Getting started with the FPGA demo bundle for Intel FPGA

Xillybus Ltd.

Version 3.1

1

Introduction

Xillybus is a DMA-based end-to-end solution for data transport between an FPGA and a host that runs Linux or Microsoft Windows. It offers a simple and intuitive interface to the designer of the FPGA logic as well as the programmer of the software.
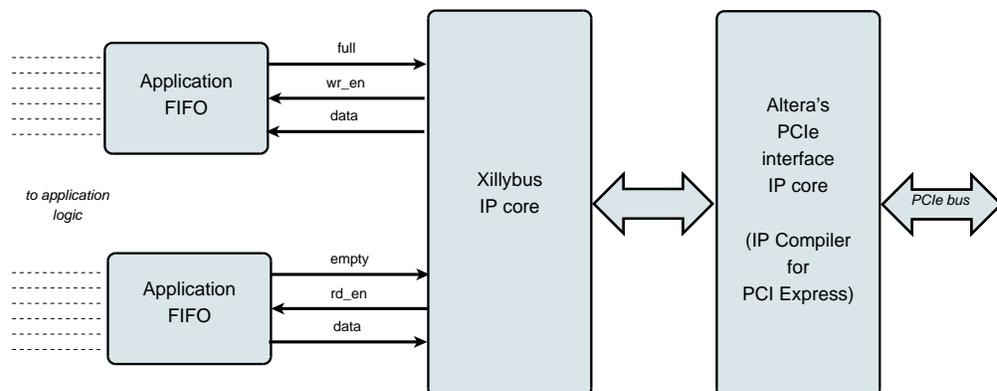
As shown above, the application logic on the FPGA only needs to interact with standard FIFOs.

For example, writing data to the lower FIFO in the diagram makes the Xillybus IP core sense that data is available for transmission in the FIFO's other end. Soon, the IP core reads the data from the FIFO and sends it to the host, making it readable by the userspace software. The data transport mechanism is transparent to the application logic in the FPGA, which merely interacts with the FIFO.

On its other side, the Xillybus IP core implements the data flow utilizing PCI Express' Transport Layer level, generating and receiving TLP packets. For the lower layers, it relies on Intel's official PCIe core, which is part of the development tools, and requires no additional license (even when using the Web / Lite Edition of Quartus).

The application on the computer interacts with device files that behave like named pipes. The Xillybus IP core and driver transport data efficiently and intuitively between the FIFOs in the FPGAs and their related device files on the host.

With XillyUSB, an MGT transceiver is used to implement an USB 3.0 interface, which is used for data transport instead of the PCIe interface mentioned above.

The IP core is built instantly per customer's spec, using an online web application. The number of streams, their direction and other attributes are defined by customer to achieve an optimal balance between bandwidth performance, synchronization, and simplicity of design. After going through the preparation steps with the demo bundle, as described in this guide, it's recommended to build and download your custom IP core at http://xillybus.com/custom-ip-factory.

This guide explains how to rapidly set up the FPGA with a Xillybus IP core, which can be attached to user-supplied data sources and data consumers, for real application scenario testing. The IP core is included in a demo bundle, which can be downloaded at the website.

Despite its name, the demo bundle is not a demonstration kit, but a fully functional starter design, which can perform useful tasks as is.

For those who are curious, a brief explanation on how Xillybus is implemented can be found in Appendix A of either Xillybus host application programming guide for Linux or Xillybus host application programming guide for Windows.

2

Prerequisites

2.1 Hardware

Xillybus relies on the Intel's hardware IP block for PCI Express, and is hence available for any Intel FPGA device having this component. Among the FPGA families supported:

- Arria II GX/GZ

- Cyclone IV GX

- HardCopy IV GX

- Stratix IV GX

- Arria V GX/GT/SX/ST

- Cyclone V GX/GT/SX/ST

- Stratix V GS/GX/GT

- Arria 10 GX/GT/SX

- Cyclone 10 GX

- Stratix 10 with H-tile or L-tile

XillyUSB is supported only with Cyclone 10 GX (the LP family is not supported).

The Xillybus FPGA demo bundle is packaged to work with several boards and devices, as listed on the download pages (see section 2.2 below).

Owners of other boards may run a demo bundle on their own hardware after making the necessary changes in pin placements and verifying that the MGT's reference clock is handled properly. This should be straightforward to any fairly experienced FPGA engineer. More about this in section 4.4.

2.2 FPGA project

The Xillybus demo bundle is available for download at Xillybus site's download pages. For the PCIe-based cores:

http://xillybus.com/pcie-download

And for XillyUSB:

http://xillybus.com/usb-download

The demo bundle includes a specific configuration of the Xillybus IP core, which is intended for simple tests. Therefore, it has a relatively poor performance for certain applications.

