j

Getting Started with the C FPGA Interface for RIO

Table of Contents

1 OVERVIEW 2

2 WHAT YOU NEED TO GET STARTED 3

3 SUPPORTED OS VERSIONS AND COMPILERS FOR DEVELOPMENT 3

3.1 OS Versions 3
3.2 Compilers 3

4 FILES INCLUDED IN THE C FPGA INTERFACE 4

4.1 capigen.exe 4
4.1.1 Generated .h file 4
4.1.2 Generated .lvbitx bitfile 4
4.2 Include Files 4
4.2.1 NiFpga.h 4
4.2.2 NiFpga.c 4
4.3 Libraries 5
4.4 Example 5
4.4.1 Using the Examples 5

5 DEPLOYING AND RUNNING APPLICATIONS 5

5.1 Windows 5
5.2 VxWorks 5
5.3 ETS 6
5.3.1 CVI 6
5.3.2 MSVC 6

6 WHERE TO GO FROM HERE 6

1 OVERVIEW

The C Interface to LabVIEW FPGA allows C /C++ applications to interact directly with compiled LabVIEW FPGA VIs on RIO devices without using LabVIEW. The functionality of this interface includes downloading a VI to a RIO target, performing DMA data transfers, waiting on and acknowledging interrupts, and reading and writing named controls and indicators using C function calls.

With this feature, users can program the real-time processor within CompactRIO and NI Single-Board RIO devices in C/C++ and interface that code to the LabVIEW FPGA code running on the FPGA of the RIO system. Additionally, this feature allows users to program the real-time processor of a PXI system or a processor on a PC running Windows in C/C++ and interface to the LabVIEW FPGA code running on a PXI/PCI R Series FPGA I/O board.

The current version of the C FPGA Interface *does not* support the following functions:

- Read/Write of fixed-point controls/indicators
- Read/Write of cluster controls/indicators
- Certain methods, such as Read TEDS

Note: NI-RIO software, version 3.0 or later (3.x), is required for the NI Labs release of this feature. Currently, the C FPGA Interface supports only those platforms and operating systems already supported by NI-RIO 3.x.

2 WHAT YOU NEED TO GET STARTED

LabVIEW 8.6.0 or later
LabVIEW FPGA 8.6.0 or later
NI-RIO 3.x
Any RIO Device
Supported Operating System
Supported C/C++ Compiler
C FPGA Interface for RIO

3 SUPPORTED OS VERSIONS AND COMPILERS FOR DEVELOPMENT

3.1 OS Versions

Windows Vista Business (32-bit and 64-bit versions)
Windows XP Professional
Windows 2000 Professional (Service Pack 3 or higher)

3.2 Compilers

For developing applications to run in Windows or in Pharlap ETS:
NI LabWindows/CVI (9.0 or later) or Microsoft Visual C++ (2003 or later)

For developing applications to run in VxWorks:
Wind River Workbench (2.3 or later) or GNU Tool Chain for VxWorks (available for download from NI)

4 FILES INCLUDED IN THE C FPGA INTERFACE

After you run the installer, you should have the following folders and files in \Program Files\National Instruments\NI Labs C FPGA Interface\.

4.1 capigen.exe

This application generates the files needed to build and run a C FPGA Interface application. Point it to an input bitfile (.lvbit) created with LabVIEW FPGA 8.6 and optionally a different output directory. Additionally, you can also override the prefix that will be used in the generated files. The application will generate a C header file (.h) and a converted bitfile (.lvbit**x**). Your application will need the .h file to build and the .lvbitx file to run.

This application can be run from the command-line as well, as follows:

```
capigen.exe <lvbit-file> [<output-directory> [<prefix-override>]]
```

This application requires the LabVIEW Run-Time Engine 8.6 to be installed.

4.1.1 Generated .h file

This is a C header file that contains all the constants required by function calls in your application. For example, `NiFpga_Open` requires the name of the generated .bin file, provided by the `Bitfile` constant; and a digital signature of the FPGA bitstream in the .bin file, provided by the `Signature` constant. The other constants represent register offsets of each control and indicator in your VI, and other necessary information.

4.1.2 Generated .lvbitx bitfile

This is a converted version of the original .lvbit bitfile. `NiFpga_Open` must find this file in order to ensure the bitstream is downloaded to the FPGA.

4.2 Include Files

4.2.1 NiFpga.h

This is a C-compatible header file. It declares all the types, constants, and functions needed to write an application. Most of these functions are defined in `NiFpga.c`.

