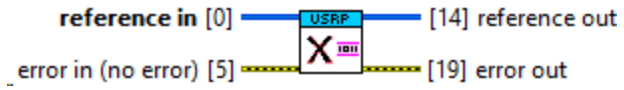
© 2019 Novator Solutions AB

Channel 2 tuner 2

Channel 3 tuner 3

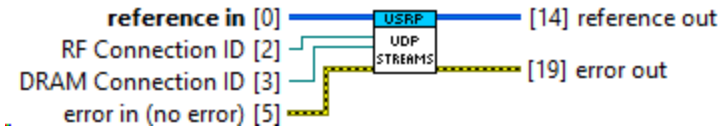
IP Address of destination can be both unicast or multicast.

3.3.4.6 Disable RF Stream 2955.vi



This VI disables any ongoing RF stream and clears any remaining data in the UDP buffer. This data will still be sent out but will have the valid flag set to False.

3.3.4.7 Set UDP Connection Streams.vi



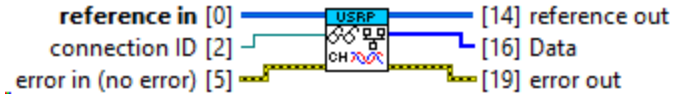
This VI updates the RF Connection ID and the DRAM Connection ID. Only valid connection IDs will be updated.

Input Parameters

RF Connection ID UDP unicast or multicast connection to read RF stream from.

DRAM Connection ID UDP unicast connection to read DRAM data from.

3.3.4.8 Read Channel Data UDP.vi



Reads channel data from UDP connection wired to connection ID. Data obtained is packed with the VITA49 standard.

Input Parameters

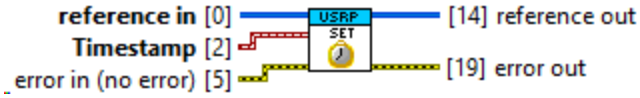
connection ID UDP connection for channelized data, either multicast or unicast.

Output Parameters

Data VITA49 packed channel data.

© 2019 Novator Solutions AB

3.3.4.9 Set Start Time

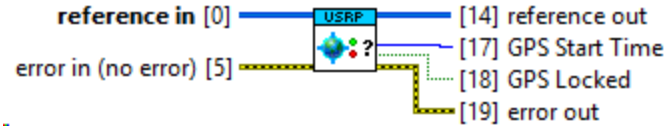


This VI sets the start time in FPGA and resets timestamp related seconds and picoseconds counters in FPGA. *If this VI is used to write Start Time, GPS NMEA time will not be used.*

Input Parameters

Timestamp Default timestamp in LabVIEW

3.3.4.10 GPS Status.vi



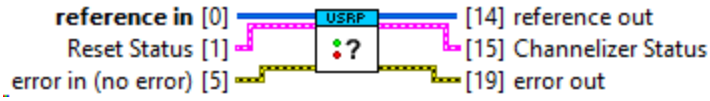
GPS Start time can be the time since GPS first locked or it can be the host written time with Set Start Time.vi. Time unit is seconds. GPS Locked lets you know if the USRP has been able to lock itself to a GPS signal.

Input Parameters

GPS Start Time Time in seconds from the moment the unit locked itself to a GPS, or Start Time set from host in seconds.

GPS Locked Indicates if USRP-2955 is locked to GPS signal.

3.3.4.11 Check Status



This VI retrieves status information about the NC-10 Channelizer and resets the counters and timeout indicators.

See Channel Settings for information about the channel configuration overflow status.

The counters are useful in verifying that data is flowing into the FPGA and out from it. Buffer Overflow indicates whether the internal buffers have overflown and data has been lost, which can happen if the channel data is not read fast enough.

Input Parameters

Reset Status clears the status information.

Reset Data IO contains controls to reset counters and timeouts.

© 2019 Novator Solutions AB

Reset RF In Count resets the counter RF In Count.

Reset Channel Out Count resets the counter Channel Out Count.

Reset RF Timeout resets the RF Output Timeout and RF to CCF Timeout.

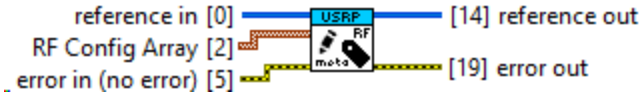
Reset Buffer Overflow resets the channel buffer overflow.

Output Parameters

Channelizer Status contains information about the current status of NC-10.

- **Data IO Status** contains information about data into the FPGA and out from it.
  - **RF In Count** is the number of input samples. During operation this should always count up, otherwise the channelizer is not receiving RF data.
  - **Channel Out Count** is the number of channel output samples (including headers). When any channel is enabled, this should count up.
  - **RF Output Timeout** indicates a timeout on the FPGA when writing RF data to the UDP buffer.
  - **RF to CCF Timeout** indicates a timeout on the FPGA when writing RF data to the channelizer core.
- **Channel Buffering Overflow** contains information on the FPGA internal DRAM status
  - **FIFO Overflow** is True if one of the FPGA internal data transfer FIFOs has overflown causing data loss.
  - **Channel Buffer Overflow** is a vector of Booleans showing internal buffer overflows for each of the configured channel. If any of these are true, data in that channel buffer has been overwritten with new data and channel data has been lost.

3.3.4.12 Write RF Metadata.vi



Use this VI to write all meta data for context and data headers for RF streaming.

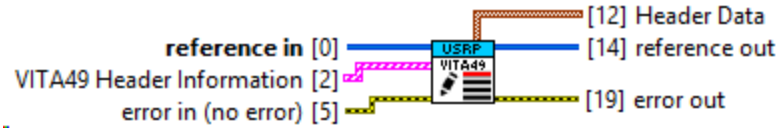
Input Parameters

RF Config Array configuration for each tuner of USRP-2955

- **frequency** center frequency of the specific tuner.
- **gain** applied to specific tuner.
- **sample rate** of the specific tuner. This must always be 100M.

© 2019 Novator Solutions AB

3.3.4.13 Write Header VITA49.vi



This VI writes the header data for each channel, formatted in accordance with VITA49 standard.Use Read Header VITA49.vi together with Header Data out to verify what was written with this VI.

Input Parameters

VITA49 Header Information Array of header information for all channels

VITA49 Header Information header data for each channel

- **Remote MAC**: array of 6 elements for MAC address. If all fields are zeroes the USRP-2955 will ARP for the remote MAC
- **Remote IP**: array of 4 elements for IP address, can be both multicast and unicast
- **remote port** used for the current channel
- **Stream ID** for the current channel, for ease of use set this to the channel number
- **Timestamp Integer**: seconds since start time
- **Timestamp Fractional**: fractional seconds for current second
- **Bandwidth** of the current channel
- **RF Reference Frequency** is the tuners center frequency
- **RF Reference Frequency Offset** is the channels center frequency offset in relation to the tuner center frequency
- **Reference Level**: not used for USRP-2955 as it is not a calibrated instrument
- **Gain**: level of gain to be used for the current channel
- **Sample Rate** of the current channel

Output Parameters

Header Data Array of raw header data

Header data partial header segment of 16 elements of I32 data

- **Pos 0**
- **Pos 1**
- **Pos 2**
- **Pos 3**
- **Pos 4**

© 2019 Novator Solutions AB

Pos 5

Pos 6

Pos 7

Pos 8

Pos 9

Pos 10

Pos 11

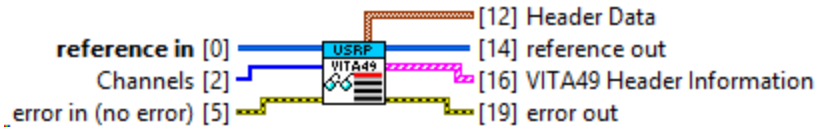
Pos 12

Pos 13

Pos 14

Pos 15

3.3.4.14 Read Header VITA49.vi



Reads out the header data, both raw and VITA49 format for the channels specified in Channels input. Use Header Data output to compare what was written to confirm the write operation worked as expected.