Custom IP cores can be configured, automatically built and downloaded using the IP Core Factory web application. Please visit http://xillybus.com/custom-ip-factory for using this tool.

Any downloaded bundle, including the Xillybus IP core, is free for use, as long as this use reasonably matches the term "evaluation". This includes incorporating the core in end-user designs, running real-life data and field testing. There is no limitation on how the core is used, as long as the sole purpose of this use is to evaluate its capabilities and fitness for a certain application.

2.3 Development software

The recommended tool for the implementation of Xillybus' demo bundle (as well as other designs involving Xillybus) is listed below, depending on the FPGA's family.

Xillybus for PCIe:

- For FPGAs of series-IV and Arria II: Quartus 12.0 and later.

- For Arria 10 and Cyclone 10: Quartus Prime 17.1 and later. Both the Standard and Pro editions are supported.

- For Stratix 10: Quartus Pro 19.2 and later.

- All other FPGA families: Quartus II, version 15.0 and later.

XillyUSB:

- For Cyclone 10: Quartus Pro 17.1 and later should be used.

This software can be downloaded directly from Intel's website (https://www.intel.com).

Note that Web / Lite Editions of Quartus, which are available at no cost, support several FPGA families, in particular Cyclone devices.

The implementation of Xillybus relies on some IP cores that are supplied by Quartus. All editions of Quartus cover these IP cores, without any need for additional licensing.

2.4 Experience with FPGA design

When the design is intended for a board that appears in the list of demo bundles, no previous experience with FPGA design is necessary to have the demo bundle working on the FPGA. When other boards are used, it's required to have some knowledge with using Intel FPGA's tools, in particular defining pin placements and clocks.

To make the most of the demo bundle, a good understanding of logic design techniques, as well as mastering an HDL language (Verilog or VHDL) are necessary. Nevertheless, the Xillybus demo bundle is a good starting point for learning these, as it presents a simple starter design to experiment with.

3

The implementation of the demo bundle

3.1   Overview

There are three possible methods for the implementation of Xillybus' demo bundle, and obtaining a bit stream file (SOF):

- Using the project files in the bundle as they are. This is the simplest way, and is suitable when working with boards that appear in the list of demo bundles.

- Modifying the files to match a different FPGA. This is suitable when working with other boards, and/or other FPGAs. More information about this in paragraph 4.4.

- Setting up the Quartus projects from scratch. Possibly necessary when integrating the demo bundle with existing application logic. Further details in paragraph 4.3.

In the remainder of this section, the first work procedure is detailed, which is the simplest and most commonly chosen one. The other two work procedures are based upon the first one, with differences that are detailed in the paragraphs given above.

IMPORTANT:

The evaluation bundle is configured for simplicity rather than performance. Significantly better results can be achieved for applications requiring a sustained and continuous data flow, in particular for high-bandwidth cases. For these scenarios, a custom IP core is easily built and downloaded with the web application.

3.2   File outline

The bundle consists of five directories:

- core – The Xillybus IP core is stored here

- instantiation templates – Contains the instantiation templates for the core (in Verilog and VHDL)

- verilog – Contains the project file for the demo bundle and the sources in Verilog (in the 'src' subdirectory)

- vhdl – Contains the project file for the demo bundle and the sources in VHDL (in the 'src' subdirectory)

- pcie_core (PCIe bundles only) – Intel's PCIe IP core is built here before building the rest of the bundle.

- quartus-essentials (XillyUSB bundles only) – Various files that are common to the Verilog and VHDL projects.

Note that each demo bundle is intended for a specific board, as listed at the site's web page from which the demo bundle was downloaded. If another board is used, or if certain configuration resistors have been added or removed from the board, the constraints file must be edited accordingly.

Also note that the vhdl directory contains Verilog files, but none of these should need significant changes.

The interface between Xillybus' IP core and the application logic takes place in the xillydemo.v file or xillydemo.vhd file (in the respective 'src' subdirectories). This is the file to edit in order to try Xillybus with your own data.

3.3   Building the demo bundle

3.3.1   Opening the project

Depending on your preference, double-click the 'xillydemo.qpf' file in either the 'verilog' or 'vhdl' subdirectory. Quartus will launch and open the project with the correct settings.

If a series-V (e.g. Cyclone V) or series-10 (e.g. Arria 10) FPGA is used, continue with paragraph 3.3.3.

Otherwise, build the PCIe block as described next.

