Getting started with the FPGA demo bundle for Intel FPGA

Xillybus Ltd.

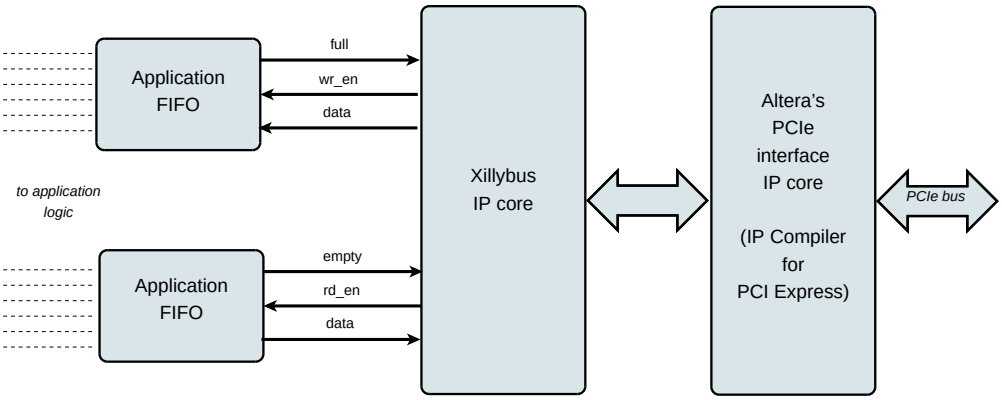
www.xillybus.com

Version 2.4

1

Introduction

Xillybus is a straightforward, intuitive, efficient DMA-based end-to-end turnkey solution for data transport between an FPGA and a host running Linux or Microsoft Windows.



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 Xillybus 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 TLPs. 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 IP core is built instantly per customer's spec, using an online web interface. It's

recommended to build and download your custom IP core at http://xillybus.com/custom-ip-factory after walking through the demo bundle flow described in this guide.

The number of streams, their direction and other attributes are defined by customer to achieve an optimal balance between bandwidth performance, synchronization, and design simplicity.

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

This guide explains how to rapidly set up the FPGA with a demo Xillybus IP core, which can be attached to user-supplied sources or sinks for real application scenario testing. The IP core is “demo” in the sense it’s not tailored to any specific application.

Nevertheless, the demo core allows creating a fully functional link with the host.

For the 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

The Xillybus FPGA demo bundle is packaged to work out of the box with several boards, as listed on the website:

http://xillybus.com/pcie-download

Owners of other boards may run the demo bundle on their own hardware after making the necessary changes in pin placements and setting up clocking 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 page:

http://xillybus.com/pcie-download

The demo bundle includes a specific configuration of the Xillybus IP core, having a relatively poor performance for certain applications, as it's intended for simple tests.

Custom IP cores can be configured, automatically built and downloaded using the IP Core Factory web interface. 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 implementing the Xillybus demo design (as well as other designs involving Xillybus) is Quartus II, version 15.0 (possibly the Web / Lite Edition when targeting Cyclone IV and V).

Later releases should work properly as well.

For Series-IV and Arria II FPGAs, Quartus 12.0 and later are adequate as well.

Arria 10 and Cyclone 10 designs should be implemented with Quartus Prime 17.1 and later. Both the Standard and Pro editions are supported.

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

All editions of Quartus cover the Intel-supplied IP cores necessary to implement Xillybus for PCIe, with no extra licensing required.

2.4 Experience with FPGA design

When targeting a board, which appears in the demo bundle list, no previous experience with FPGA design is necessary to have the demo bundle running on the FPGA. Targeting another board requires 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 tech-

niques, 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

Implementing the FPGA demo bundle

3.1 Overview

There are three flows for implementing the Xillybus demo bundle and obtaining a bit stream file (SOF) to program the FPGA with:

- Using the project files in the bundle as they are. This is the simplest way, and is suitable when targeting the boards that appear in the list of demo bundles.
- Modifying the files to match a different target. This is suitable when targeting other boards, and/or FPGA models. More information in 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 flow is detailed, which is the simplest and most commonly chosen one. The other two flows are based upon the first one, with differences 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. Custom IP cores are easily built and downloaded with the web interface.

3.2 File outline

The bundle consists of five directories:

- core – The binary of the Xillybus core is stored here
- instantiation template – Contains the instantiation template for the core in Verilog and VHDL
- verilog – Contains the project file for the demo and the sources in Verilog (in the 'src' subdirectory)
- vhdl – Contains the project file for the demo and the sources in VHDL (in the 'src' subdirectory)
- pcie_core – Intel's PCIe IP core is built here before building the rest of the bundle.