Input Parameters

Channels specify which channel number to read out the headers for

Output Parameters

Header Data Array of raw header data

Header data partial header segment of 16 elements of I32 data

Pos 0

Pos 1

Pos 2

Pos 3

Pos 4

Pos 5

Pos 6

Pos 7

Pos 8

Pos 9

© 2019 Novator Solutions AB

Pos 10

Pos 11

Pos 12

Pos 13

Pos 14

Pos 15

VITA49 Header Information Array of header information for all channels

VITA49 Header Information header data for each channel

Remote MAC: array of 6 elements for MAC address. If all fields are zeroes the USRP-2955 will ARP for the remote MAC

Remote IP: array of 4 elements for IP address, can be both multicast and unicast

remote port: used for the current channel

Stream ID for the current channel

Timestamp Integer: seconds since start time

Timestamp Fractional: fractional seconds for current second

Bandwidth of the current channel

RF Reference Frequency is the tuners center frequency

RF Reference Frequency Offset is the channels center frequency offset in relation to the tuner center frequency

Reference Level: not used for USRP-2955 as it is not a calibrated instrument

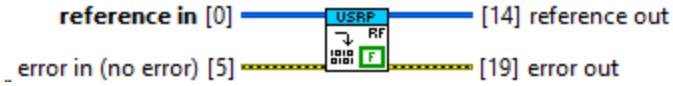
Gain: level of gain to be used for the current channel

Sample Rate of the current channel

3.3.4.15 Set GPS Refresh Flag.vi

This VI sets the GPS refresh flag to True, which will make the FPGA register the GPS time and store it as a new start time if the unit has locked itself to a GPS satellite. Use GPS Status.vi to check that Start Time got updated. Call to this VI will also restart the timestamp related seconds and picoseconds counters.

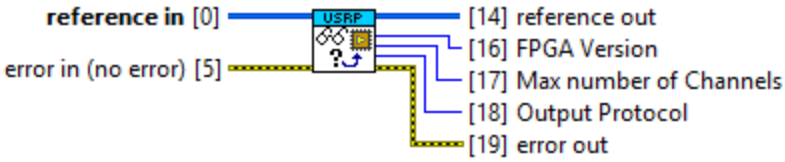
3.3.4.16 Set Data Invalid for RF Reconfig.vi



© 2019 Novator Solutions AB

Sets the Invalid flag to true so that we can indicate that RF packets sent during reconfiguration do not have valid header data. When using Write RF Metadata.vi this flag is reset back to false and data is again valid.

3.3.4.17 Get FPGA Information.vi



Returns information about the bitfile loaded to the USRP-2955.

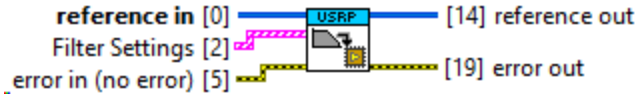
Output Parameters

FPGA Version is a number that helps identify what bitfile is running currently on the target.

Max number of Channels are the maximum amount of channelizer channels available in the loaded bitfile.

Output Protocol indicates what protocol the specific bitfile is using. Either NCR or VITA49. If N/A is returned there was an error reading the protocol.

3.3.4.18 Write IQ Filter Coefficients.vi



Write all filter coefficients and filter states for each channel to FPGA. Make sure the FilterCoefficients array are size 55 for each channel to update, if not a unity filter will be applied.

Must be called before first time execution of Write Channel Configuration to FPGA.vi either with all channels defined or input not wired.

Input Parameters

Filter Settings, an array of clusters with the following elements:

- **Ch ID** is the unique channel identifier (0…511 or 0…255 depending on channel output protocol).
- **Filter Active** enables/disables the IQ filter.
- **FilterCoefficients** 55 symetric filter taps (+/- 1 value range), empty or wrong array size results in using a unity filter.

3.3.4.19 ReadAdvancedConfiguration.vi

© 2019 Novator Solutions AB

Used to read out the current settings for headers shape and size, old samples and CCF Core Parameters.

Output Parameters

Advanced Configuration

DRAM Active Headers defines which headers should be included in the channel data stream and how the dynamic header should be generated. See Header section for further details.

idHeader is an internal header.

numOfHdrs is the amount of headers including dynamic header to be used, ID and footer are excluded to this number

dynamicHeaderIdx define on which index the dynamic header will be inserted, zeroindexed and must be less than numOfHdrs. -1 to disable dynamic header. Read more in dynamic header.

footer is an internal header.

Dynamic Header Settings, see Dynamic Headers for further details.

Timestamp Index configures the location of the timestamp.

Count Index configures the location of the counter.

Byte Enable configures how the header should be written.

Old Data use these settings to specify how many and if they should be active. One user case scenario are when the signal needs to be filtered.

sendOldData enables or disabled the usage of old samples

nrOldSampleBuffers sets the amount of old samples buffers should be in use, each buffer are 16 samples.

CCF Core Parameters are used to configure the CCF core, depending on bitfile the values might need changes

Number of Filters should be set to 32 or 0 depending on bitfile

Number of Filter Taps if filter exist they should be 31 taps

Limits for the CCF core

Sample Rate Lower

Sample Rate Upper

Fine Gain Lower

Fine Gain Upper

Coarse Gain Lower

Coarse Gain Upper

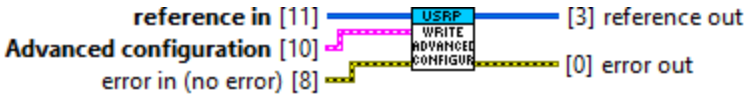
Number of Sources depending on the bitfile this number can be 1 or 4.

Number of Channels depending on bitfile this number can vary from 32-1024

© 2019 Novator Solutions AB

maxIpRate depending on bitfile this value might change

3.3.4.20 WriteAdvancedConfiguration.vi



Can be used to modify the headers shape and size, enable and edit old samples needed for filtering for example. CCF Core Parameters can be left as is if. Novator Solutions will notify upon bitfile delivery if any changes are needed.

Input Parameters

Advanced Configuration

DRAM Active Headers defines which headers should be included in the channel data stream and how the dynamic header should be generated. See Header section for further details.

idHeader is an internal header.

numOfHdrs is the amount of headers including dynamic header to be used, ID and footer are excluded to this number

dynamicHeaderIdx define on which index the dynamic header will be inserted, zeroindexed and must be less than numOfHdrs. -1 to disable dynamic header. Read more in dynamic header.

footer is an internal header.

Dynamic Header Settings, see Dynamic Headers for further details.

Timestamp Index configures the location of the timestamp.

Count Index configures the location of the counter.

Byte Enable configures how the header should be written.

Old Data use these settings to specify how many and if they should be active. One user case scenario are when the signal needs to be filtered.

sendOldData enables or disabled the usage of old samples

nrOldSampleBuffers sets the amount of old samples buffers should be in use, each buffer are 16 samples.

CCF Core Parameters are used to configure the CCF core, depending on bitfile the values might need changes

Number of Filters should be set to 32 or 0 depending on bitfile

Number of Filter Taps if filter exist they should be 31 taps

Limits for the CCF core

Sample Rate Lower

Sample Rate Upper

Fine Gain Lower

© 2019 Novator Solutions AB

Fine Gain Upper

Coarse Gain Lower

Coarse Gain Upper

Number of Sources depending on the bitfile this number can be 1 or 4.

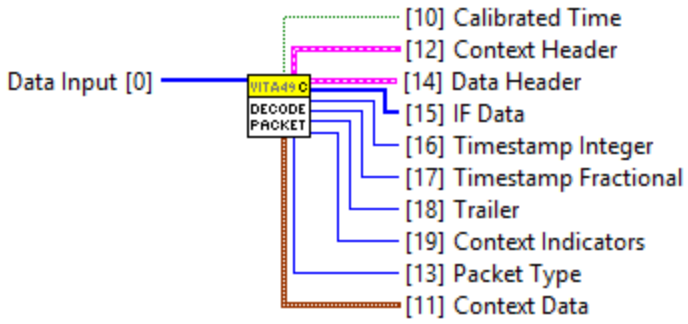
Number of Channels depending on bitfile this number can vary from 32-1024

maxIpRate depending on bitfile this value might change

3.3.4.21 VITA49

This library contains functions used to convert, pack, and unpack VITA49 header data easily.

3.3.4.21.1 Decode Packet.vi



Data received with Read RF Data.vi or Read Channel Data UDP.vi can be decoded using this function. Packet type out determines whether a UDP packet is a context or a data packet. Refer to VITA49 documentation for better understanding of this protocol.

Input Parameters

Data Input data to be decoded.