3.3.2   Building the PCIe block (Arria II and series-IV FPGAs)

> **IMPORTANT:**
>
> *Note that this paragraph does not relate to series-V FPGAs and later.*

If a Quartus version later than 12.0 is used, the megafunction variation file needs manual editing. Otherwise, the MegaWizard Plug-In Manager will refuse to accept this file.

If editing is required, open the file in the pcie_core/ directory (e.g. pcie_core/pcie_s4_4x.v) with a text editor. Replace all the occurences of the older Quartus version with the one used. Just change the version number (e.g. replace 12.0 with 15.0).

Typically, lines like or similar to the following need modification:

```
// megafunction wizard: %IP Compiler for PCI Express v12.0%
```

and

```
// Retrieval info: <MEGACORE title="IP Compiler for PCI Express"  version="12.0"
```

and possibly also

```
// Generated using ACDS version 12.0 178
```

Launch the MegaWizard Plug-In Manager from within Quartus:

- Quartus 14 and later: Open a Command Prompt Window (or a terminal in Linux). Make sure that Quartus' utilities are in the execution path (typically /some/path/quartus/bin/). Type "qmegawiz" to launch the MegaWizard Plug-In Manager.

- Quartus 13 and earlier: In the Tools menu, pick MegaWizard Plug-In Manager.

The following (or similar) window should appear:
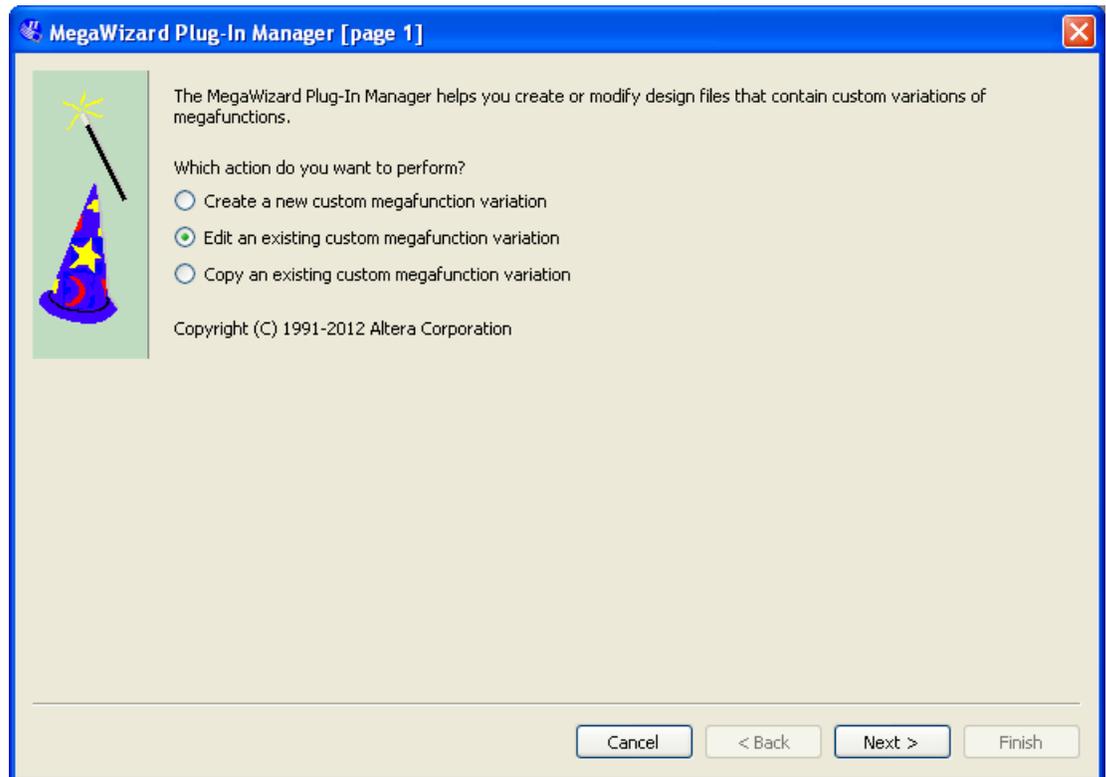
Choose "Edit an existing custom megafunction variation", and click Next.

Next, a window (not shown here) requests the custom megafunction variation file to edit. Pick the pcie_c4_1x.v (or similar) file in the pcie_core directory (only one suitable file will be present). Typically, navigating one directory up from the starting point reveals the directory to enter.

