LabVIEW FPGA Advanced Interface Tools

These tools are deprecated and no longer maintained. For new projects, use the LabVIEW FPGA Advanced Session Resources instead.

LabVIEW FPGA System

DAQmx Comparison

To understand the paradigm of a LabVIEW FPGA system, it is important to understand the specific components found in a traditional measurement system. A traditional measurement and control system consists of three basic components (see figure 1):

- The application software running in Windows or a Real-Time operating system (RTOS). For you, this would be the LabVIEW Development Environment.
- The driver software and functions to interface to your hardware. This would be the NI-DAQmx driver.
- And finally, the I/O hardware. For this specific example, we are calling out an X-series DAQ device.

With this paradigm, end users receive only the functionality that our developers designed into the ASIC of the X Series. Additionally, an extensive driver is required to make all of the device functionality accessible through software. The drawback to this approach is that all of the device functionality is fixed. The customization of such systems comes through the programming at the application software layer which can result in limitations of the hardware functionality and complex software driver calls to perform particular tasks.

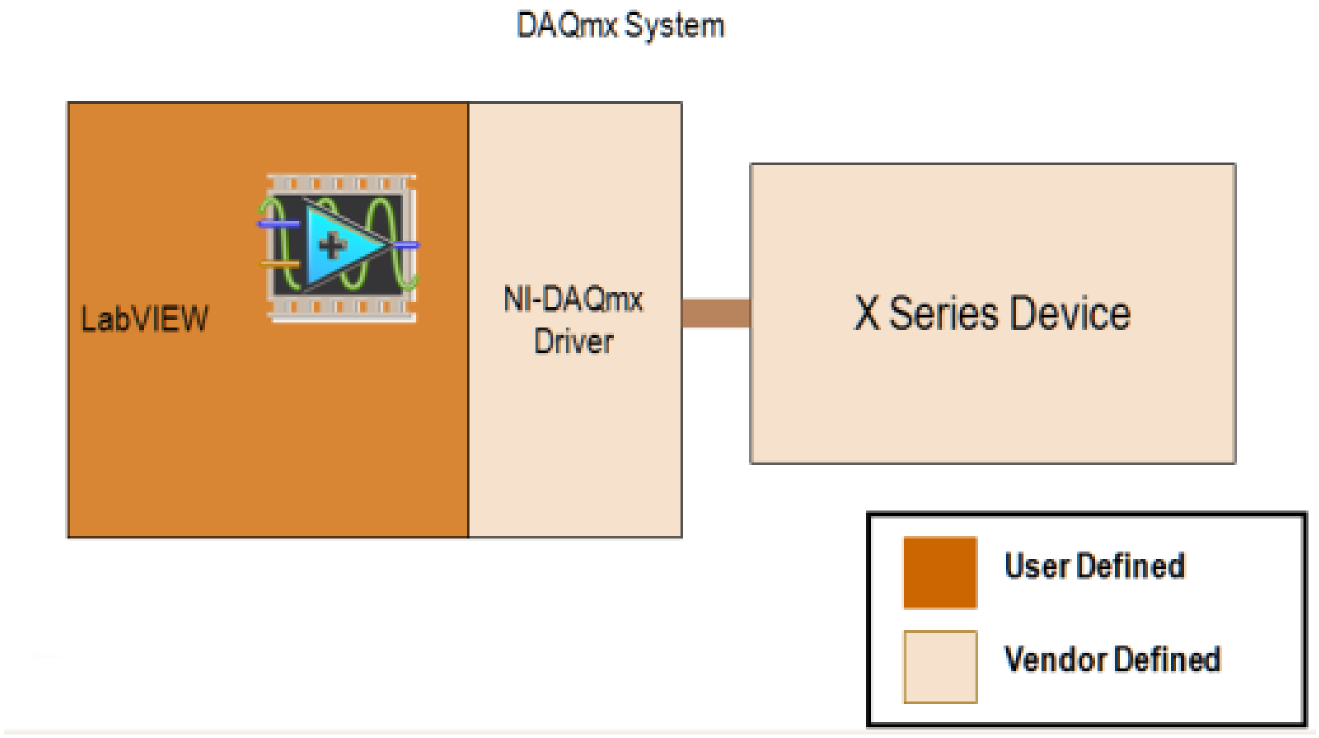


Figure 1: DAQmx System

When using a LabVIEW FPGA-based measurement system, the paradigm changes. While you are still using COTS hardware from National Instruments, you can now modify the hardware functionality of this device using LabVIEW FPGA. On the RIO hardware platforms, the FPGA (instead of an ASIC) defines the device functionality. You still have the application software running in Windows or on an RTOS, however, now there is a very thin driver called NI-RIO that provides the interface functions to our hardware. With this paradigm, end users have an I/O platform with the performance, flexibility, and customization of an FPGA without the added complexity of having to understand FPGA and board-level hardware design.

However, the additional development effort required for a LabVIEW FPGA system can often be overlooked. It requires programming of the physical hardware or FPGA. This program can be thought of as the firmware for the RIO device. The firmware is accessed through the NI-RIO driver, as stated previously, but this is an incomplete view. There is an FPGA

Interface that sits on top of the NI-RIO driver. This interface is dependent upon the firmware created for the RIO hardware. Even with this interface, there is still an incomplete view when comparing the RIO platform to a traditional DAQ platform. To give a fair comparison, one has to evaluate the RIO platform on the basis that a user driver must be created at a layer above the FPGA interface (see figure 2). A good user driver is needed for maintainability, scalability, and reusability for high return on investment.

