Getting started with the FPGA demo bundle for Xilinx

Xillybus Ltd.

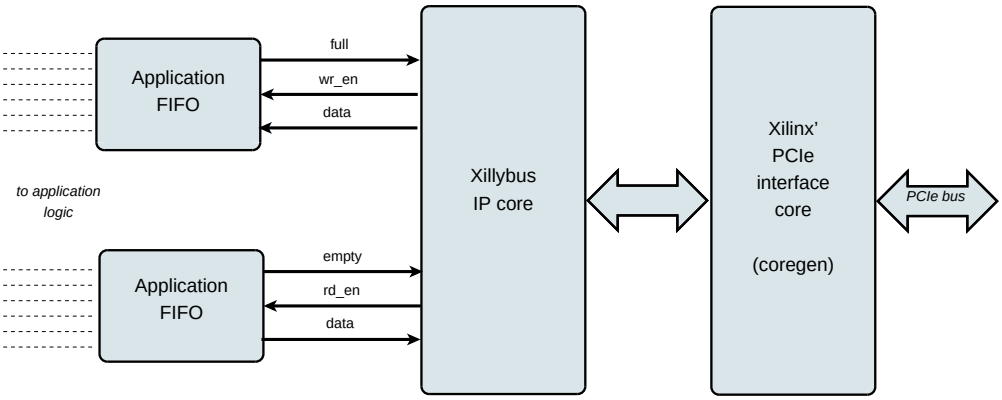
www.xillybus.com

Version 2.7

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 Xilinx' official PCIe core, which is part of the development tools, and requires no additional license (even when using the Webpack edition).

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

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

http://xillybus.com/pcie-download

Owners of other boards may run one of the demo bundles on their own hardware after making the necessary changes in pin placements and verifying that the PCIe reference clock is handled properly. This should be straightforward to any fairly experienced FPGA engineer. More about this in section 4.3.

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 listed below, depending on the target FPGA's family.

- When targeting Virtex-5 FPGAs, release Xilinx ISE 13.1 is preferred, see paragraph 4.3.3.
- For Spartan-6 and Virtex-6, Xilinx ISE 13.2 and later.
- For Kintex-7 and Virtex-7 with Gen2 interface (485T and non-XT/HT), Vivado 2014.1 and later is the preferred tool. Among the ISE revisions, release 14.2 and later is recommended.
- Virtex-7 with Gen3 interface (XT/HT except 485T) is preferably implemented with Vivado 2014.1 and later. If ISE is chosen, revision 14.6 and later is required.
- For Artix-7, Vivado 2014.1 and later should be used. ISE 14.6 and later is fine as well.
- For Kintex / Virtex Ultrascale, Vivado 2015.2 and later should be used. No ISE revision supports these devices.
- Ultrascale+ targets require Vivado 2016.4 and later.

This software can be downloaded directly from Xilinx' website (http://www.xilinx.com).

Any of the design suites' editions is suitable. If the target device is covered by the WebPACK Edition, it may be the preferred choice, as it can be downloaded and used with no license fee for an unlimited time.

All design suite editions cover the Xilinx-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 Xilinx' 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.

Users targeting FPGAs which are covered by Vivado, and who want to avoid HDL programming altogether, may opt for the Block Design Flow, which is in particular adequate in conjunction with High Level Synthesis (HLS) projects. As this flow presents a smaller set of Xillybus' features, it's not recommended for those having even a basic knowledge in one of the HDL languages.

For more information, see The guide to Xillybus Block Design Flow for non-HDL users.

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 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, except for ML506 (Virtex-5).
- Modifying the files to match a different target. This is suitable when targeting other boards, and/or FPGA models. This is also necessary when targeting Virtex-5 FPGAs. More information in about this in paragraph 4.3.
- Setting up the ISE (or Vivado) projects from scratch. Possibly necessary when integrating the demo bundle with existing application logic. Further details in paragraph 4.2.

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 some of the following directories (which ones depends on the target FPGA).

- 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)
- blockdesign – This directory is found in bundles for FPGA targets supported by Vivado. Contains the files related to the Block Design Flow.
- blockplus or pcie_core (or none) – This directory is found in bundles for Virtex and series 7 FPGAs. In the ISE suite flow, it must be tended to before attempting to build the rest of the bundle. Xilinx' PCIe core wrapper is built here, as it can't be included directly as a Coregen module in the FPGA's main projects (due to a slight Coregen quirk).
- vivado-essentials – Definition files and build directories for processor-related and general-purpose logic for use by Vivado..