Output Parameters

Calibrated Time: Indicates if timestamp out is calibrated or not.
Context Header: static information about IF data either from the channel or from the RF stream
Packet Type "IF Context Packet", if data is valid in this cluster.
Class ID always false.

Timestamp Mode will be false and according to VITA49 standard this means that the time stamps in the packets are of "precise type", i.e. in the best precision the system can provide.

Timestamp Integer will be "Other". See VITA49 help documentation for more information.

© 2019 Novator Solutions AB

Timestamp Fractional will be "Real Time". See VITA49 help documentation for more information

Packet Count: unused for context packets.

Packet Size determines the size of the packet in amount of 32 bit words.

Stream ID: channel number, not used for RF data.

Data Header: static information about IF data either from the channel or from the RF stream

Packet Type IF Data with Stream ID, if data is valid in this cluster.

Class ID: always false.

Trailer: always true.

Timestamp Integer will be "Other". See VITA49 help documentation for more information

Timestamp Fractional will be "Real Time". See VITA49 help documentation for more information

Packet Count: incremental byte counter, use this to keep track of lost packets.

Packet Size: the number of 32-bit words in the packet.

Stream ID: tuner number or channel number.

IF Data IQ pair of IF data, either from channels or from tuners.

Timestamp Integer: time in seconds since start time was sent from host or the unit had locked itself to a GPS satellite.

Timestamp Fractional: fractional second that resets every time the second changes.

Trailer: information about data, always in pairs of indicator and status telling if this status should be considered or not.

Context Indicators: indicates which meta data are enabled for this context packet.

Packet Type: IF Data with Stream ID or IF Context Packet, use this to determine which indicators to update.

Context Data shows the enabled meta data indicated with context indicators.

Bandwidth of the tuner or the channel received.

IF Reference Frequency always zero for RF data and user configurable for channel data.

RF Reference Frequency of the tuner receiving the signal for RF data and user configurable for channel data.

Reference Level: since USRP-2955 is not a calibrated instrument, this reference level can be, set it is always 0 in the examples. Calibration of the system is needed to be able to use this with precision.

Gain (Stage 1) for RF data this is the gain applied to the signal, for channel data this is the Coarse Gain.

© 2019 Novator Solutions AB

Gain (Stage 2 Optional) not used for RF data, for channel data this is the Fine Gain.

Sample Rate of the channel or tuner data

3.3.4.21.2 Context Data Type Conversion.vi

This VI converts Contex data to a format required by the FPGA.

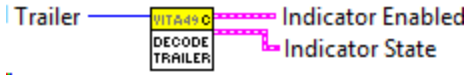
Input Parameters

Context Data Readable fixed point format.

Output Parameters

VITA49 Context Fields fixed point data converted to data type of VITA49 registers on FPGA.

3.3.4.21.3 Decode Trailer.vi



This VI decodes the trailer from VITA49 stream and outputs all indicators in use and their state.

Input Parameters

Context Data fixed point data format and readable format.

Output Parameters

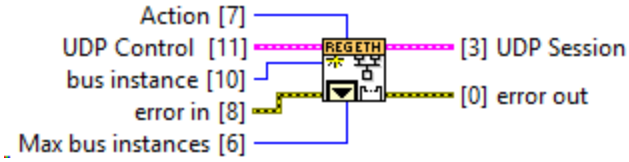
Indicator Enabled indicates if this state is to be considered.
Indicator State indicates status of all enabled indicators.

3.3.4.22 Register Bus Addition

Novator Solutions has developed additional VIs that need to be added to NI Register bus to account for the UDP connection.

After installation, these VIs will be found in the following location: %programfiles(x86)% \National Instruments\LabVIEW 2016\instr.lib_niInstr\Register Bus\v0\Host\Additional

3.3.4.22.1 RB_UDP_Connection_FGV.vi



© 2019 Novator Solutions AB

This VI will hold connection ID for communication with NC-10 USRP over UDP. This VI is used in the examples and USRP-setup.lvclass which is just an example on how you can configure the RF parameters and the operation of a USRP.

Input Parameters

UDP Control data needed to communicate with the USRP over UDP

- **Target SFP 0 IP** required for sending data with UDP write function in LabVIEW.
- **Register port** required for sending data with UDP write function in LabVIEW.
- **RegisterBus.connection** is used when reading data from USRP over UDP. This connection is always unicast.

Action "Read", "Reset", and "Write"

bus instance which instance of a bus you want, each USRP connected to the same system is meant to have 1 bus. This input is required when reading out a UDP session.

Max bus instances needs to be defined when you reset this VI, on reset all previous bus instances are closed and an empty array of max bus instances are created. Used Write action to add bus by bus to this FGV. Default value is 1 bus maximum.

Output Parameters

UDP Session data needed to communicate with the USRP over UDP for a specific bus instance

- **Target SFP 0 IP** required for sending data with UDP write function in LabVIEW.
- **Register port** required for sending data with UDP write function in LabVIEW.
- **RegisterBus.connection** is used when reading data from USRP over UDP. This connection is always unicast.

3.4 NC-10 Error Codes

403200	No available channels.
403201	Channel already removed.
403202	The requested channel bandwidth would cause the maximum aggregate output rate to be exceeded. Configuration not applied.
403203	The requested channel bandwidth is not within the limits.
403204	The CCF filter contained too few elements.
403205	The CCF filter contained too many elements.
403206	The multiple channel configuration contained errors. No changes have been applied. See error details for information about configuration errors.

© 2019 Novator Solutions AB

403207	The channelizer has not been configured and will therefore not start. Call the VI "Configure Channelizer.vi before "Open FPGA.vi".
403208	The requested center frequency is not within $\pm f_{in}/2$.
403209	Channel ID was coerced to a valid value (0-1023).
403210	Coefficient set was coerced to a valid value (0-31).
403211	Coefficient was coerced to a valid value (-131072-131071).
403212	Channels in the channel configuration was dropped.
403213	Gain not within limits (0-102.3 dB).
403214	Sum of filter coefficients not less than or equal to 131071.
403215	The buffer size value has been coerced to a valid range (40-2048).
403216	Filter coefficient set not within limits (0-31).
403217	The requested channel number does not exist.
403218	The requested Coarse bandwidth is not within the limits (0<Coarse BW<Sample Rate)
403219	Not a valid RF input source index for this target.
403220	Coarse gain not within limits (0-42,14 dB)
403300	Checksum failure, meaning some value written to FPGA was not received correctly.
403800	The number of FPGA Read configuration commands does not match the number of replies from the FPGA.
403801	The FPGA Read Configuration commands does not match the commands in the reply.
403802	The FPGA was not ready for update within 1 second from the command. Please try commiting the channel configuration again.
403803	The FPGA did not recieve the channel configuration within 1 second. Please try commiting the channel configuration again.
403804	The FPGA did not become ready for update after the update trigger was sent.
403805	The read back channel configuration did not match the configuration written to the FPGA.
403806	The pending FPGA configuration does not match the configuration sent from host.
403807	No configuration changes since last commit.
403808	FPGA was not ready for update when trying to write filter configuration.
403809	Read back filter configuration did not match the configured. Configure the filter again.

© 2019 Novator Solutions AB

403810	Address array and Data array is not of the same size.
403811	FlexRIO S/N from licence file was not found in the system.
403812	The S/N of the selected resource was not found in the license file.
403813	Error calculating the tuning parameters.
403814	Could not reset since reset was already taking place or done.
403815	Reset did not complete within 10 s.
403816	Coarse BW was coerced and rounded to the nearest integer Hz.
403817	Timeout while trying to read channel overflow information.
403818	The target does not support the selected chOutput option
403819	The coarse bandwidth have been coerced because of the internal processing. Use Get Channel Lists.vi to read actual bandwidth.

© 2019 Novator Solutions AB

NC-10 Operation

4 NC-10 Operation

This section aims to give a more detailed view and understanding of the options the API gives to the user.

The flowcharts in Figures 3 and 4 below demonstrate the different signal paths. The blocks in the flowcharts are explained in ensuing chapters.