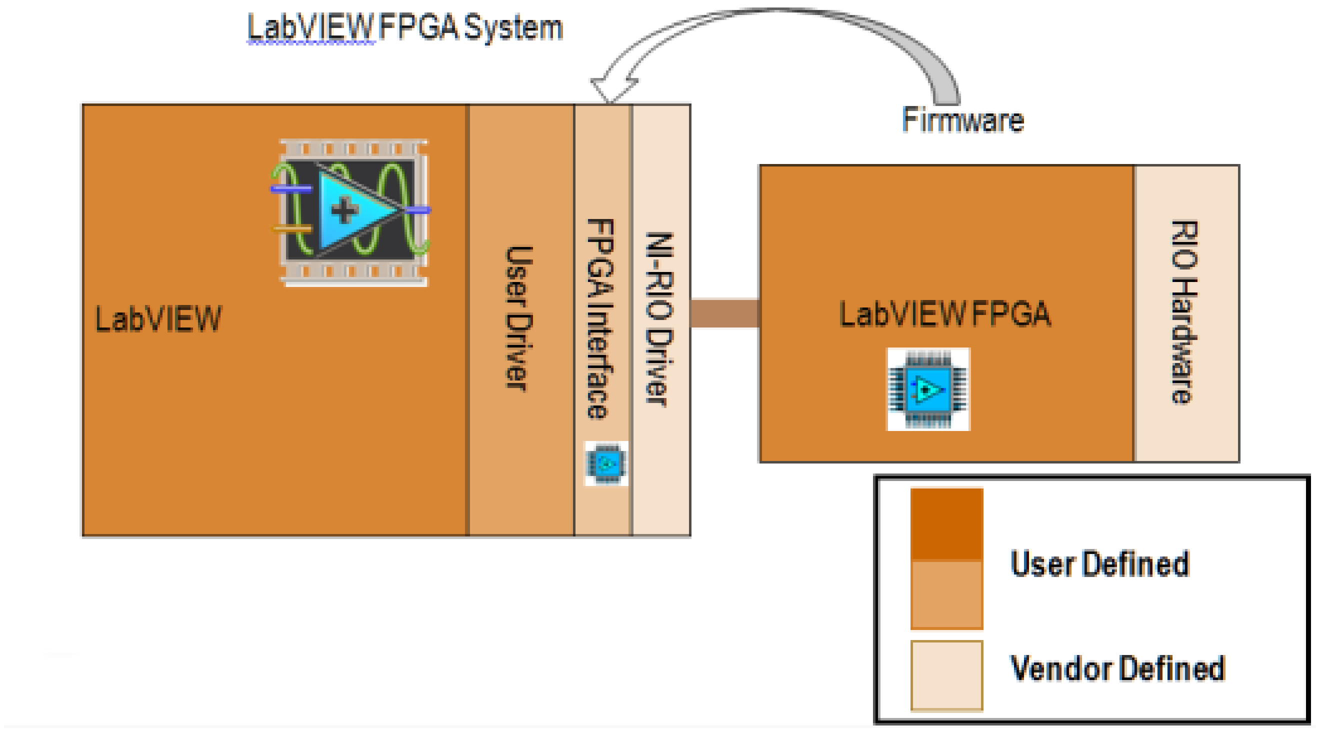


Figure 2: LabVIEW FPGA System

FPGA Interface Comparison

The user driver and firmware both cost money in terms of development effort, and the capabilities of the FPGA interface significantly impact the cost of the user driver.

Static Mode

Prior to LabVIEW 2010, the FPGA interface was completely static and very inflexible to future changes because FPGA references are incompatible with each other in this mode. Even though the control, indicator, and FIFO names may be identical between multiple bitfiles, they are unable to share a single FPGA reference wire in static mode. The static nature of the interface cost significantly due to its negative impact on scalability, maintainability, and reusability. Figure 3 demonstrates an example user driver developed in static mode. As you can see, the FPGA Interface is abstracted by the user driver in order to provide a standard API to access the instrument created by the RIO platform and LabVIEW FPGA.

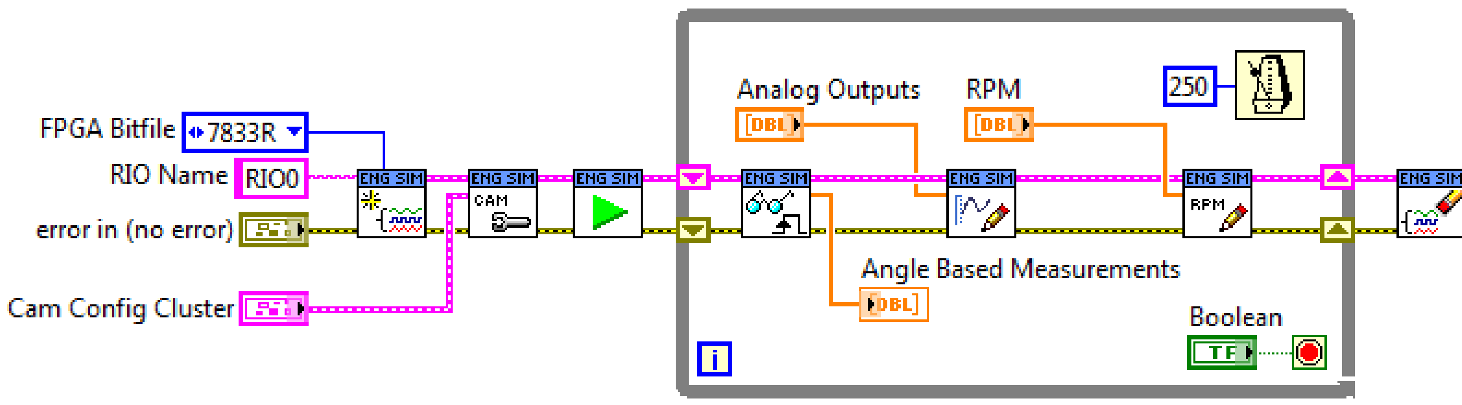


Figure 3: Static Interface Driver

The user driver shown in figure 3 supports multiple types of FPGA targets as seen by the enum input labeled *FPGA Bitfile* by providing a cluster that contains this enum along with each type of bitfile reference supported byt he driver. The Init subVI for the user driver is shown in figure 4 which demonstrates how the enum and references are used in the cluster.