4.2.2 NiFpga.c

This is a C source file that you must include in your application. It defines all the functions your application can call. `NiFpga.c` loads and unloads the NiFpga.* library at runtime, and forwards function calls to that library.

4.3 Libraries

Libraries are organized in folders by target operating system: Windows, VxWorks, and PharLap ETS. The libraries are named `NiFpga.dll` or `NiFpga.out`.

4.4 Example

The Example folder contains the LabVIEW Project source code for the example, and subfolders containing examples generated for certain supported operating systems and compilers. Each leaf directory contains example source code and project files specific to the given compiler; the generated .h and .bin files; and the `NiFpga.*` library for the given operating system.

It is recommended to make a copy of any example files before modifying them. If you wish to revert changes made to an installed example, you can uninstall NI Labs C FPGA Interface from Add or Remove Programs, delete your modified files, and reinstall.

4.4.1 Using the Examples

For your target operating system and compiler, find the associated project file and double-click it. For CVI, this is a .cws file. For Visual C++ this is a .sln file. For Wind River Workbench, you must import the entire directory using **File>>Import…>>Existing Project**.

The application itself is written in `Example.cpp` or `Example.c`, depending upon the compiler. All the examples are written to test the temperature of the FPGA device, and then wait while either toggling the FPGA LED (on CompactRIO devices) or Connector0/DO0 (on R Series devices).

Please note that the generated .bin files assume a specific set of hardware. In order to use the example on your hardware, launch the .lvproj file and recompile the correct FPGA VI for your device by right-clicking and selecting **Compile**. Next, use `Generate C FPGA Interface.vi` to regenerate the .h and .bin files into the appropriate directory.

5 DEPLOYING AND RUNNING APPLICATIONS

5.1 Windows

In Windows, you can run examples from within your development environment. Additionally, you can run the examples on remote targets by changing the resource passed into `NiFpga_Open` from a local resource ("RIO0") to a remote resource ("rio://<*hostnameOrIpAddress*>/RIO0").

5.2 VxWorks

If you are developing for a VxWorks system, you must set up the system to run your application on startup. FTP the build binary (`Example.out`), the support library (`NiFpga.out`), and the .bin file to the root directory of the target. Then modify `ni-rt.ini` by appending to `StartupDLLs` under the [LVRT] section. For example, you would change “`StartupDLLs=<stuff>;`” to “`StartupDLLs=<stuff>;Example.out;`”. Next, add this line under the [DEPENDENCIES] section: “NiFpga.out=NiRioSrv.out;”. Now your application will run every time the system starts up. Note that Scan Interface mode is not supported on VxWorks targets.

5.3 ETS

5.3.1 CVI

The CVI example assumes you have the LabWindows/CVI Real-Time Module installed. Select **Run>>Select Execution Target for Debugging>>New Execution Target**… to specify your target system. Then select **Run>>Manage Files on Real-Time Execution Target…** to add NiFpga.dll and the .bin file. You can then run the application by selecting **Run>>Debug Project**. Please see the CVI documentation for more information about running and deploying applications.

If you do not have the Real-Time Module, you can still create DLLs and deploy them in the same manner as in the MSVC section below.

5.3.2 MSVC

If you are developing for an ETS system, you must set up the system to run your application on startup. FTP the build binary (`Example.dll`), the support library (`NiFpga.dll`), and the .bin file to the root directory of the target. Then modify `ni-rt.ini` by appending to `StartupDLLs` under the `[LVRT]` section. For example, you would change “`StartupDLLs=<stuff>;`” to “`StartupDLLs=<stuff>;Example.dll;`”. Now your application will run every time the system starts up. Note that your application must reference your .bin file using an absolute path (“C:\\*.bin”) instead of a relative path.

6 WHERE TO GO FROM HERE

Refer to the *API Reference* document for explanations and descriptions of errors, types, and functions in the C FPGA Interface.

For any NiFpga_Status code returned, you can look up an error string by following the directions found here:
http://digital.ni.com/public.nsf/allkb/C241CCB3206265B686256A140064D2CD

The C FPGA Interface is essentially a C version of the LabVIEW FPGA Interface, which installed with the LabVIEW FPGA module. Functions in the C FPGA Interface correspond to VIs, Read/Write Nodes, and Method Nodes on the FPGA Interface palette in LabVIEW. Referring to the *FPGA Interface* topic of the *LabVIEW Help* may help you understand how the functions work.

If you have any other questions or feedback, please see the appropriate forum at http://ni.com/labs.