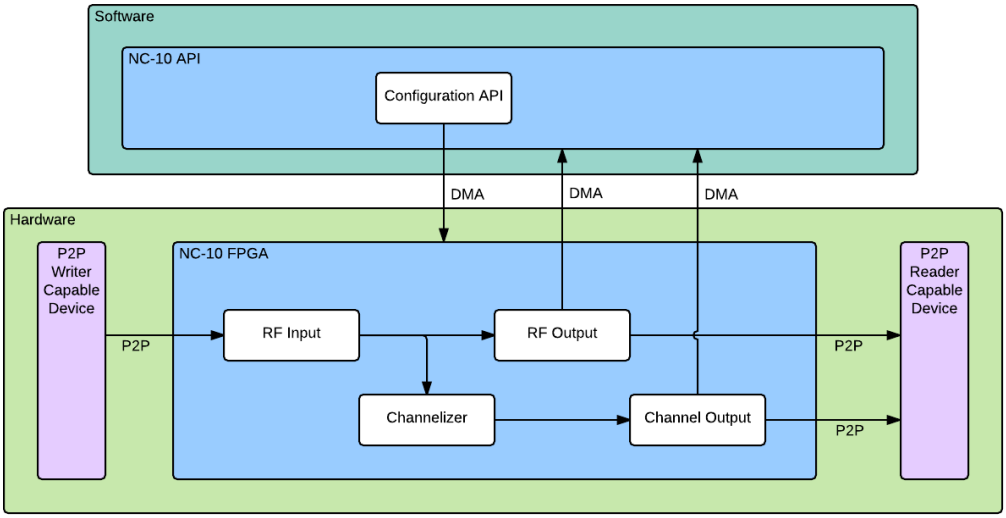


Figure 3: FlexRIO Target Flowchart.

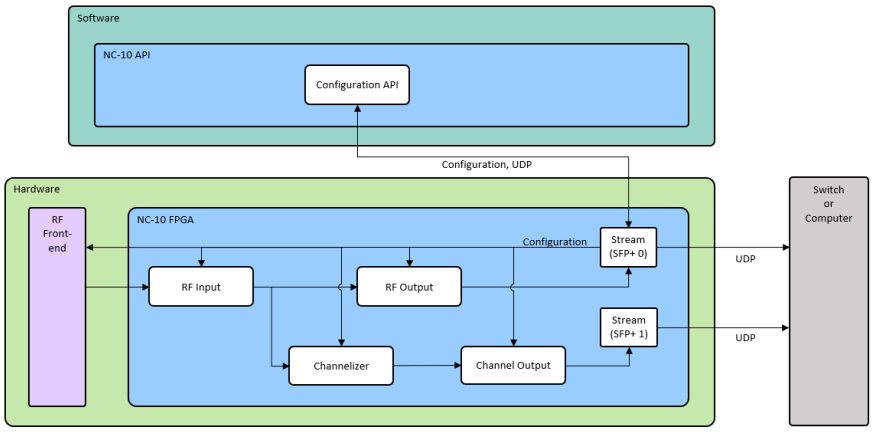


Figure 4: USRP-2955 Target Flowchart.

4.1 RF Input

NC-10 handles I16 IQ-data where one *I* sample and one *Q* sample are joined together to form an I32, where the Q sample constitutes the upper 16 bits.

Complex and Real Data

© 2019 Novator Solutions AB

All NC-10 functionality is developed with complex IQ-data in mind, but it is possible to operate with real data by setting the Q-part to 0. If the Q-part is set to 0 in your application (i.e. outside the scope of NC-10 and I32 is still input to NC-10), NC-10 should still operate in complex mode.

If only real data is sent to NC-10 (single I16 per sample), the real mode should be used.

Input Rate

NC-10 supports input rates of up to 250 MSps for FlexRIO but it is difficult to create applications with sustainable input rates of 250 MSps and using a lower rate is recommended. For USRP-2955 this input rate is fixed at 100MSps.

Scaling and offset

Scaling can be applied to the RF input of FlexRIO by using Configure_Channelizer_FlexRIO.vi and/or Configure IO FlexRIO.vi. When scaling, the I16 samples are internally converted to fixed point during multiplication of scale factor and addition of offset. The result is then converted back to I16 (round up half-even).

For a USRP target scaling is not available as it is an uncalibrated device.

4.2 Channelizer

The Channelizer (powered by ChannelCoreFlex Technology from RFEL LTD., UK) operates on the input RF data and extracts the channels. Table 1 lists the specifications of the channelization process.

Input rate (I/Q), f_{in}	≤250 MSps*
Number of RF Inputs	1 (FlexRIO) or 4 (USRP)
Number of output channels	512, 1024
Data width (in and out)	16-bit I, 16-bit Q
Output channel center frequency resolution	$f_{in}/2^{32}$
Output channel sample rate	Min: $f_{in}/2^{17}$ Max: $f_{in}/2$
Output channel sample rate resolution	$<f_{in}/2^{30}$
Maximum achievable aggregate output sample rate	250 MSps (FlexRIO) or 185MSps (USRP)
SFDR	>100 dBFS
SINAD	>90 dB
Output Channel alias-free bandwidth	80% of output sample rate
Output Channel alias filter rejection	>100 dB
Output Channel passband ripple	< ±0.1dB
Output Channel programmable gain	Coarse: 0 - 42.14dB (6.02 increment) Fine: 0 - 102.3dB (0.1 increment)

© 2019 Novator Solutions AB

Number of filters	32
Number of filter taps per filter	31

Table 1: Channelizer Specifications.

*Theoretical maximum. A lower input rate is recommended.

4.2.1 Channel Settings

The 512 or 1024 channels can be configured individually with regards to the following settings:

- Enabled/Disabled
- Source (relevant only for USRP)
- Coarse channel gain
- Fine channel gain
- Center frequency
- Channel sample rate
- Coarse bandwidth
- Filter coefficient set (31-tap filter).

VIs to configure these settings can be found in the Channel Configuration Palette.

It is possible to combine wide and narrow channels, have them overlap, etc. The channels can be reconfigured in runtime without disrupting the channelization of the other channels.

The channel sample rate is the only channel setting that depends on the settings of the other channels. The total aggregate output sample rate cannot exceed 250 MHz for FlexRIO and 185MHz for USRP-2955. When configuring channels with the Set Channel Sample Rate.vi or Set Channel Config.vi, the aggregate output rate of all enabled channels is calculated. If the sum exceeds the limit with the new sample rate, the sample rate is not updated and the channel is disabled. Due to the internal processing on the FPGA and the sample rates of the channels, the configured sample rate for a specific channel might differ from the ones used in the calculations of the aggregate output rate. The calculated internal aggregate sample rate could thus be higher than the aggregate sample rate would be by adding all output rates configured by the API.

When the configuration has been applied, variations for example in input timing (input data sent over the PXI-bus arrives in bursts for instance) could force the internal processing of the channelizer to discard channels. This is called *channel configuration overflow*. There are internal buffers in NC-10 to even out the bursts of input data, but if the variations become too large, channels with the highest ID will be disabled until the aggregate output sample rate criterion can be fulfilled. The channels will not be added until a new channel plan is committed to the channelizer. The closer the channel configuration is to a maximum capacity, the more likely this is to happen, use Check Status.vi to check for overflow conditions.

4.2.2 Output Shaping Filter

The internal processing in the channelizer handles the filtering in the channel decimation process. The guaranteed alias-free bandwidth is 80%. To shape the output and remove the possible remaining aliasing effects or do other shaping of the output, NC-10 includes one 31-tap filter.

31-tap Filter

© 2019 Novator Solutions AB

The 31-tap filter can always be used regardless of the selected operation mode. There are 32 sets of coefficients that can be configured separately and the channels can be configured to be processed by one of those 32 filters.

The filter is symmetrical, and thus only 16 coefficients need to be programmed.

On the FPGA a bitwidth of 18 bits is used for the coefficients, giving a range from -131072 to 131071. The sum of all the filter coefficients must be equal to or less than 131071 for unity gain.

If no shaping filtering is desired, the filter coefficient at index 16 should be set to 131071 and the rest to 0.

The filter coefficients are configured with Set CCF Filter.vi (can be read by Get CCF Filter.vi) and assigned to a channel with Set Channel Coefficient Set.vi.

4.2.3 Channel Gain

Two different gain settings exist for each channel, *coarse* and *fine gain*. To optimize dynamic performance the coarse gain should be as high as possible, while not causing any overflows. An overflow will lead to corrupted data which most likely will be unusable. Coarse gain overflow will result in wrapped data while fine gain overflow results in a clipped signal.

The fine gain can then be used to achieve the desired final gain.