Note that the bundle targets a specific board, as listed at the site's web page from which the 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.

For Vivado projects, this file is vivado-essentials/xillydemo.xdc, and for ISE projects it's the UCF file in the chosen 'src' directory under verilog/ or vhdl/.

Also note that the vhdl directory contains Verilog files, but none of which should need editing 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 Generating the bit file with Vivado

ISE users: Please skip to paragraph 3.4.

Vivado's outline of intermediate files is relatively complex and difficult to control. In order to keep the file structure in the bundle compact, a script in Tcl is supplied for creating the Vivado project. This script creates a new subdirectory, "vivado", and populates this directory with files as necessary.

The project relies on the files in the src/ subdirectory (no copies of these files are made). The processor, its interconnect and peripherals, as well as the FIFOs used by the logic are defined in vivado-essentials/, which is also populated with intermediate files by Vivado as the project is implemented.

Start Vivado. With no project open, Pick Tools > Run Tcl Script... and choose **xillydemo-vivado.tcl** in the verilog/, vhdl/ or blockdesign/ subdirectory, depending on your preference. A sequence of events takes place for less than a minute. The success of the project's deployment can be verified by choosing the "Tcl Console" tab at Vivado's window's bottom, and verify that it says

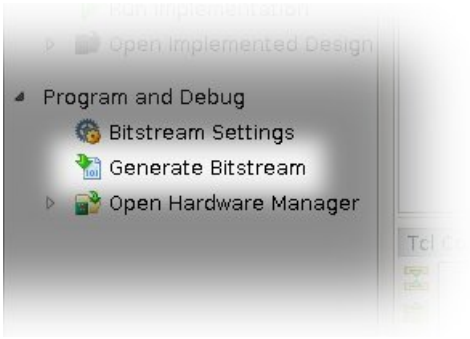
```
INFO: Project created: xillydemo
```

If this is not the bottom line of the Tcl console, something went wrong, most likely because the wrong revision of Vivado is used.

Warnings will appear during this stage, but no errors. However if the project has already been generated (i.e. the script has been run already), attempting to run the script again will result in the following error:

```
ERROR: [Common 17-53] User Exception: Project already exists on disk,
       please use '-force' option to overwrite:
```

After the project has been created, run an implementation: Click "Generate Bitstream" on the Flow Navigator bar to the left.



A popup window asking if it's OK to launch synthesis and implementation is likely to appear – pick "Yes".

Vivado runs a sequence of processes. This takes normally takes a few minutes. Several warnings are issued, none of which are classified critical (but some critical warning may still remain in the logs from the execution of the Tcl script).

A popup window, informing that the bitstream generation was completed successfully will appear, giving choices of what to do next. Any option is fine, including picking "Cancel".

The bitstream file, xillydemo.bit, can be found at vivado/xillydemo.runs/impl_1/

The implementation is never expected to fail. There are however a one error condition worth mentioning:

An error saying "Timing constraints weren't met" can happen when custom logic has been integrated, causing the tools to fail meeting the timing requirements. This means that the design is syntactically correct, but needs corrections to make certain paths fast enough with respect to given clock rates and/or I/O requirements. The process of correcting the design for better timing is often referred to as *timing closure*.

A timing constraint failure is commonly announced as a critical warning, allowing the user to produce a bitstream file with which the FPGA's behavior is not guaranteed. To prevent the generation of such a bitstream, a timing failure is promoted to the level of an error by virtue of a small Tcl script, "showstopper.tcl", which is automatically executed at the end of a route run. To turn this safety measure off, click "Project Settings" under "Project Manager" in the Flow Navigator. Choose the "Implementation" button, and scroll down to the settings for "route_design". Then remove showstopper.tcl from tcl.post.

Vivado users may skip the following sections, going directly to paragraph 3.6.

3.4 Setting up Xilinx' PCIe IP core

This part relates only to the ISE toolchain, for targets other than Spartan-6. If Vivado is used, please refer to paragraph 3.3. Those targeting Spartan-6 FPGAs may jump directly to paragraph 3.5.