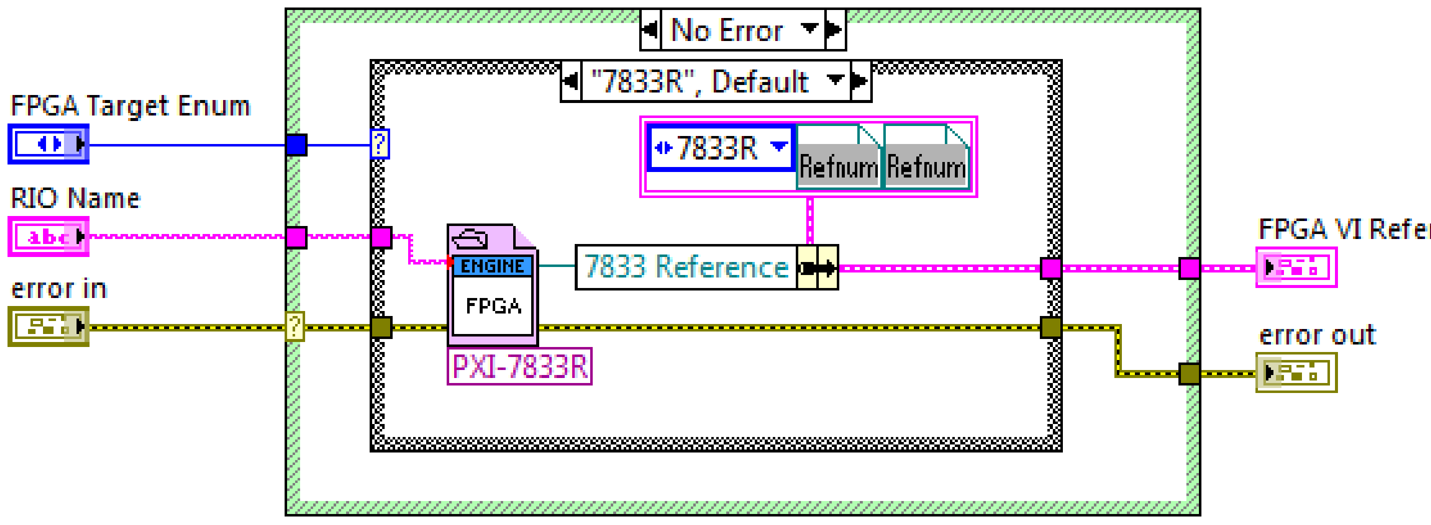


Figure 4: Static Interface Driver's Init subVI

Each user driver subVI call, must read the enum type and use a case structure to operate on the appropriate FPGA reference as shown in figure 5. In order to maintain the user driver, each case of any subVI that may need a change must have the same change updated to it. In order to scale the user driver to support another bitfile, the developer must update the enum with the additional selection and then update the cluster with the addition FPGA reference. Then the developer must add the corresponding case to each subVI within the user driver to handle the new bitfile. Even if the same FPGA type (i.e. 7833R) is used, but a new bitfile is compiled for different I/O connectivity, a new selection in the enum and a new case in the case structures must be created. The static nature of this interface type results in a very large cost associated with maintainability, scalability, and reusability of a user driver developed with this interface.

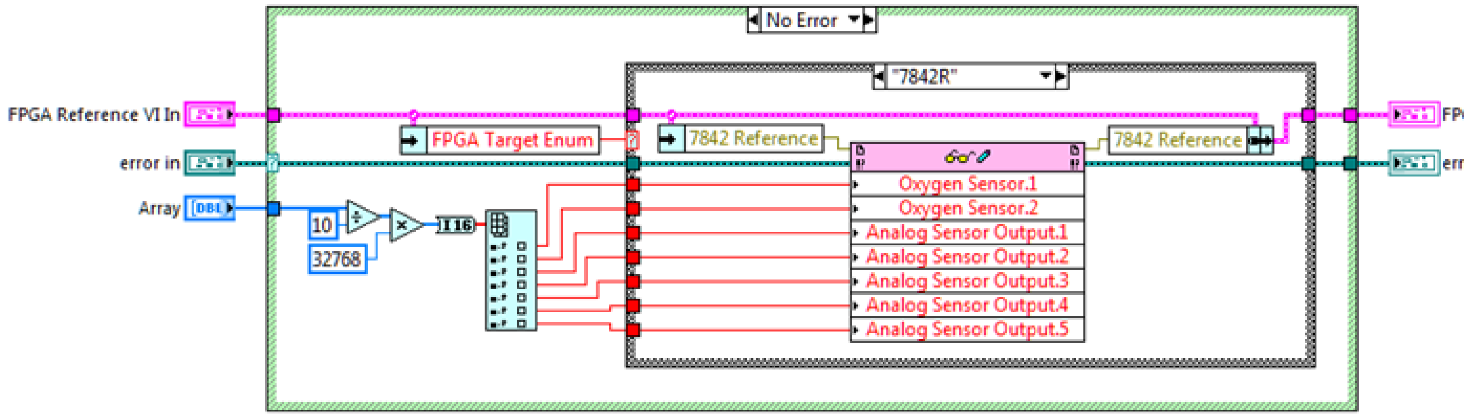


Figure 5: Static Interface Driver's Write subVI

Dynamic Mode

LabVIEW 2010 introduced the dynamic mode for the FPGA interface. Dynamic mode provides a mechanism for mutiple bitfiles to share a common FPGA reference as long as they have the same named set of controls, indicators, and FIFOs. A user driver using dynamic mode is shown in figure 6.