After a notice saying "Loading MegaWizard...", a window like this appears:
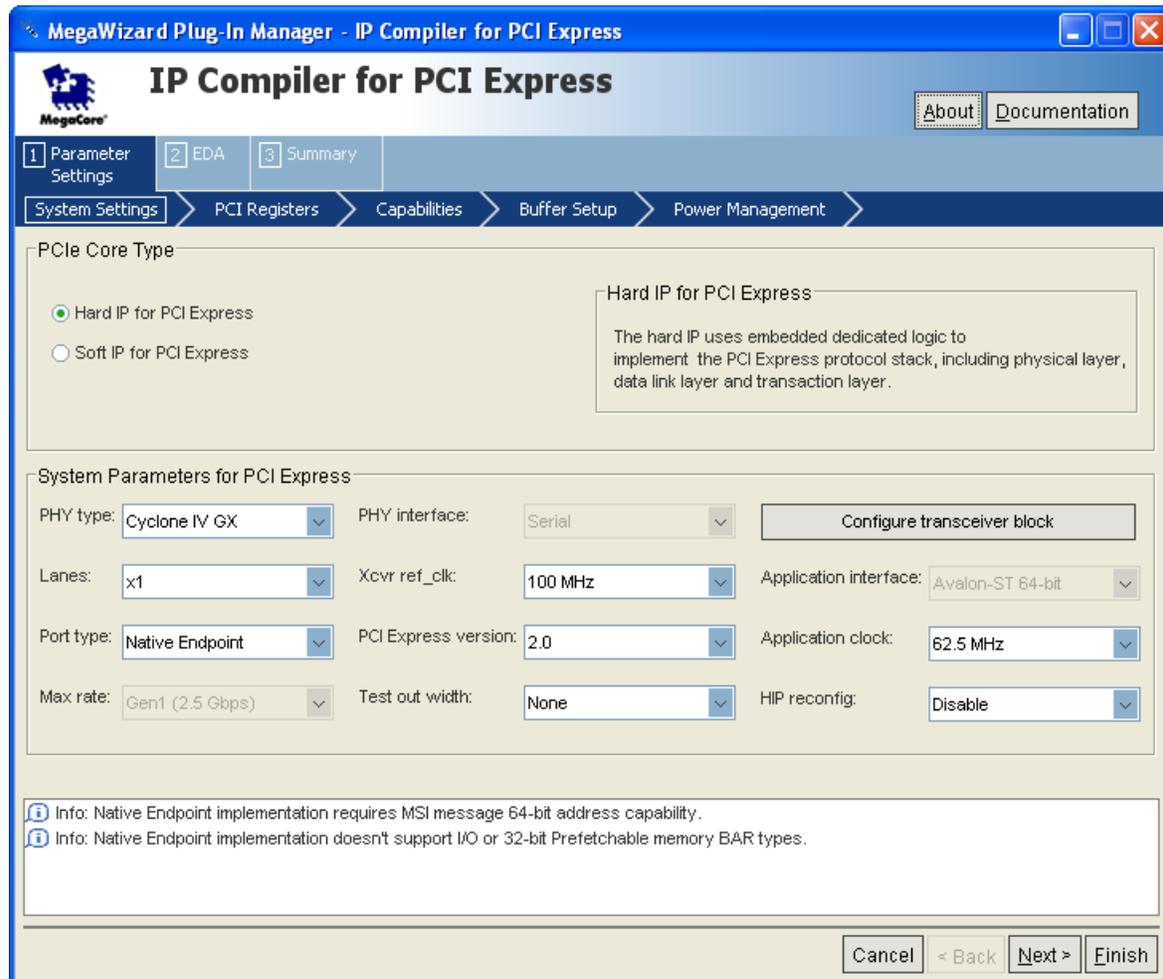
If the Megawizard refuses to open the file, saying "Specify a valid MegaWizard-generated variation file", there are a few possibilities for this problem:

- You're working with a recent FPGA (e.g. Cyclone V). In this case there's no need to build the PCIe block at all.

- The variation file wasn't edited properly to change the Quartus revision number to the current one. Please refer to the beginning of this section.

- If Megawizard rejects a file for being invalid, editing the file and attempting to load it again will not help. The Megawizard program must be exited and invoked again from the command prompt, or it will continue to reject the file, even if it has been edited properly.

The window that opens and the parameters it presents vary from one FPGA family to another (in particular, a title different from "IP Compiler for PCI Express" is fine).