A somewhat peculiar organization of the Coregen IP core for PCIe doesn't allow the inclusion of the (XCO) core file in the implementation project, but instead, the core generating software creates Verilog files for inclusion. This is the case only when targeting Virtex-5, Virtex-6 and series 7 FPGAs

Virtex-5 FPGAs require some special handling of the XCO files. See paragraph 4.3.3.

Please find the project (.xise) file in either blockplus or pcie_core directory (as found in your bundle) and double-click it to open ISE with it. Under "Design Utilities", click "Regenerate all cores" and wait for the process to finish. Then just close ISE. There is no need for any further action.

This procedure generates a set of Xilinx PCIe core wrapper Verilog files, which are referenced by the project used to create the Xillybus evaluation bitfile. These files are good regardless of your own Verilog vs. VHDL preference.

These is no need to manually include these Verilog files in the main implementation projects, but these files have to be generated once before the implementation of the entire project is attempted. There is no need to repeat this procedure, even when reimplementing the main design.

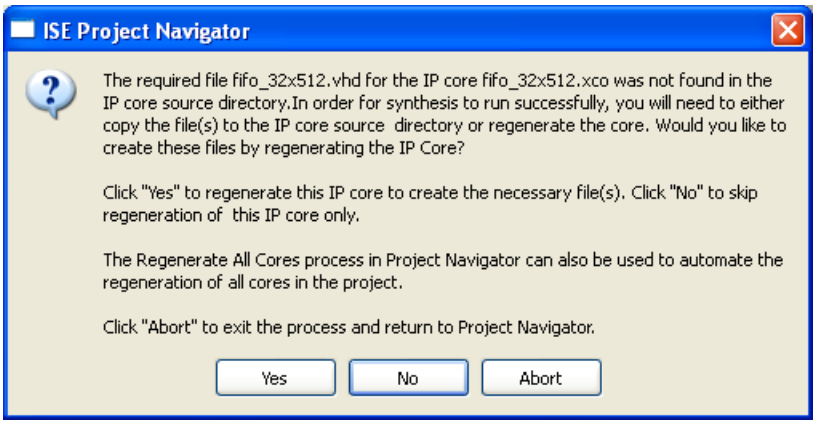
3.5 Generating the bit file with the ISE suite

When targeting a Virtex or series 7 FPGA, please make sure you've prepared the PCIe wrapper, as instructed in paragraph 3.4 above.

Depending on your preference, double-click the 'xillydemo.xise' file in either the 'verilog' or 'vhdl' subdirectory. ISE will launch and open the project with the correct settings. It shouldn't complain that any file is missing. If it does, and you're targeting a Virtex / series 7 FPGA, it's likely that your PCIe wrapper wasn't prepared as mentioned in paragraph 3.4.

Click "Generate Programming File" to implement.

During the first implementation there is a need to regenerate two or three Xilinx coregen cores. This process is time-consuming, but fortunately it's done only once. A popup window looking something like this will appear two or three times:



Click “Yes” all times.

The procedure will produce several warnings (FPGA implementations always do) but should not present any errors. When the process is finished, the bitfile can be found as xillydemo.bit.

Always verify that the timing constraints were met by looking for the following sentence in the log console (somewhere near the end):

```
All constraints were met.
```

This is crucial, in particular after making changes in the project. If the constraints aren’t met, the tools will still generate a bitfile without making a fuss about it, but the FPGA may behave in an unpredictable manner (sometimes as a result of heating within the allowed temperature).

A similar message can be found in ISE’s design summary.

Failing to meet timing constraints could be a result of adding logic that isn’t fast enough to withstand the rate of bus_clk. But if a failure occurs for no apparent reason, it could be that Xilinx’ tools made a poor initial guess when attempting to place the logic components on the FPGA’s fabric, and the optimization algorithm running later on couldn’t fix this.

The latter case can be fixed by changing the placer cost table figure, which is just a seed to the randomness of the initial placement. In the ISE Project Navigator’s Processes pane, right-click “Map” and pick “Process Properties...”. Make sure that the Property display level is “Advanced” and change the “Starting Placer Cost Table” to just any other number that hasn’t been tried yet. The magnitude of this number has no significance. Then restart “Generate Programming File”.