4.2.4 Coarse Bandwidth

The *coarse bandwidth*, configured in Hz, is a filter in the early stages of the channelization process that allows you to filter undesired signals and noise. We recommend to set Coarse Bandwidth to the same value as sample rate. When committed to the FPGA, the bandwidth value will be coerced and the actual value can be read back in the "Commited Channels" list by using Get Channel Lists.vi.

4.2.5 Channelizer Overflows

Depending on target and number of channels activated there is a possibility to overflow the channelizer. If an overflow happens there will be loss of data. There are a few different overflows that can happen in the different processes of FPGA.

FlexRIO Target

Input sample rate might be too high making the internal FIFOs between RF tuner processes and channelizer core, as well as DMA/P2P output, overflow. When this overflow occurs loss of data will happen. If too many channels are enabled with too high rates there is a possibility that the internal DRAM process gets internal overflows that cause data to be lost. These overflows can be read with Check Status.vi. It is also possible to get an overflow on the receiving side, if the receiving process is running too slow. When reading DMA there are methods to check this state, whereas for P2P overflow indicators need to be implemented.

USRP Target

There can be internal overflows between the *RF tuner process* and the *channelizer core process* if the core does not get ready in time to receive new samples. There can also be overflows between the *RF tuner process* and the *UDP output streaming process* if too much traffic is sent through the specific port. If too many channels are enabled with too high rates there is a possibility that the internal DRAM process gets internal overflows that cause data to be lost. These overflows can be

 © 2019 Novator Solutions AB

read with Check Status.vi. It is also possible to get an overflow on the receiving side, if the receiving process is running too slow. This can be seen when the packet count does not increment correctly.

4.3 Timestamp

RF data and/or channel data can be timestamped and the way in which this is done depends on the target.

FlexRIO Timestamping

In FlexRIO, channel data is timestamped. On the FPGA, a counter is running, incrementing the count on each clock cycle.

The sent timestamp is calculated as:

Time=t0+counter

Where counter is the time since the last clock reset and t0 is an offset in nanoseconds set by the user through the API. The counter can be reset manually from the host. The configuration of the timestamp is done in Configure Timestamp.vi. The timestamp is taken after the sample leaves the channelization process. Because of the processing steps, the timestamp accuracy is ±40 µs from when the sample entered the channelizer.

The timestamp for the channel data is obtained by using Dynamic Headers and the timestamp is for the first sample in the buffer. Since the sample rate is known, the timestamps for the other samples can be calculated based on that.

USRP Timestamping

In USRP both RF and Channel data are timestamped. Since USRP has also GPS capability, it is possible to lock the internal clocks to GPS and have GPS timestamped data. If a GPS antenna is not connected, or if the internal clocks fail to lock to a GPS satellite, it is still possible to have timestamped data.

In order to make the timestamp GPS based, it is necessary to begin the application by implementing USRP driver calls required to start the locking procedure. Refer to the USRP Timestamp.vi example to see how this is done. The locking procedure may take about 10 minutes. After the GPS is locked it is necessary to get the GPS Start Time and start seconds and picoseconds counters on FPGA. To do this Set GPS Refresh Flag.vi must be called. If GPS Refresh Flag is set to True from host and the GPS is locked, new GPS time will be stored in a register that can be read through an API call. After that, seconds and picoseconds counters will start counting up. These counter values will be sent along with data via the VITA49 protocol. At the receiving side, it is important to keep in mind that timestamp of VITA49 package actually refers to the time *passed between the moment when the GPS start time is registered and the absolute time* that corresponds to data. Both GPS locking time and counter time inherit their accuracy from GPS.

Even if the GPS lock is lost, the internal FPGA counters will continue to count up. If the GPS lock is returned and the user wishes to update GPS Start Time, it is possible to do that by repeating the procedure described above. This will also make counters reset so that they count time passed from new Start Time.

If GPS is not used data can still be timestamped by using an API call to Set Start Time. In this case, the FPGA counters will start counting up from the computer time. The accuracy of the

 © 2019 Novator Solutions AB

absolute time will then depend on computer time and the accuracy of the relative time is equal to the internal FPGA clock accuracy. Every time new Start Time is written from host, the counters will be reinitialized. Note that if Start Time is written from the host using an API call, the timestamp will always be based on that even if the GPS is locked. In order to be sure that the GPS based timestamp is used, it is necessary to write 0 as time in Set Start Time.vi.

For information on how timestamp is output within VITA49 packages refer to Outputting VITA49 Data. Note that one second and one picosecond timestamp is output per packet and it refers to the first sample in the packet.

4.4 Synchronization

NC-10 has an option to have time synchronized channels. To obtain this it is necessary that the channels are committed to FPGA using Write Channel Configuration to FPGA.vi all at the same time, and with the same settings with regards to center frequency and sample rate. This due to the fact that each time the sample rate or the center frequency is changed for a channel, internal processing elements are restarted, which may shift the phase of the output signal. Since channel data are output from FPGA in an interleaved fashion, two channels with exactly the same settings committed so that they become synchronized, should have exactly the same content on a per buffer basis. This means that first buffer of one channel should have same data as first buffer of another channel with same settings and so on, provided that they are committed simultaneously.

Additionally, when using USRP, it may be required to phase synchronize channels coming from different RF inputs. For this, all RF inputs to the channelization process must be synchronized as well. The USRP tuners have this option. For this purpose it is necessary that the two daughterboards of the USRP share local oscillators (LO). When this is configured through USRP driver calls, then all RF inputs are phase coherent meaning that the phase difference between them is fixed. In order to make them phase aligned it is necessary to create software that would read RF data from all four inputs and calculate the phase difference. Then, this phase difference can be compensated for by setting the phase offset of the digital downconverters used in USRP FPGA. NC-10 gives the user the option to obtain RF data from all four inputs so that they can calculate the phase difference. It is recommended to take one of the RF inputs as a reference and then calculate the phase offset for the other RF channels so that they all match. In order to write the phase it is required to either use Write Register Array.vi or Write Register.vi from niInstr Register Bus v0 Host.lvlib using the Register Bus session wired to USRP-2955_Create.vi. Phase offsets for the four RF inputs are placed in subsystem 109 at the addresses 23, 37, 48 and 59. Phase offset data type expected by the register bus is Fxp <21,0>.

4.5 RF Output

The incoming RF data might be used for purposes other than channelization such as analysis or recording and it is therefore possible to forward the incoming RF data from the target device.

FlexRIO Target

When using a PXIe based target FPGA, raw RF data can either be forwarded to the host via a DMA FIFO or to other FPGA resources through peer-to-peer streaming. No handshaking is used when forwarding the data, which means that the incoming RF will be sent whether the downstream nodes are ready to receive the data or not. If the internal FPGA buffers fill up, data will be lost.

 © 2019 Novator Solutions AB

If the data is sent through P2P, all data is always forwarded. For this, it is necessary to set RF Output destination to P2P in Configure Channelizer_FlexRIO.vi.

If the data is sent to the host through a DMA FIFO, it is possible to limit the amount of data sent, which creates bursts of data. Parameters of burst can be set in Configure Channelizer_FlexRIO.vi or in Read RF Data.vi. If the RF data destination is Host, the data stream must be enabled as well. This can be done with Enable RF Stream FlexRIO.vi. Read RF Data.vi can be used to read RF data and/or check if a timeout has occurred. Timeout information can also be read in Check Status.vi.

The raw RF IQ data is sent in a form of 32 bit number where the most significant 16 bits represent Q. No metadata is sent.

USRP Target

When using a USRP target, RF data is forwarded via UDP packets. It is possible to limit the amount of data sent, which creates bursts of data. Parameters of burst can be set in Configure_Channelizer_2955.vi or in Read RF Data.vi. RF Data stream must be enabled as well. This can be done with Enable RF Stream 2955.vi. Data can be read by using Read RF Data.vi. RF IQ data is sent together with various Metadata using VITA49 formatting.

4.6 Channel Output

The channelized data can be sent from hardware in different ways depending on the target.

FlexRIO Target

Channel IQ data can be sent to the host application via DMA FIFO or via P2P to another FPGA-based NI product. If the channel output data is not read fast enough, the internal buffers in NC-10 will overflow since NC-10 does not have anywhere to send the data to. This will result in data loss and new data will overwrite the old data. The overflow status can be read in Check Status.vi.