No changes should be made. Just click "Finish". A window like the following shows the progress (its final state is shown here):
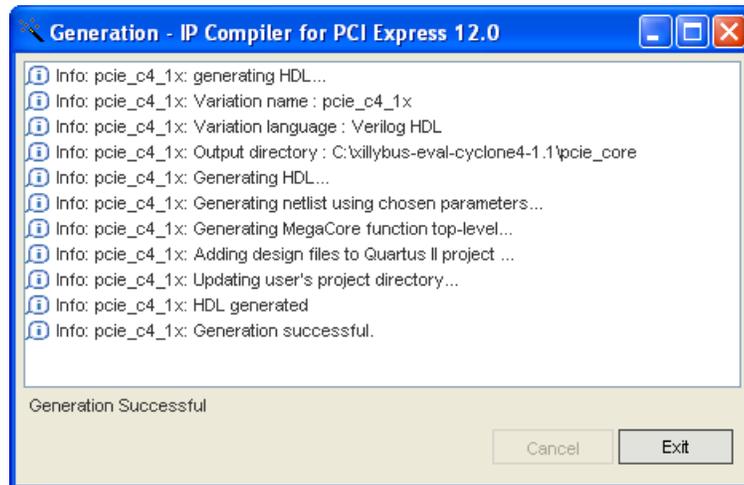
Once finished, click Exit. The following window *may* appear immediately afterwards, offering to add the Quartus IP (QIP) file to the project.
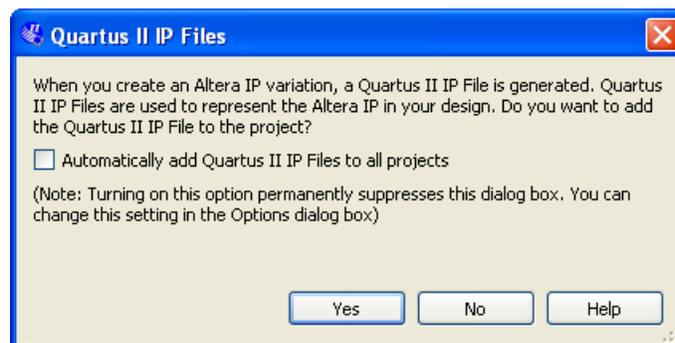
The preferred answer is "No", since adding the QIP file to the project indirectly adds an unnecessary SDC file, containing constraints which are all ignored anyhow. Even though it's harmless having this QIP file in the project, it causes a lot of warnings during the compilation, typically two warnings for each ignored constraint in the SDC file.

3.3.3 Compilation of the design files

In Quartus' main window, click "Compile Design" to create the FPGA bitstream file. Note that on some Quartus revisions, the default procedure for compilation may be set to "Rapid Recompile", which allows only "Rapid Recompile" in the task pane. If this is the case, change it to "Compilation", and proceed.
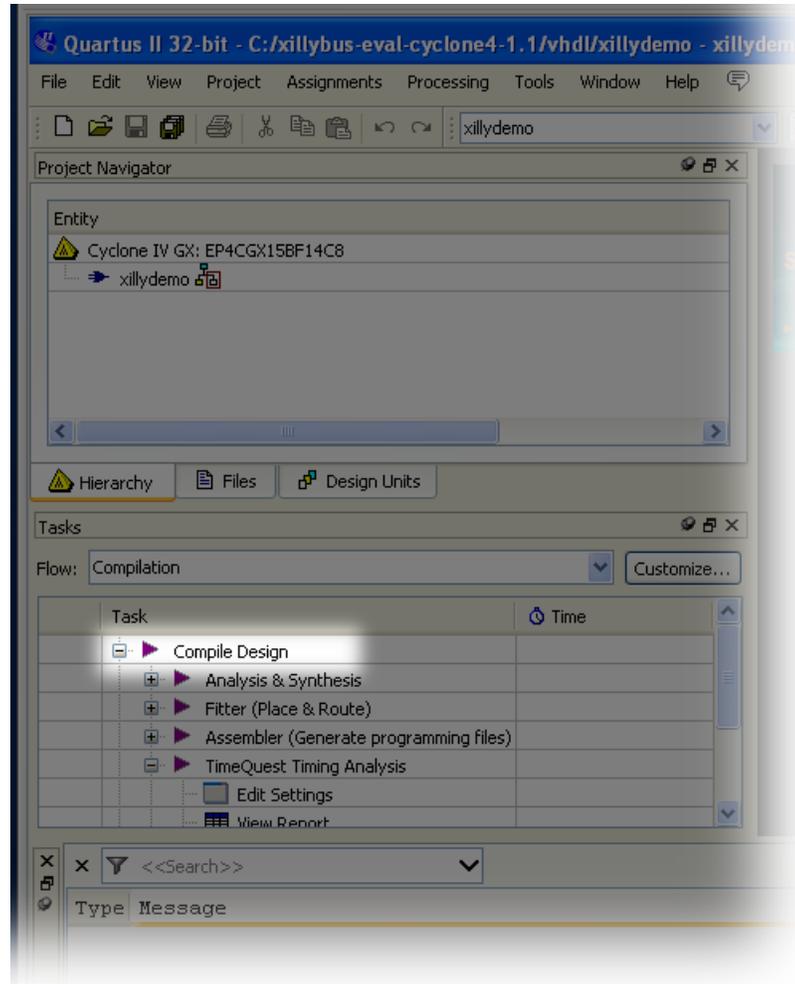
The process produces 20-50 warnings (depending on whether the QIP file was included in the project), but should end with a dialog box saying "Full Compilation was successful".