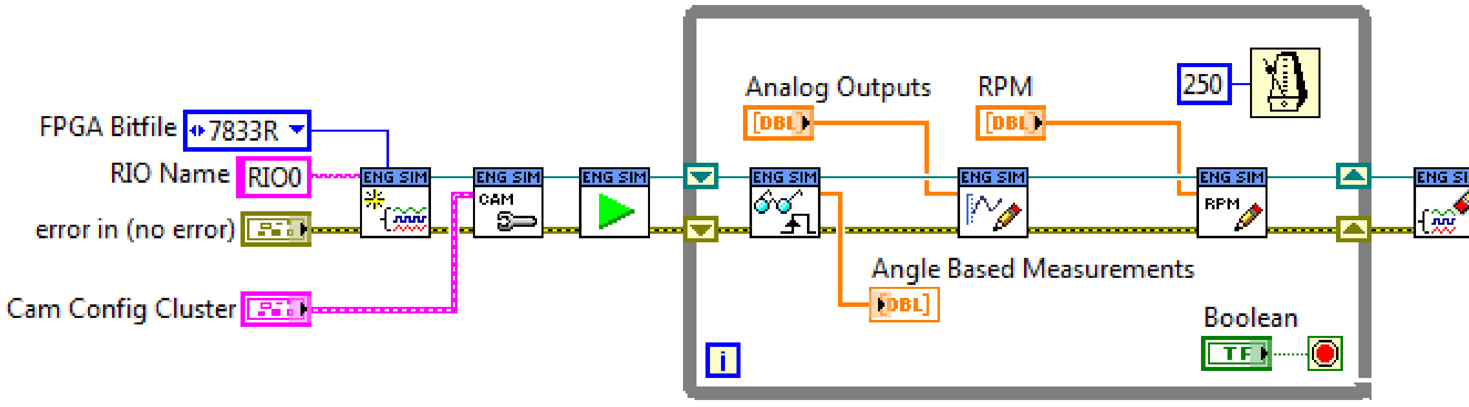


Figure 6: Dynamic Mode Driver

As you can see in figure 6, the dynamic mode allows the user driver to use a single FPGA reference (teal colored wire passed to each subVI). Figure 7 shows the init subVI of the user driver which demonstrates that a single reference wire is shared among all the cases.

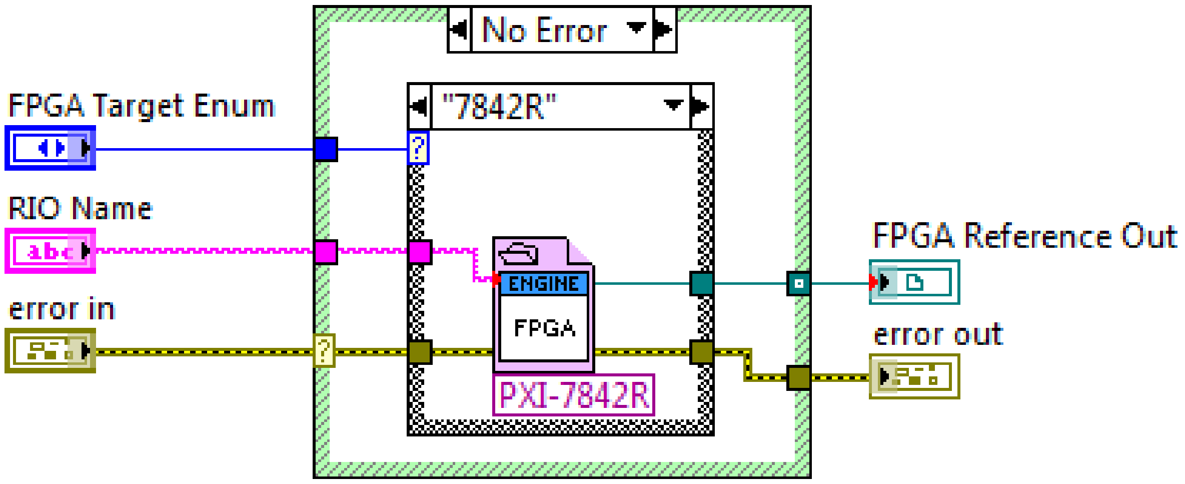


Figure 7: Dynamic Mode Driver's Init subVI

With the dynamic mode FPGA interface, there is no longer any maintenance on the user driver API calls other than the Init in order to scale to multiple FPGA bitfiles. As shown in figure 8, user driver subVIs do not require any type of changes to support additional bitfiles. If all the user driver subVIs use FPGA references that are bound to a single TypeDef, the maintenance on the interface is handled by the TypeDef and any changes made propagates to all user driver subVIs.

Figure 8: Dynamic Mode Driver's Write subVI

Dynamic Interface

The dynamic mode introduced in LabVIEW 2010 has resolved most of the issues from the static mode interface that negatively impacted maintainability, scalability, and reusability. In that evaluation, the Init subVI of the user driver still has some maintenance issues associated with supporting multiple bitfiles. The cost burden associated with the Init subVI can be eliminated by removing it from the user driver's API. By decoupling the Open FPGA Interface from the user driver, the user driver becomes very reusable and no longer has maintenance associated with supporting extra bitfiles as seen in figure 9.