Note that both 'verilog' and 'vhdl' directories also contain the QSF file for the boards, for which the demo bundle is targeted. This file must be edited if another board is targeted, as it contains pin placements.

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

The interface with Xillybus takes place in the xillydemo.v or xillydemo.vhd files in the respective 'src' subdirectories. This is the file to edit in order to try Xillybus with your own data sources and sinks.

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 targeted, 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, or the MegaWizard Plug-In Manager will refuse to accept it.

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"
```

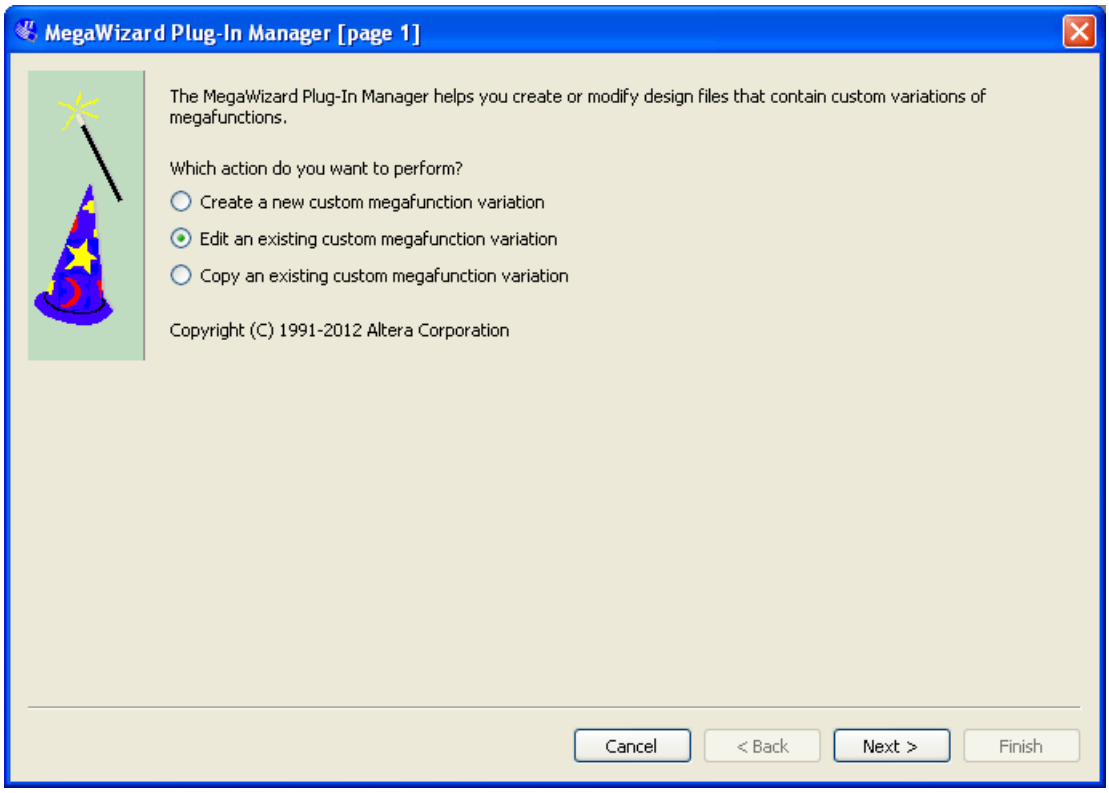
and possibly also

```
// 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.