It's mandatory to verify that no errors nor Critical Warnings were generated (the tabs

at the bottom of the main Quartus window indicate the number of messages for each type).

At the end of the process, the programming file can be found as xillydemo.sof.

3.4 Programming the FPGA

In early development stages, it's recommended to load the FPGA via JTAG. This is usually done with an USB Blaster / Intel FPGA Download Cable (on-board or external). Please refer to your board's instructions on how to load the FPGA via JTAG.

For XillyUSB projects, the FPGA can be loaded and reloaded at any time, even while the USB interface is connected to a working computer.

With PCIe projects, the FPGA must be loaded with the bitfile before the computer is powered up: The computer expects the PCIe peripheral to be in a proper state when it powers up, and may not tolerate any surprises afterwards.

Therefore, do **not** reload the FPGA as long as the host is running. Even though the PCIe specification requires support for hotplugging, motherboards don't normally expect a PCIe card to disappear and then reappear. Accordingly, some motherboards may not respond correctly. Nevertheless, reloading of the FPGA, while the operating system is running, works on *some* motherboards.

Xillybus' driver is designed to respond sanely to hotplugging, however there is nothing to assure the computer's general stability. This is dicussed on this page:

http://xillybus.com/doc/hot-reconfiguration

If the FPGA powers up and is loaded from a flash memory along with powering up the computer, it's essential to ensure that the FPGA is loaded quickly enough, so the PCIe device is present when the BIOS scans the bus.

4

Modifications

4.1 Integration with custom logic

The Xillybus demo bundle is constructed for easy integration with application logic. The place for connecting data is the xillydemo.v or xillydemo.vhd file (depending on the preferred language). All other HDL files in the bundle can be ignored for the purpose of using the Xillybus IP core for transporting data between the host (Linux or Windows) and the FPGA.

Additional HDL files with custom logic designs may be added to the project that was prepared as described in paragraph 3.3, and then rebuilt by clicking "Compile Design". There is no need to repeat the other steps of the initial deployment, so the development cycle for logic is fairly quick and simple.

When attaching the Xillybus IP core to custom application logic, it is warmly recommended to interact with the Xillybus IP core only through FIFOs, and not attempt to mimic their behavior with logic, at least not in the first stage.

An exception for this is when connecting memories or register arrays to Xillybus, in which case the example that is shown in the xillydemo module should be followed.

In the xillydemo module, FIFOs are used to perform a data loopback. In other words, the data that arrives from the host is sent back to it. Both of the FIFO's sides are connected to the Xillybus IP core, so the core is both the source of the data and the consumer of the data.

In a real-life usage scenario, only one of the FIFO's sides is connected to the Xillybus IP core. The FIFO's other side is connected to application logic, which supplies or consumes data.

The FIFOs that are used in the xillydemo module work with only one common clock

for both sides (scfifo), as both sides are driven by Xillybus' main clock. In a real-life application, it may be desirable to replace them with FIFOs that have separate clocks for reading and writing. This allows driving the data sources and data consumers with a clock other than bus_clk. By doing this, the FIFOs serve not just as mediators, but also for proper clock domain crossing.

Note that the Xillybus IP core expects a *plain* FIFO interface, (as opposed to "show-ahead") for streams from the FPGA to the host.

The following documents are related to integrating custom logic:

- The API for logic design: Xillybus FPGA designer's guide

- Getting started with Xillybus on a Linux host

- Getting started with Xillybus on a Windows host

- Xillybus host application programming guide for Linux

- Xillybus host application programming guide for Windows

- The guide to defining a custom Xillybus IP core

4.2   Making changes in Intel's PCIe IP / Megafunction

Unless absolutely necessary, changes in the configuration of the Intel IP compiler for PCI Express Megafunction should not be made.

In particular, there is a high senstivity to the allocation of receive buffers. The Xillybus IP core relies upon the sizes of those buffers according to the numbers that are given by the IP Compiler / MegaWizard in the demo bundle. If the PCIe core allocates less buffer space than expected by the Xillybus IP core, sporadic data errors may occur in the communication from the host to FPGA. This can also cause a complete stall of the communication in this direction. Such problem would be the result of packets arriving to the FPGA that cause an overflow of these buffers.