Note that is not possible to turn the Channel Output off, which is possible with the RF output. To turn off channel output, disable the channels with the channel configuration VIs.

Channel IQ data for all enabled channels is sent through the same DMA or P2P FIFO in a single stream. Moreover, additional metadata can be sent alongside the data. Refer to Channel Buffering for more information on how data from different channels is combined with metadata and send through a single stream.

USRP Target

Channel IQ data is sent via UDP through SFP+ port 1 on USRP. Channel data from all enabled channels is combined with additional metadata and sent through the same UDP connection. Channel Data and Metadata are sent in VITA49 format.

4.7 Channel Buffering

In FlexRIO target data from the different channels are buffered and sent from the channelizer when the channel buffer is full. Each channel has its own buffers, which makes it possible to receive multiple consecutive samples from the same channel instead of single samples. This makes processing in downstream nodes easier.

© 2019 Novator Solutions AB

All channel data are streamed through the same DMA or P2P FIFO in a way that buffers containing data from different channels are interleaved. To be able to separate the channels Data Headers can be used.

As for the USRP target, the buffers are managed by the Write Header VITA49.vi, and it is not recommended that user adjusts these by calling Write Header.vi, even though it is possible.

4.7.1 Buffer Size

The buffer size can be configured individually for each channel with Write Buffer Size.vi.

The buffered samples are always sent in multiples of sixteen and the total number of samples will be:

Number of samples= buffer size*16.

> *NOTE: Having different buffer sizes for different channels leads to more complicated programming when reading the data, especially when reading on the host. If the same buffer size is always used, the same number of samples can always be read from the channel stream to get precisely one channel.*

4.7.2 Data Headers

Since all channel data is sent through the same peer-to-peer stream or DMA FIFO, it is necessary to be able to separate the channels. This can be done by using headers with identifiers, which can be unique for each channel. The headers can also be used to send information about the specific channel. Please refer to the examples regarding how headers can be used in channel identification.

A header is 512-bit long (cluster with 16 elements of I32) and can be configured to contain data relevant to your application.

There are 252 user defined headers available and one can be dynamic, which can be inserted to any index 0-251. There is also one fixed header and one footer that can be used for channel identification. The data output is ordered as follows:

- Fixed Header (512 bits)
- Hdr0 (512 bits)
- ..
- Hdr251 (512 bits)
- Dynamic (512 bits), replaces one of the 252 user defined headers
- Old data (32*16*number of old samples bits)
- Data (32*16*number of samples bits)
- Fixed Footer (512 bits).

The header content is specific for each channel but the option of sending a header is common for all channels. Which headers that are sent is configured in WriteAdvancedConfiguration.vi for FlexRIO and WriteAdvancedConfiguration.vi for USRP.

Example: If Hdr0 is configured to be sent, it will be sent for all channels regardless of whether it has been specified for all channels. Therefore, it is a good practice to define a default header for all channels.

Design Advice

© 2019 Novator Solutions AB

The header is a 16-element I32 cluster. If a 16-element array is more appropriate in your design, use the LabVIEW VI Convert Array to Cluster.

4.7.2.1 User Defined Headers

User defined headers always contain the same information until reconfigured via the API (Write Header.vi). If unconfigured, the data for that header is undefined. Examples of information the header can contain are sample rate and center frequency of the channel, but any information can be added. The static headers can also be used for channel identification when reading the channel stream.

4.7.2.2 Dynamic Headers

Dynamic headers contain information that changes between the different buffers while static header contain the same data until they are reconfigured. Dynamic headers have certain bytes that will be overwritten by the dynamic information.

Available dynamic information:

- Timestamp (U32+U64)
- Count (U32)

Timestamp are split in two sections, a U32 showing seconds and a U64 showing fractional seconds in picoseconds.

Count is the current buffer number for the specific channel since the last update of the channel.

The creation of dynamic headers is based on the byte index (0-63) in the 512-bit header and a byte enable, which enables which of the 64 bytes that should be written. The indices of Timestamp and Count, and byte enable are configured in the Write Advanced Configuration.vi. The dynamic headers are written using Write Header.vi. Note that the configuration is done in bytes, i.e. 64 bytes per header while the headers are written as 16 32-bit values.

A byte enable of -1 (binary representation all 64 places equal 1) would mean all bytes are written. A byte enable value of 1 (binary representation all 0's except least significant bit) means that only the least significant byte is written. If some of the dynamic information is not needed, set the corresponding index to a byte value not written to by some other value and use the byte enable to exclude the data.

Example

A header which contains the timestamp and count is required. The buffer count should be at index 2 in a byte array and the timestamp at index 7.

First the byte enable must be calculated, see table below. The byte enable and indices are used in the Channelizer Configuration VI. When the FPGA has started, the dynamic header is written using Write Header.vi, this corresponds to the D_0-D_{63}. When the channels are streamed with the dynamic header active, in this example the values D_2-D_5 and D_7-D_{18} will be overwritten with buffer count number and timestamp, respectively. The byte enable ensures that no other information is overwritten.

The table illustrates how it can be used.

Byte	**63 ..**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**

© 2019 Novator Solutions AB

Dynamic Header Info	D_{63}	$D_{..}$	D_{20}	D_{19}	D_{18}	D_{17}	D_{16}	D_{15}	D_{14}	D_{13}	D_{12}	D_{11}	D_{10}	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Timestamp (index 7)					T	T	T	T	T	T	T	T	T	T	T	T							
Count (index 2)																		C	C	C	C		
Byte Enable (524220)	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	0	0
Resulting Header	D_{63}	$D_{..}$	D_{20}	D_{19}	T	T	T	T	T	T	T	T	T	T	T	T	D_6	C	C	C	C	D_1	D_0

4.7.2.3 ID Header and Footer

This header is fixed and it is used internally to identify channels but can also be output and used in other applications to avoid using one of the static headers as channel identifier. The header can be used by comparing 16 elements that are read from the channel stream and match a certain number sequence. If it is a match, the program knows that the first header arrived and that a new channel buffer will follow.

Index	Data
0	Channel ID
1	Buffer number
2	Type (0=header, 1=footer, 2=N/A)
3-15	Number sequence to be matched.

4.7.2.4 Updating Channel Headers and Race Conditions

The header configuration is independent of the channel and buffer size configurations. Calling the channel configuration VI simultaneously with the header commit VI does not guarantee that they will be applied simultaneously.

Example

If a header contains information about the buffer size and the buffer size of the channel is changed, both header and channel configuration must be changed. Since nothing guarantees they are changed simultaneously, the data with the new buffer size might be output with the header for the old buffer size, causing a mismatch between header information and the actual data content.

To avoid problems with headers not matching the data, the recommended approach is to disable the channel and then reconfigure the header of that particular channel. To confirm the headers have been updated, read back the last written header and make sure it matches the written one. The channel can then be enabled with the new channel configuration which corresponds to the information stored in the header.

Another option is to pause the data output. The channelization process still continues but does not output any data. Channel data that is channelized during a pause is lost.

© 2019 Novator Solutions AB

4.7.3 Send Old Samples

It is possible to send overlapping data where at the beginning of one buffer, the last elements of the previous buffer are included. This makes it possible to do signal processing where the result depends on previous inputs.

Example 1:

Buffer 1 Data: 1, 2, 3, 4, 5

Buffer 2 Data: 6, 7, 8, 9, 10

Buffer 3 Data: 11, 12, 13, 14, 15

If old samples is activated and set to include 2 old samples, when Buffer 2 and 3 is sent out the following data will be sent:

Buffer 2 output: 4, 5, 6, 7, 8, 9, 10

Buffer 3 output: 9, 10, 11, 12, 13, 14, 15

Note that this example has been simplified to demonstrate the concept. Sample data are always sent as multiples of 16, and setting Old Samples to e.g. 10 means that 10*16=160 old data points will be sent with the new buffer.

Note that for the first buffer in a new channel configuration, data from the previous channel configuration will be sent as old samples. If no data has been sent from the channel before, the old data is undefined. This is to avoid having problems with different lengths of the first and next coming buffers.

Example 2:

When working with 512 or 1024 channels it is often not possible to have 512/1024 parallel signal processing paths and resources must be reused. When the channel data leave NC-10, buffers from different channels are time interleaved and the downstream processing must be able to handle switching between different channels if they share processing elements. In order to illustrate this, observe the following use case.