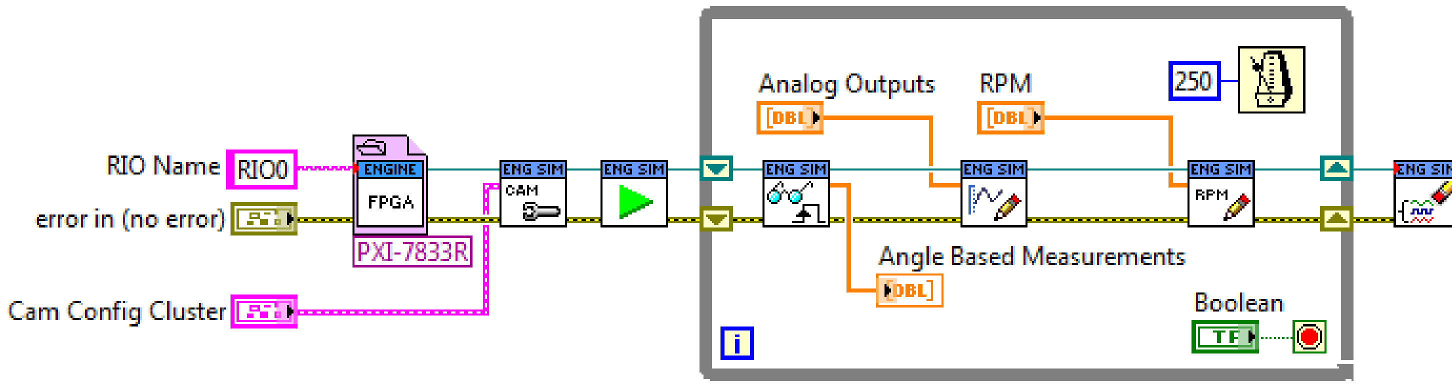


Figure 9: Dynamic Mode Driver with Decoupled Init

Once the Open FPGA Reference is used in place of an Init subVI in the user driver, how can an application choose which bitfile to use at run-time? It turns out that the Open FPGA Reference does not allow bitfile specification at run-time. In the example of a PXI-7833R bitfile, the top-level VI has a file size of ~21 kB before the Open FPGA Reference is configured with the bitfile and has a size of ~1,283 kB after the bitfile is specified in the Open FPGA Reference. The reason for this discrepency is because the top-level VI will store the bitstream within the VI so that the bitstream is always transferred with the VI.

The implication of this design decision is that a bitfile cannot dynamically be chosen at run-time. If multiple bitfiles need to be supported, a case structure which chooses the appropriate Open FPGA Reference is required as was previously demonstrated within the Init subVI of the user driver. Any time an application needs to support a new bitfile (even if the control, indicator, and DMA FIFO names are the same) or updated to utilize a new compiled version of an FPGA VI, the top-level application must be recompiled in order to support the new bitfile. This places an undesired maintenance burden on the application instead of only needing to update a single bitfile on disk.

Dynamic mode allows FPGA bitfiles to share a single reference because it determines register addresses at run-time with its underlying architecture being built upon dynamic features. Those dynamic features are not, however, available to the developer creating the user driver. The bitfile and names of controls, indicators, and FIFOs being accessed must be statically set by the developer at edit time. In order for Dynamic Mode to truly be a Dynamic Interface it needs the following two additional features:

1. Dynamic (or run-time) specification of bitfile.
2. Dynamic (or run-time) specification of controls, indicators, and DMA FIFOs.

LabVIEW FPGA Advanced Interface Tools

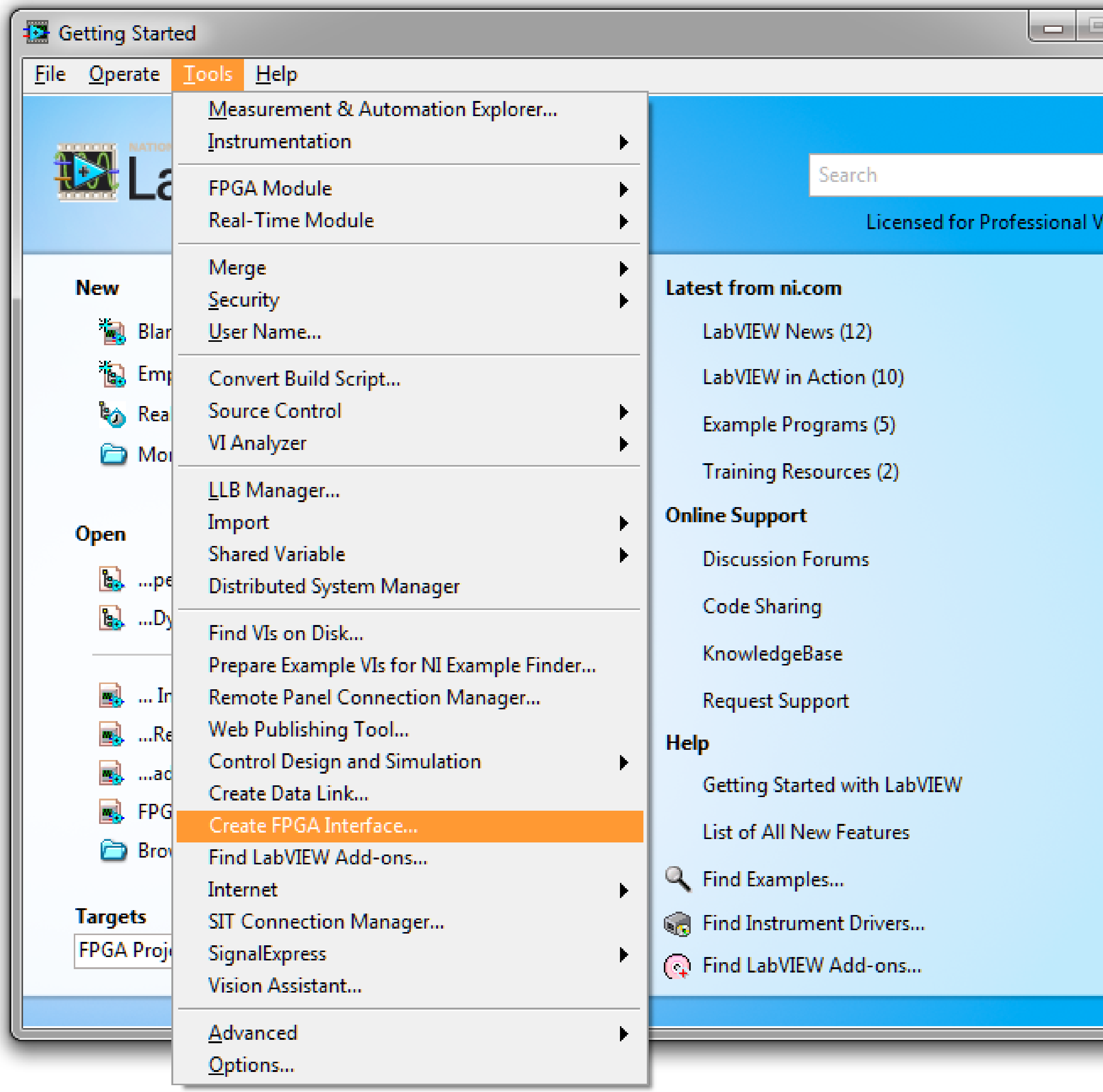
Overview

The *LabVIEW FPGA Advanced Interface Tools* were created in order to provide dynamic interfaces by providing the two features left out of the *Dynamic Mode* interface. The download provided at the bottom of this document installs *LabVIEW FPGA Advanced Interface Tools* that includes:

1. *Firmware-Independent FPGA Interface* which provides dynamic (or run-time) specification of bitfile.
2. *Fully Dynamic FPGA Interface* which provides dynamic (or run-time) specification of controls, indicators, and DMA FIFOs in addition to dynamic specification of bitfile.

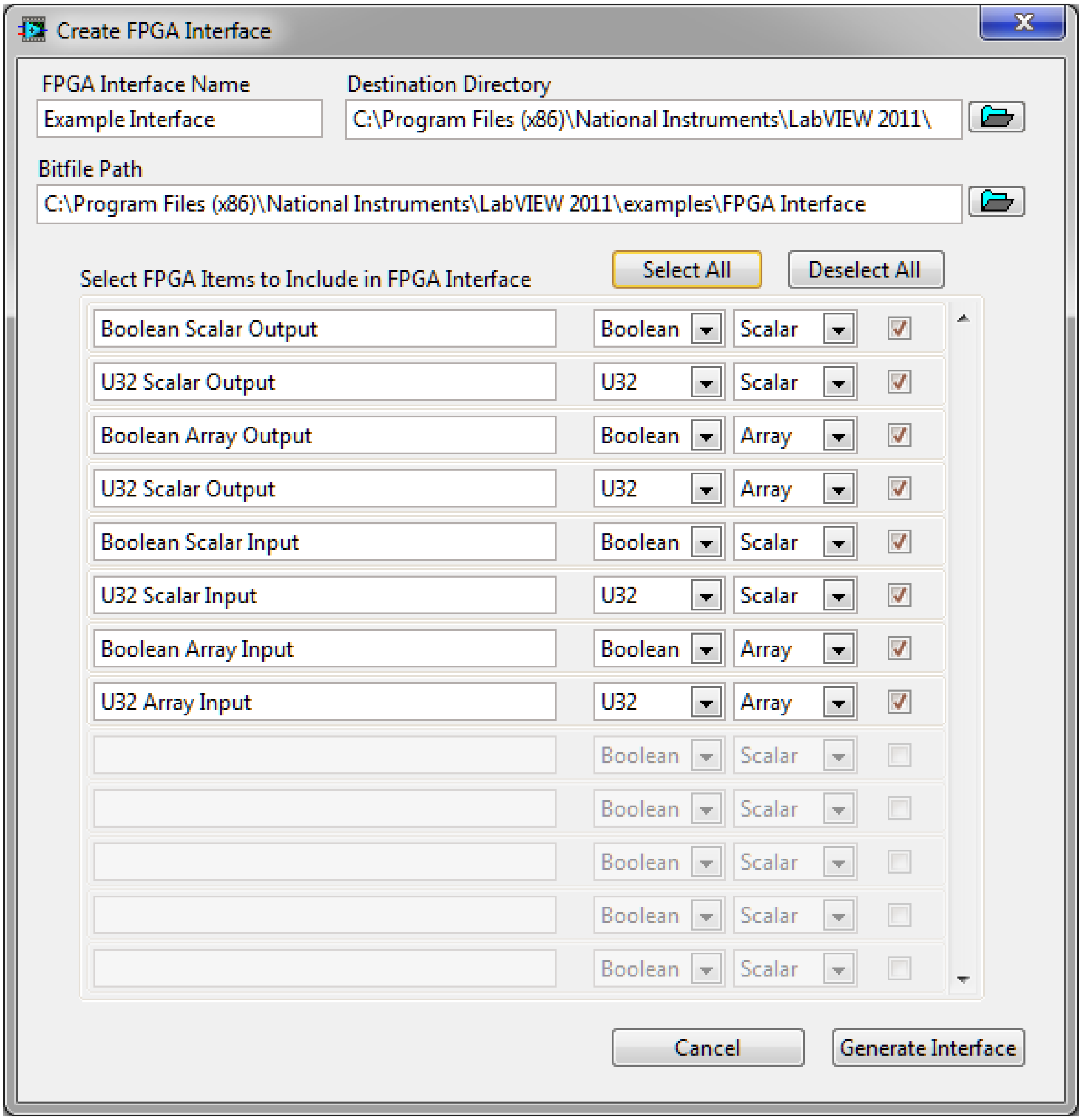
Firmware Independent FPGA Interface

The firmware-independent FPGA interface still provides a static specification of control, indicator, and FIFOs like the dynamic mode interface, but it provides run-time specification and loading of the bitfile (or firmware) by file path. This enables firmware updating without the need for user driver or application recompilation. After installation of the tools, one can create a firmware-independent FPGA interface by going to the "Tools" menu in LabVIEW and selecting "Create FPGA Interface...".



10: Tools Menu Selection

After opening the tool to create the firmware independent FPGA interface, configure the interface name, the controls, indicators, and FIFOs supported by the interface, and the directory where the interface will be saved as shown in Figure 11. The directory browse path will begin in the user.lib directory. Save the interface to user.lib if you'd like the interface to show up on the palette under *User Libraries*. A bitfile with the interface controls, indicators, and FIFOs is required prior to running this utility.