If any change is needed in the IP / Megafunction, please seek assistance through the email address published at Xillybus' web site.

4.3   Inclusion in a custom project

If desired, it's possible to include the Xillybus IP core in an existing Quartus project, or create a new project from scratch. This section relates to Xillybus for PCIe, however a similar procedure applies for XillyUSB.

If the project doesn't exist already, start a new project, and set it up as based upon your preferred HDL language and intended FPGA.

To include the Xillybus IP core in the project:

- Copy the pcie_c4_1x.v file (or alike) from the pcie_core/ subdirectory to a directory related to the project.

- For Cyclone V, Arria V, Stratix V, and series-10 FPGAs, also copy pcie_core/pcie_reconfig.qsys (or pcie_core/pcie_ip.ip) into the same directory.

- Follow the procedure that is outlined in 3.3.2 if the chosen FPGA requires that, in order to build Intel's PCIe block.

- These files should be added (possibly a copy of them) for Cyclone IV:

  - pcie_c4_1x.v
  - pcie_c4_1x_core.v
  - pcie_c4_1x_serdes.v
  - pcie_c4_1x_examples/chaining_dma/pcie_c4_1x_rs_hip.v
  - ip_compiler_for_pci_express-library/altpcie_hip_pipen1b.v
  - ip_compiler_for_pci_express-library/altpcie_reconfig_3cgx.v
  - ip_compiler_for_pci_express-library/altpcie_rs_serdes.v

  For Stratix IV and Arria II, these files should be added:

  - pcie_s4_4x.v
  - pcie_s4_4x_core.v
  - pcie_s4_4x_serdes.v
  - pcie_s4_4x_examples/chaining_dma/pcie_s4_4x_rs_hip.v
  - ip_compiler_for_pci_express-library/altpcie_hip_pipen1b.v
  - ip_compiler_for_pci_express-library/altpcie_reconfig_4sgx.v
  - ip_compiler_for_pci_express-library/altpcie_rs_serdes.v

  For series-V FPGAs, there is no need to copy similar files, as they are generated and included by Qsys.

- Copy the two HDL files, xillybus.v and xillydemo.v(hd), in one of the two src/ subdirectories (depending on your language preference) and add them to the project.

- Adopt the constraints in the SDC file in either of the src/ directories (they may need slight modifications to match names of top level signals in the project).

- Copy xillybus_core.qxp or xillybus_core.vqm from the core/ directory and add this file to the project as well.

- If the xillydemo module isn't the top level module of the projects, connect its ports to the top level.

- To attach the Xillybus IP core to custom application logic, edit the xillydemo module, replacing the existing application logic with the desired one.

4.4 Using other boards

4.4.1 General

When working with a board which doesn't appear in the list of demo bundles, some slight modifications in the bundle are necessary.

4.4.2 Device family

The demo bundle includes a QXP or VQM file, which is a synthesized file for a certain device family. It can be used with any device within this family.

4.4.3 Pin placements

Most purchased boards have their own example of an FPGA design, which shows how the PCIe interface is used on that board. It's often easiest to locate the relevant pin assignments in the intended board's QSF file, and modify the pins' names to those that are used in Xillybus' QSF file. Having done this, it's possible to replace the relevant rows in the QSF file that is used in Xillybus' project.

There are also four user_led wires, which have no functional significance. But if there are vacant LEDs on the board, it's recommended to connect them, as they supply some indications on the communication status. The user_led signal assumes active low logic (a logic '0' means LED is on).

4.4.4 PCIe only: Clocking

The xillydemo module requires some or all of these three clocks as inputs. The clocks that are needed are those that are inputs to the xillydemo module.

- pcie_refclk – Connected directly to the reference clock that is supplied by the host's motherboard. This clock should have a frequency of 100 MHz. This clock may not be active when the host is powered off, and in other situations when this is allowed by the PCIe specification. A clock cleaning circuit may be present between the motherboard's clock and the one that is connected to the FPGA, as required by the FPGA.

  If a different clock frequency is supplied to this input port (e.g. 125 MHz supplied by a clock cleaner), it's necessary to change the Xcvr_ref_clk parameter in the IP Compiler for PCI Express settings accordingly. This is done when invoking the relevant IP Compiler / MegaWizard plug-in manager, as shown in paragraph 3.3.

- clk_50 – Drives the gigabit transceiver's reconfig_clk, and must always be active. Therefore it must not depend on the motherboard's reference clock. The allowed frequency range of this clock depends on the FPGA family (e.g. 37.5 MHz to 50 MHz on Cyclone IV). The specifications for this clock can be found by searching for "reconfig_clk" in the device handbook for the relevant FPGA.