Consider a signal processing scenario where the processing output y[n] is dependent on current and previous input values x[n]…x[n-N], e.g. a FIR filter. The equation for a FIR filter of order N can be written as:

$$y[n] = \sum_{i=0}^{N} h_i * x[n-i] = h_0 x[n] + h_1 x[n-1] + \cdots + h_N x[n-N]$$

If Channel A is being sent from NC-10 and it needs to be FIR filtered, for calculating filtering output y[n], the algorithm must have current input sample x[n] and previous input samples x[n-1]… x[n-N] as well. Those data points are usually stored in registers etc. in the method performing the algorithm and thus saving the state. However, most of the times the method is not programmed to handle different incoming datasets and to save multiple states. Saving multiple states (one per channel in the case of processing channel data from NC-10) could potentially require a substantial amount of memory, especially on an FPGA.

© 2019 Novator Solutions AB

Instead, a certain amount of data from a previous buffer can be sent together with the current one in order to "reset" the state of the algorithm to a good state when the data from the current buffer is to be processed.

The number of old samples is user configurable and has to be sufficient so that it clears the whole memory of the processing element that was filled with data from different channel.

1 Channel

Channel A buffer is sent to the algorithm. When the buffer has been processed, the last samples are stored in registers etc.

The next buffer of Channel A is sent to the algorithm. The last processed Channel was A and no switching must be done.

2 Channels

Channel A buffer is sent to the algorithm. The last samples from Channel A are now stored in the registers.

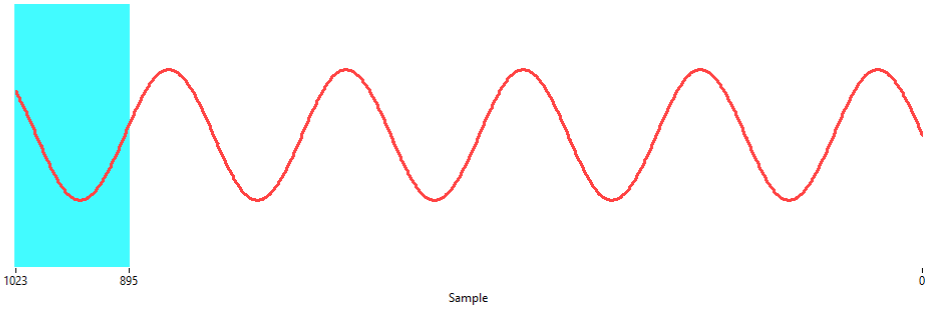


Figure 5: Data First Channel.

Channel B buffer is then sent to the algorithm.

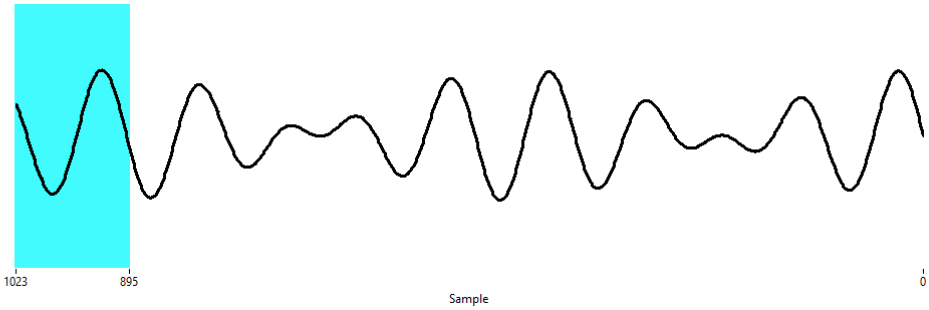


Figure 6: Erraneous Old Data 2nd Channel.

The last samples from Channel A are still stored in the registers when Channel B arrives. The output $y_B[n]$ is thus dependent on $x_A[n-1]$ to $x_A[n-N]$ which would give incorrect results.

The result when Channel A is sent again would also be incorrect.

2 Channels with Old Samples

© 2019 Novator Solutions AB

Channel A buffer is sent to the algorithm. The last samples from Channel A are now stored in the registers.

Channel B buffer is sent to the algorithm, but now the last samples from the previous Channel B buffer have been added in front of the current buffer.

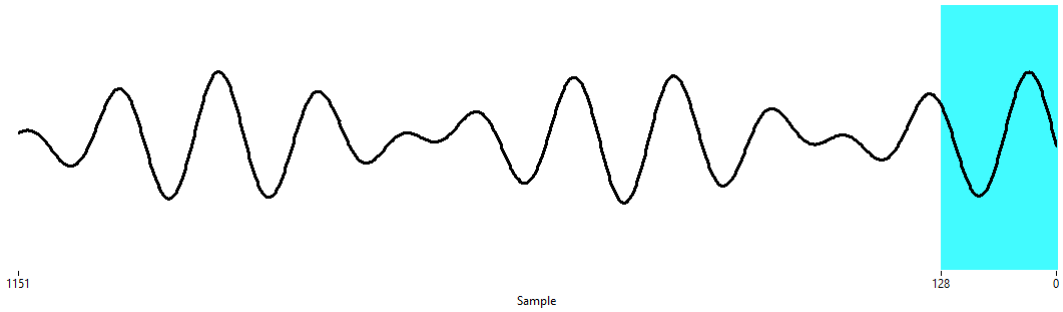


Figure 7: Two Channels with Old Samples Enabled.

The registers in the algorithm still have the values from Channel A, but as Channel B is being processed the Channel A Buffer values will be pushed out by the old samples from Channel B. When the current buffer of Channel B is about to be processed, the state of the algorithm is only dependent on $x_B[n-1]$ to $x_B[n-N]$

4.8 Outputting VITA49 Data

VITA49 is a packet based protocol used commonly in Software Defined Radio applications to send digitized signal data and metadata (or context data). In NC-10 for USRP, VITA49 is used for both RF data and Channel data. It is expected that future releases of NC-10 will support VITA49 for FlexRIO as well. In the remainder of this explanation both Channel and RF data will be referred to as IF data.

VITA49 outputs packets in such a way that first a Context Packet is sent and then an IF data packet, see figure 8 below. This pattern is then continuously repeated. The context packet is 106 bytes and the IF data packet is 8258 bytes, both including all layers (Link, Internet, Transport, Application).

© 2019 Novator Solutions AB

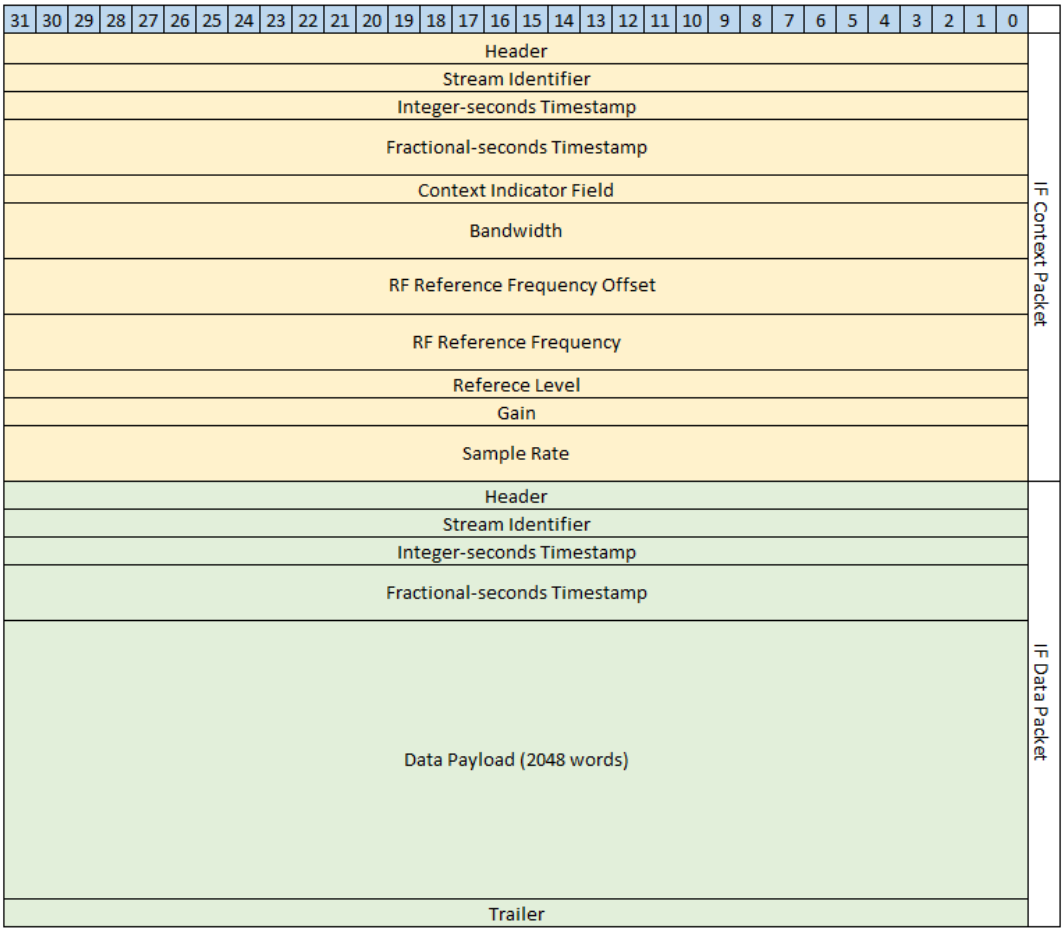


Figure 8: IF Packets.