FPGA Interface Name: Example Interface

Destination Directory: C:\Program Files (x86)\National Instruments\LabVIEW 2011\

Bitfile Path: C:\Program Files (x86)\National Instruments\LabVIEW 2011\examples\FPGA Interface

Select FPGA Items to Include in FPGA Interface

Boolean Scalar Output	Boolean	Scalar	✓
U32 Scalar Output	U32	Scalar	✓
Boolean Array Output	Boolean	Array	✓
U32 Scalar Output	U32	Array	✓
Boolean Scalar Input	Boolean	Scalar	✓
U32 Scalar Input	U32	Scalar	✓
Boolean Array Input	Boolean	Array	✓
U32 Array Input	U32	Array	✓
	Boolean	Scalar	
	Boolean	Scalar	
	Boolean	Scalar	
	Boolean	Scalar	
	Boolean	Scalar	

Figure 11: Create FPGA Interface Dialog

Fully Dynamic FPGA Interface

The fully dynamic FPGA interface provides dynamic specification of control, indicator, and DMA names at run-time. To make this feature useful, it also provides a bitfile parser that enables a user to query the avaiable control, indicator, and FIFOs that were compiled in the bitfile. The fully dynamic FPGA interface and bitfile utility are both accessible from the Addons palette within the Functions palette.

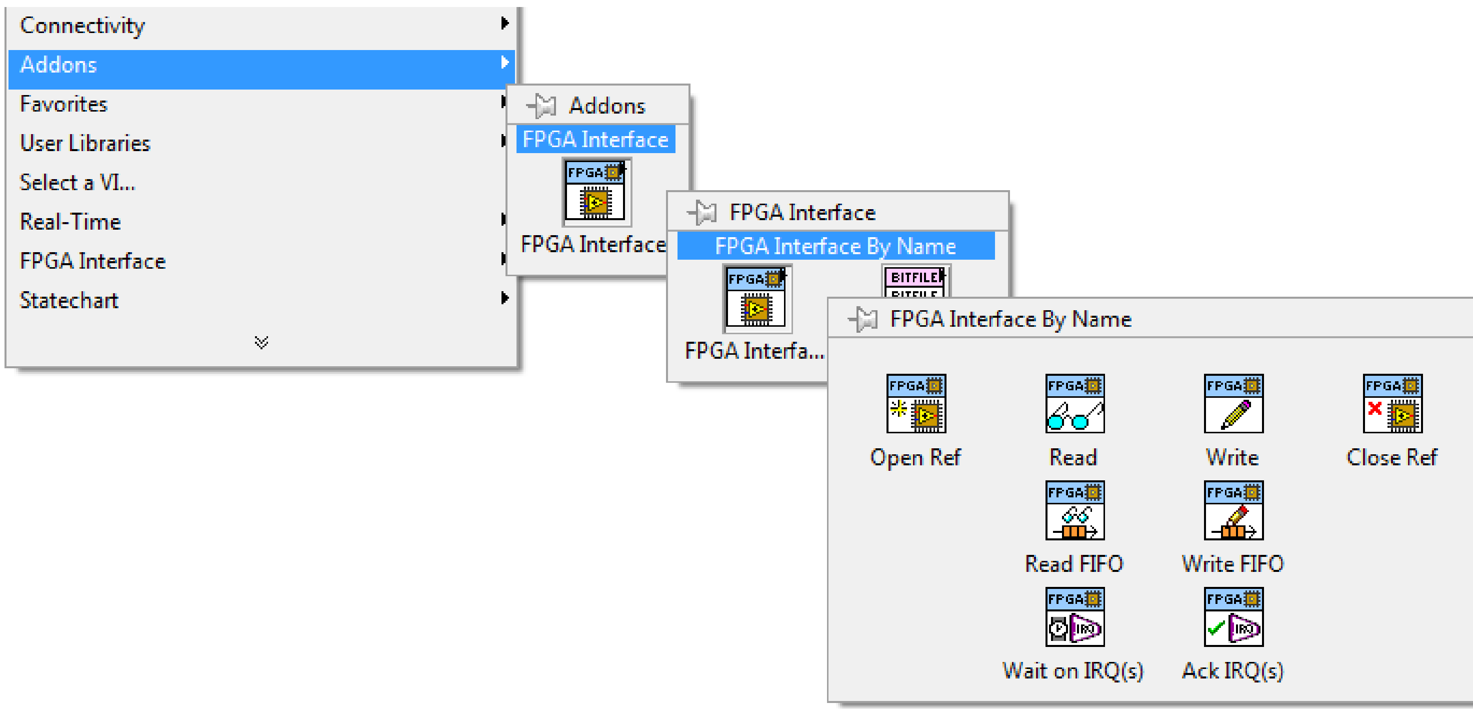


Figure 12: Fully Dynamic Interface Palette

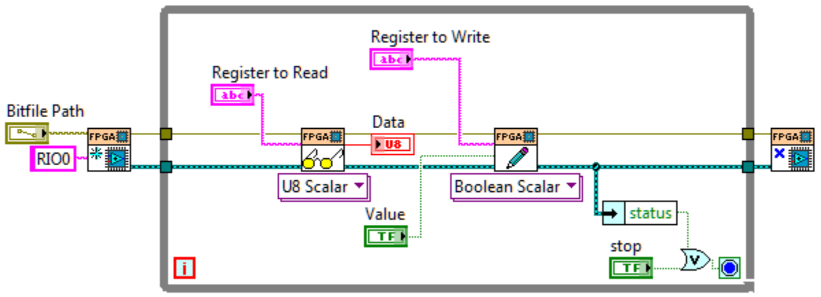


Figure 13: Using the Fully Dynamic Interface to Read and Write by String Name

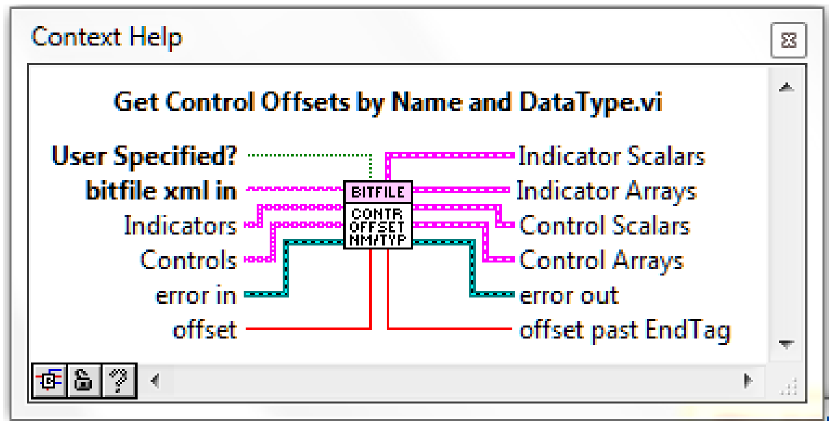


Figure 14: One of the many Bitfile Utility VIs to Query the Available Bitfile Items