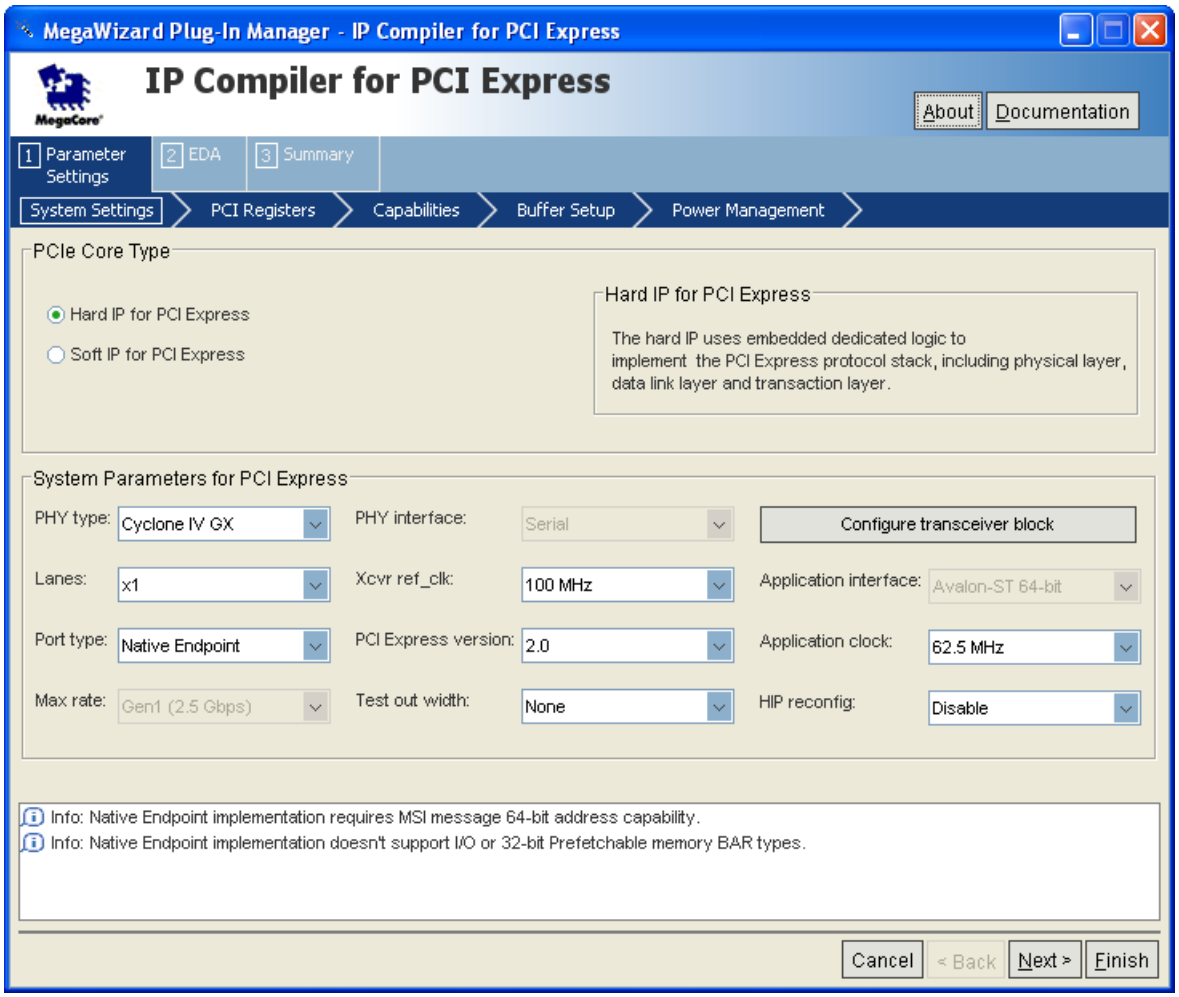
The following or similar windows should appear:



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:

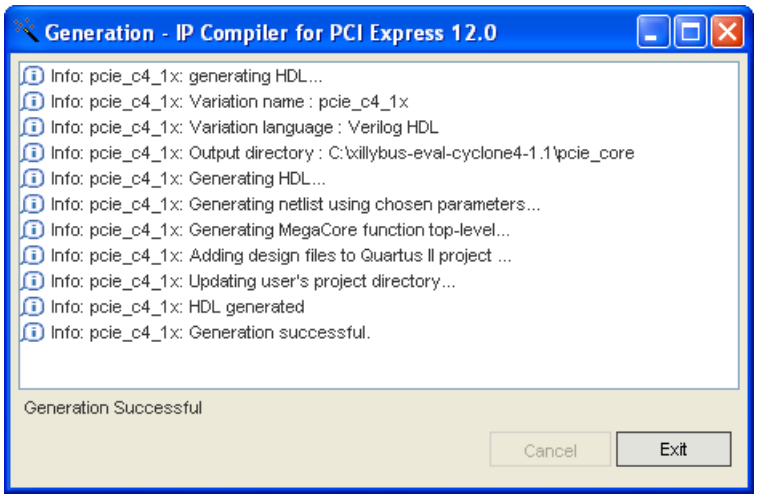


If the Megawizard refuses to open the file, saying “Specify a valid MegaWizard-generated variation file”, there are a few possibilities for this problem:

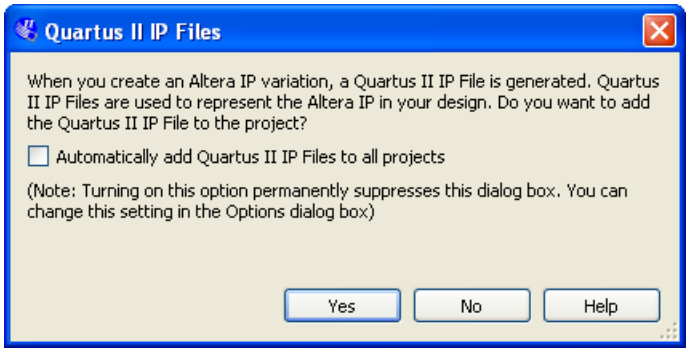
- You’re targeting a series-V FPGA (e.g. Cyclone V), in which case there’s no need to build the PCIe block at all.
- The variation file wasn’t edited properly to change the Quartus revision number to the current one. Please refer to the beginning of this section.
- If Megawizard rejects a file for being 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 its paramaters varies 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 windows like the following shows the progress (the final state is shown):



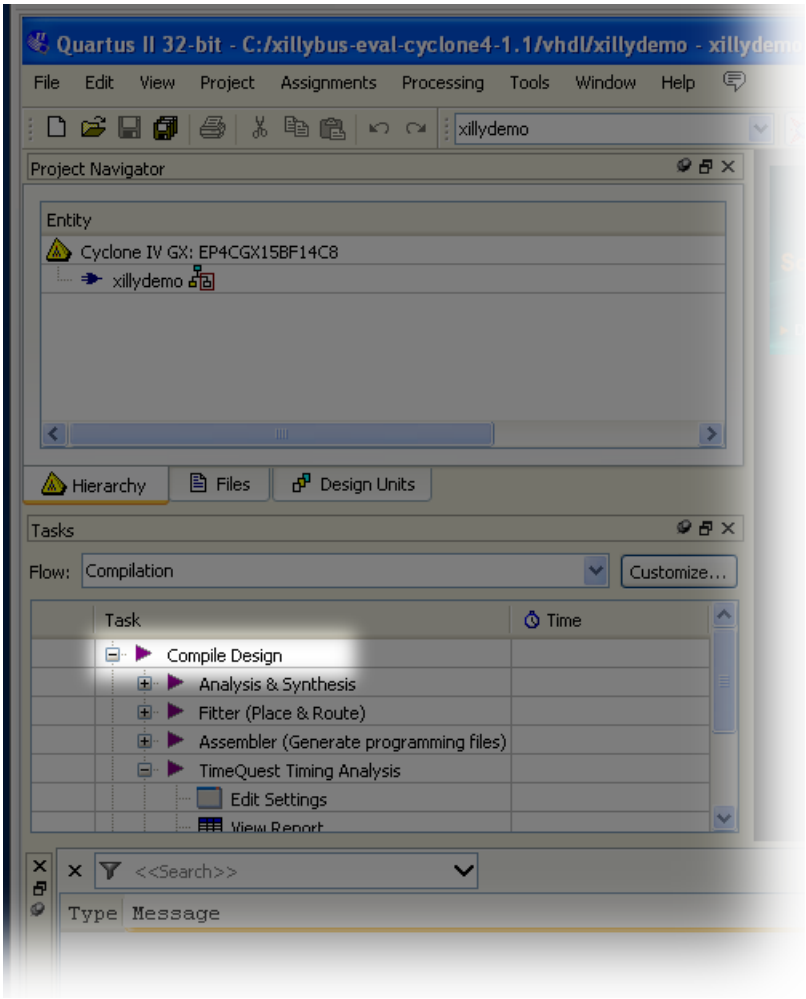
Once finished, click Exit. The following window *may* appear immediately afterwards, offering to add the Quartus IP (QIP) file to the project.



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 Compiling the design files

In Quartus' main window, click "Compile Design" to create the FPGA programming file. Note that on some Quartus revisions, the default flow may be set to "Rapid Recompile", which allows only "Rapid Recompile" in the task pane. If this is the case, change the flow 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 informing that the "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 counts for each message type).

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

3.4 Programming the FPGA

IMPORTANT:

The host computer expects the PCIe peripheral to be in proper state when it powers up, and does not tolerate any surprises afterwards. In other words, it's your responsibility to make sure that the FPGA is loaded quickly enough, if both the host and the FPGA power up at the same time.

In early development stages, it's recommended to load the FPGA via JTAG (typically with an on-board or external USB Blaster). Please refer to your board's user guide regarding how to do this. Load the FPGA, and only then power on the hosting computer.