- clk_125 – Drives the always-active clock that is required by the transceiver. The frequency of this clock must be 125 MHz, and must not depend on the motherboard's reference clock.

IMPORTANT:

*pcie_refclk must **not** be driven by just some oscillator on the board. Any slight frequency difference with the host's clock leads to unreliable communication or failure to detect the FPGA as a PCIe device.*

In the demo bundle for Cyclone IV GX Transceiver Starter Board, the clk_50 and clk_125 inputs are driven directly by I/O pins that are connected to clock sources of 50 MHz and 125 MHz respectively on the board. In the absence of suitable clocks directly from the board, a PLL can be used to generate both, as described in section 7 of the IP Compiler for PCI Express User Guide, which is published by Intel.

When using a PLL, xillybus.v must be edited, connecting the reconfig_clk_locked signal to the PLL's lock indicator. This has already been done on the demo bundle for Stratix IV, because the DE4 board doesn't supply an always-active clock with a 125 MHz frequency.

An example of using a PLL is available among the files in the pcie_core subdirectory, after the PCI Compiler for PCI Express files are generated. This can be found

as pcie_c4_1x_example_chaining_pipen1b.v in the pcie_c4_1x_examples/chaining_dma directory (or similar files when working with a family other than Cyclone IV).

4.4.5 Working with XillyUSB

XillyUSB can be used on other boards that have an SFP+ interface. In this case it's just a matter of setting the design's constraints to use the MGT that is wired to the SFP+ connector.

The board should also supply a 125 MHz reference clock with low jitter for the MGT. Despite the requirement in the USB specification, Spread Spectrum Clocking (SSC) should *not* be enabled (if such option exists): The MGT doesn't lock properly on the received signal if an SSC reference clock is used.

For custom boards, it's recommended to refer to the sfp2usb module's schematics, as the pins of the SFP+ connector are connected directly to the FPGA's MGT. It is optional to swap the SSRX wires, as done on the sfp2usb module. This is recommended only if it simplifies the PCB design.

Swapping the SSTX wires is also possible, if desired. This requires editing the *_frontend.v file so that the polarity of the transmitted bits is reversed, and hence compensates for the wire pair swap. Note that there's a good chance that the USB connection will operate properly even without this edit, since the USB specification requires the link partners to work properly even with a polarity swap. It's however recommended to not rely on this.

5

Troubleshooting

5.1 Errors during implementation

Sometimes there are slight differences between releases of Intel's tools. This can result in failures to run the implementation for creating a SOF file.

If the problem isn't solved fairly quickly, please seek assistance through the email address that is given at Xillybus' web site. Please attach the output log of the process that failed, in particular around the first error that is reported by the tool. Also, if custom changes were made in the design (i.e. diversion from the demo bundle) please detail these changes. Also please state which version of the Quartus tools was used.

5.2 PCIe Hardware problems

Normally, the PCIe card is detected properly by the host's BIOS and/or operating system, and the host's driver launches successfully.

On most PC computers, the BIOS briefly displays a list of detected peripherals early in the boot process. When the Xillybus interface is detected successfully, a peripheral with vendor ID 1172 and device ID EBEB appears on the list.

As for the operating system's detection of the card, please refer to one of these two documents, whichever applies:

- Getting started with Xillybus on a Linux host

- Getting started with Xillybus on a Windows host

The failure to detect the card (or a failure in the computer's boot process) is not related to the Xillybus IP core, which relies on Intel's PCIe IP core for interfacing with the bus.

At first, it's recommended to verify the following:

- The bitstream was already loaded into the FPGA when the computer was powered on (or soon enough after it was powered on, in terms of the PCI-SIG specification).

- The pinouts of the PCIe wires, including the reference clock are correct (this can be verified in the fitter's report).

- The board supplies the correct reference clock to the FPGA.

If the problem isn't spotted immediately, it's recommended to attempt the sample project for PCIe that came with the board. This may reveal wrong jumper settings and possibly defective hardware.

If the card is detected with this sample, but not with Xillybus, it may be helpful to compare the pinouts of the two designs. If they are equal, the next step is comparing the attributes of Intel's PCIe cores, by invoking the relevant GUI tool with each.

The following configuration elements may need adjustment:

- The frequency of the reference clock.

- The base class and sub class (not likely, but some relatively old PC computers have failed the boot process if the class was considered unknown).

- Any other attribute that is configured differently, except for the base address register settings, vendor ID, device ID and interrupt settings, which should not be altered.

If the problem remains, please seek assistance through the email address that is given at Xillybus' web site.