4.8.1 IF Context Packet

IF Context Packet is shown in Figure 9 below.

© 2019 Novator Solutions AB

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Header (Mandatory)																															
Stream Identifier (Mandatory)																															
Class Identifier																															
Integer-seconds Timestamp																															
Fractional-seconds Timestamp																															
Context Section (Variable, Mandatory)																															

Figure 9: VITA49 Context Packet.

Header

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Packet Type				C	Rsv		TM	TSI		TSF		Packet Count				Packet Size															

Packet Type is 4 (0100) for IF Context Packet.

Class Identifier is not included in the IF data packet and set to false (0).

Bits 26 and 25 are reserved.

Timestamp Mode is false (0).

Timestamp-Integer is 3 (11) for “Other” time.

Timestamp-fractional is 2 (10) for “Real Time (picoseconds) Timestamp”.

Packet count is not used in IF context packet and will be constant 0.

Packet size is 14 (14x32bit words).

Stream Identifier

Stream identifier is used differently depending on whether we send RF or channel data. For RF data, this denotes RF tuner ID and for channel data it is Channel ID.

Class Identifier

Class identifier is not used since it is disabled in the header.

Timestamp Integer

This is a time in seconds since the GPS module locked to a GPS signal or since time set with Set Start Time.vi.

© 2019 Novator Solutions AB

Timestamp Fractional

This is a fractional seconds counter which will reset when Timestamp Integer changes seconds value.

Context Section

The last section of a context packet is the context section.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Context Indicator Field																															
Context Fields (Present as indicated by the Context Indicators)																															

Figure 10: Vita49 Context Section.

© 2019 Novator Solutions AB

Bit Position	Context Indicator Field	Number of words
31	Context Field Change Indicator	0
30	Reference Point Identifier	1
29	Bandwidth	2
28	IF Reference Frequency	2
27	RF Reference Frequency	2
26	RF Reference Frequency Offset	2
25	IF Band Offset	2
24	Reference Level	1
23	Gain	1
22	Over-range Count	1
21	Sample Rate	2
20	Timestamp Adjustment	2
19	Timestamp Calibration Time	1
18	Temperature	1
17	Device Identifier	2
16	State and Event Indicators	1
15	Data Packet Payload Format	2
14	Formatted GPS Geolocation	11
13	Formatted INS Geolocation	11
12	ECEF Ephemeris	13
11	Relative Ephemeris	13
10	Ephemeris Reference Identifier	1
9	GPS ASCII	Variable
8	Context Association Lists	Variable
7	Reserved	
6		
5		
4		
3		
2		
1		
0		

The enabled context indicators are

- 29, Bandwidth
- 27, RF Reference Frequency
- 26, RF Reference Frequency Offset
- 24, Reference Level

© 2019 Novator Solutions AB

- 23, Gain
- 21, Sample Rate

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bandwidth																															
RF Reference Frequency																															
RF Reference Frequency Offset																															
Referece Level																															
Gain																															
Sample Rate																															

Figure 11: Context Indicators.

The size of the context section is 11x32bit words.

4.8.2 IF Data Packet

IF Data Packet format is shown in Figure 12 below.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Header (Mandatory)																															
Stream Identifier																															
Class Identifier																															
Integer-seconds Timestamp																															
Fractional-seconds Timestamp																															
Data Payload (Variable, Mandatory)																															
Trailer																															

Figure 12: VITA49 Data Packet.

Header

IF data packet header is the same as the header of the IF context packet except for the omission Timestamp Mode bit and the addition of the Trailer bit.

© 2019 Novator Solutions AB

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Packet Type				C	T	Rsv		TSI		TSF		Packet Count				Packet Size															

Packet Type is 1 (0001) for IF Data Packet with Stream Identifier.

Class Identifier is not included in the IF data packet and set to false (0).

Trailer is included and set to true (1).

Bits 25 and 24 are reserved.

Timestamp-Integer is 3 (11) for “Other” time.

Timestamp-fractional is 2 (10) for “Real Time (picoseconds) Timestamp”.

Packet count contains a modulo-16 count of IF data packets.

Packet size is 2054 (2054x32bit words).

Stream Identifier

Stream identifier is used as RF tuner ID or channel ID.

Class Identifier

Class identifier is not used since it is disabled in the header.

Timestamp Integer

This is a time in seconds since GPS module locked to GPS signal or since time set with Set Start Time.vi.

Timestamp Fractional

This is a fractional seconds counter which will reset when Timestamp Integer changes seconds value.

Data Payload

Data Payload is always 8192 bytes (2048x32bit words).

Trailer

Trailer is enabled in the header. The trailer contains fields that indicate the validity of the data. The enabled bits are

- 31, Calibrated Time
- 30, Valid Data

© 2019 Novator Solutions AB

Enable Bit	Indicator Bit	Indicator Name
31	19	Calibrated Time
30	18	Valid Data
29	17	Reference Lock
28	16	AGC/MGC
27	15	Detected Signal
26	14	Spectral Inversion
25	13	Over-range
24	12	Sample Loss
23	11	User-Defined 0
22	10	User-Defined 1
21	9	User-Defined 2
20	8	User-Defined 3

Calibrated Time

The Calibrated Time Indicator, when set to one, indicates that the Timestamp in the IF Data packet is calibrated to some external reference. When set to zero it indicates that the Timestamp is free running and may be inaccurate.

Valid Data

The Valid Data Indicator, when set to one, indicates that the Data in the packet is valid. When set to zero it indicates that some condition exists that may invalidate the Data. The meaning of Valid Data Indicator is application dependent. When used in the output of an RF tuner, for example, it indicates that the Data in the packet is invalid during frequency changes.

4.9 Troubleshooting

Using the USRP version of NC-10, the communication with the device can fail with error -367001. This can occur when too wide channels are configured or too many channels are enabled, and all channels are sent to the same computer that also manages the commands and RF data via its SFP+0 interface.
Preferred use case is to separate command/RF data interface from channel output, either by broadcasting the channel data via switch or directly but to another computer/network interface.

© 2019 Novator Solutions AB

Deployment

5 Deployment

NC-10 can be used in deployed applications. This section describes things to consider when deploying the application.

5.1 Deployment Licensing

The same license file that is used during development can be used in deployed systems to enable the full functionality of NC-10.

5.2 Building Executables

When building a LabVIEW executable, all dependencies must be included. NC-10 dependencies that are not automatically included in the build process are:

- License file
- NC-10 bitfile (see the target-specific xxx_Create.vi for more information)

FlexRIO

The dynamically called classes that needs to be added to build specifications are:

DRAM IO_Class
CCF IO_Class
SrcIO_Class

USRP

The dynamically called classes that needs to be added to build specifications are:

DRAM IO RB_Class
CCF IO RB_Class
SrcIO RB_Class

© 2019 Novator Solutions AB

Contact Information

6 Contact Information

For support related to NC-10, please email nc10support@novatorsolutions.se.

Company Name	Novator Solutions AB
Street Address	Box 744
Zip Code	SE-191 27
City	Sollentuna
Country	Sweden
Web	http://www.novatorsolutions.com
Email	info@novatorsolutions.se
Telephone	+46 8 622 63 50

© 2019 Novator Solutions AB