Do **not** reload the FPGA as long as the host is running. Even though the Xillybus driver goes a long way to behave sanely if this happens, there is nothing to assure the general stability. A PCIe card disappearing and reappearing is not something a motherboard is supposed to cope with. Even though PCIe is hotpluggable in general, this is not the expected behavior on a motherboard's PCIe card slot.

4

Modifications

4.1 Integration with custom logic

The Xillybus demo bundle is set up for easy integration with application logic. The front end for connecting data sources and sinks is the xillydemo.v or xillydemo.vhd file (depending on the preferred language). All other HDL files in the bundle kit can be ignored for the purpose of using the Xillybus IP core as a transport of data between the Linux or Windows host and the FPGA.

Additional HDL files with custom logic designs may be added to the project handled 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 schema shown in the xillydemo module should be followed.

In the xillydemo module, FIFOs are used to loop back data arriving from the host back to it. Both of the FIFOs sides are connected to the Xillybus IP core, which makes the core function as its own data source and sink.

In a more useful setting, only one of the FIFO’s ends is connected to the Xillybus IP core, and the other end to an application data source or sink.

The FIFOs used in the xillydemo module accept only one common clock for both sides (scfifo), as both sides are driven Xillybus’ main clock. In a real-life application, it may be desirable to replace them with FIFOs having separate clocks for reading and

writing (dcfifo), allowing data sources and sinks to be driven by a clock other than the bus clock. 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 FPGA to host streams.

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 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 buffer space. The sizes of those buffers are hardcoded in the Xillybus IP core according to the numbers given by the MegaWizard. If the PCIe core allocates less buffer space than expected by the Xillybus IP core, sporadic data errors and possibly a complete stall of the host-to-FPGA communication may occur as a result of packets arriving to the FPGA overflowing its buffers.

If any change is needed in the 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.

If the project doesn't exist already, start a new project, and set it up as based upon your preferred HDL language and target 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 target 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 outlines in 3.3.2 if the targeted 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 target 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 demo application logic with the desired one.

4.4 Using other boards

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

4.4.1 Device family

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

4.4.2 Pin placements

Most purchased boards have their own FPGA bundle for demonstrating some PCIe functionality. It's often easiest to locate the relevant pin assignments in the target board's QSF file, modify the pins' names to those used in the Xillybus QSF file, and replace the respective rows with the ones taken from the target board's.

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).

4.4.3 Clocking

The xillydemo module expects some or all of these three clocks as inputs (as listed in the xillydemo module):

- pcie_refclk – Connected directly to the reference clock supplied by the host's motherboard, and should have a frequency of 100 MHz. This clock may not run when the host is powered off, and when otherwise allowed in the PCIe specification. A clock cleaning circuit may be present between the motherboard's clock

and the one connected to the FPGA, but some reference design boards don't apply a clock cleaner.

If another 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 MegaWizard plug-in manager, as shown in paragraph 3.3.

- clk_50 – Drives the gigabit transceiver's reconfiguration and offset cancellation clock (reconfig_clk) and should be free running. It must not depend on the motherboard's reference clock. The allowed frequency range is device family dependent (37.5 MHz to 50 MHz on Cyclone IV). Per-device specifications for this clock can be found by searching for "reconfig_clk" in the respective device handbook.
- clk_125 – Drives the free running clock that serves as input to the fixed clock of 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 at best.*

In the demo bundle for Cyclone IV GX Transceiver Starter Board, clk_50 and clk_125 are driven directly by I/O pins 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, 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, as the DE4 board doesn't supply a free-running 125 MHz clock.

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 targeting a family other than Cyclone IV).

5

Troubleshooting

5.1 Implementation errors

Slight differences between releases of Intel's tools sometimes result in failures to run the implementation chain for creating a SOF file.

If the problem isn't solved fairly quickly, please seek assistance through the email address given at the company's web site. Please attach the output log of the process that failed, in particular around the first error 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 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 displays a list of detected peripherals briefly when the computers boots. 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 failing to boot the computer) 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 that

- the FPGA was configured while the host computer was powered off (or soon enough after it was powered on, in terms of the PCI-SIG spec).
- the pinouts of the PCIe wires, including the reference clock are correct (this can be verified in the fitter's report)
- the board is configured to supply the expected reference clock

If the problem isn't spotted immediately, it's recommended to attempt the PCIe sample project 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 the Intel's PCIe cores, by invoking the MegaWizard 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 to boot if they failed to interpret the class)
- 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 given at the company's web site.