Installation Instructions

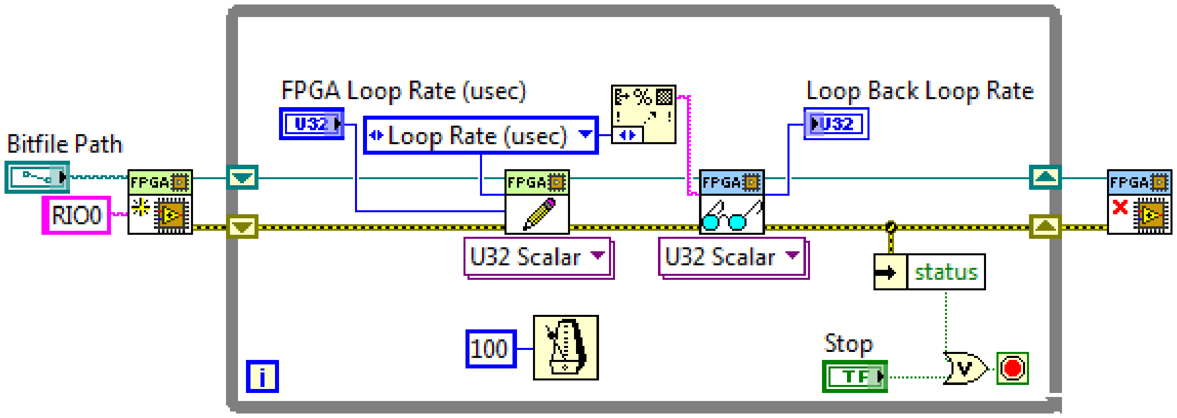
To use the LabVIEW FPGA Advanced Interface Tools, download and extract the attached zip file, open *LabVIEW FPGA Advanced Interface Tools Installer.vi* in the version of LabVIEW **with administrative privileges** (2010 or greater) that you'd like to install the interface to, accept the license agreement, and install.

Examples can be found at *labview\examples\FPGA Interface Advanced Tools*, and are searchable within NI Example Finder (search dynamic fpga interface) or browse to *FPGA_Hardware_Type>>FPGA Fundamentals>>Host Synchronization*.

Supported Platforms

1. Windows
2. Real-Time (PharLap and VxWorks)
3. Cross platform support for Linux and Mac OS X (see known issues below and contact local sales resource if support for these platforms is needed)

Known Issues

1. Linux support is not currently working because the driver calls do not correctly find the shared library.
2. Mac OS X support is untested and probably not working.
3. Firmware Independent Interfaces cannot read from FPGA controls or write to FPGA indicators. It can only write to controls and read from indicators. This limitation does not exist on the Fully Dynamic Interface. Since the Firmware Independent Interfaces inherit from the Fully Dynamic Interface, the workaround is to use the Fully Dynamic Interface read or write subVI for the appropriate data type and specify the name of the corresponding control or 

indicator.

Figure 15: Read Control Workaround

4. Firmware Independent Interface has an issue when multiple controls, indicators, or FIFOs have the same name. If all of the same named controls, indicators, or FIFOs are not all included in the interface, only the number included in the interface (whichever ones are found first) can be properly accessed. (i.e. if U32 and U16 controls are named the same and the interface is only built for the U32 control, it's possible that the U32 won't be accessible because the U16 control was found first according to the name. Since the U16 was not built into the interface it also is still not accessible. In this scenario, an error will be reported when attempt to write to the U32 control.)
5. You may need to run LabVIEW with administrator permissions for the installer to run without throwing a permissions error.

Version History

1/9/2014

- Fixed a bug interacting with DMA FIFOs inside bitfiles. This API would sometimes claim the DMA FIFOs were not present, but they were.

6/18/2013 #2

- Fixed a bug where U16 reads were only reading up to u8 values
- Slightly improved the run time performance of control and indicator reads and writes

6/18/2013

- New palette of VI to interfacing directly with the NiFPGA .dll / .out offering the lowest level access possible. These VIs require the "FPGA Session" u32 and the register or FIFO "Address" u32 and are much faster than the previously provided "read/write by name" VIs because you can look up your register addresses ahead of time instead of with every access.
- Register and FIFO address look up VIs are now public and available in the palette.
- FPGA Session Reference (u32) is now a public property, retrievable with a property node, of the advanced interface session.

2/25/2013

- Fixed the palette showing more items than intended

2/22/2013

- Installer will no longer error when running on LabVIEW versions higher than 2011
- Support for LabVIEW 2012 added

9/26/2011

- Error codes from this add-on should no longer be "unidentified" as the error code file is properlly installed into the <Program Files>\National Instruments\Shared\Errors\English directory.

Support and Contact

The LabVIEW FPGA Advanced Interface Tools is provided as open-source software. If it does not meet your exact specification, you are encouraged to modify the source code to meet your needs. It is not officially supported by National Instruments.

If you encounter a problem with these tools, or if you have suggestions for a future revision, please post to the support forum for this downlaod: LabVIEW FPGA Advanced Interface Tools Discussion.

These tools are deprecated and no longer maintained. For new projects, use the LabVIEW FPGA Advanced Session Resources instead.