IMPORTANT:

On some ISE versions, notably ISE 14.2, the build in the verilog/ directory may fail with an error saying ERROR:HDLCompiler:687 - "C:/try/xillybus-eval-kintex7-1.1/verilog/src/fifo_32x512_synth.v" Line 54: Illegal redeclaration of module fifo_32x512. (or similar). This is due to a bug in Xilinx' tools. To work around this, delete fifo_8x2048_synth.v and fifo_32x512_synth.v in the src/ directory, and restart "Generate Programming File". Several warnings will indicate that the tools miss these files, but the implementation should nevertheless run through properly.

3.6 Loading the bitfile

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. On most boards, a simple USB cable between the computer running Vivado or iMPACT and a USB connector on the board's panel will do the trick. Load the FPGA, then power on the hosting computer.

Please refer to your board's instructions on how to configure the FPGA via JTAG.

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

This section relates to the Verilog / VHDL flow, which is outlined in this document. Users who have chosen the Block Design Flow should follow the instructions in The guide to Xillybus Block Design Flow for non-HDL users.

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 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 3.5 or 3.3, and then rebuilt by clicking “Generate Programming File” or “Generate Bitstream”. 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, 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, 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 First Word Fall Through) 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 Inclusion in a custom project

If desired, it's possible to include the Xillybus IP core in an existing ISE 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,

- If Vivado is used, it's recommended to edit xillydemo-vivado.tcl to reflect the custom project's source files and settings, and create a fresh project by running this script.

  The rest of the bullets below relate to the ISE suite.

- If targeting Virtex-5/6 or series 7 FPGA, the PCIe wrapper files need to be generated separately, as detailed in paragraph 3.4 (and also paragraph 4.3.3 for Virtex-5 FPGAs). The generated Verilog files should be added in the custom project (but not the XCO file).

- Add all files in one of the two src/ subdirectories (depending on your language preference) into the project.
- Add a directory to the Macro search path: In the process menu, under “Implementation”, right-click “Translate” and choose “Process Properties...”. Add the ’core’ subdirectory to the Macro Search Path property (browsing with the button to the far right). Failing to set this property will make the Translate fail to find the xillybus_core.ngc file.
- 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.

4.3 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.

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 UCF / XDC file, modify the pins’ names to those used in the Xillybus UCF / XDC file, and replace the respective rows with the ones taken from the target board’s.

The details about how to place the pins are given below.

Note that the most likely pitfall is the reference clock. Connecting just any clock with the same frequency will not work: The tiny frequency difference between the motherboard’s crystal and the applied clock will make the physical bit transceiver lose lock sporadically, resulting in an unstable interface at best.

Since the Xillybus core is based upon Xilinx’ PCIe core, the core’s user guide is a valid source for physical layer design considerations. The PDF file is automatically copied by Coregen into the pcie/doc subdirectory during implementation of the FPGA project (but merely contains a reference to the version available on the web).

There are also four GPIO_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.

4.3.1 Targeting Spartan-6 boards

For Spartan-6, the Xillybus core interfaces with the host computer through a PCIe bus port, which consists of 7 physical wires as follows:

- A reference clock differential wire pair PCIE_250M_P and PCIE_250M_N: A 125 MHz clock, (despite the net's name) which is derived from the PCIe bus clock, is expected on these wires. If a different clock is applied, the Xilinx PCIe Coregen core (defined by pcie.xco in the bundle) must be reconfigured to expect the real clock frequency. Additionally, a timing constraint must be updated, so that the TS_PCIE_CLK time specification reflects the change.
- The host's master bus reset on PCIE_PERST_B_LS
- Serial data input pair PCIE_RX0_P and PCIE_RX0_N
- Serial data output pair PCIE_TX0_P and PCIE_TX0_N

These pins' assignments are set according to the board's wiring.

4.3.2 Targeting Virtex-6 boards

For Virtex-6 the wiring scheme is similar:

- A reference clock differential wire pair PCIE_REFCLK_P and PCIE_REFCLK_N: A 250 MHz clock, which is derived from the PCIe bus clock, is expected on these wires. If a different clock is applied, the Xilinx PCIe Coregen core (defined by pcie_v6_4x.xco in the bundle) must be reconfigured to expect the real clock frequency. Such a change may also involve changes in the constraints. Please refer to the example UCF file generated by Coregen.
- The host's master bus reset on PCIE_PERST_B_LS
- Serial data input vector pair PCIE_RX_P and PCIE_RX_N (4 wires each)
- Serial data output vector pair PCIE_TX_P and PCIE_TX_N (4 wires each)

The pin assignment is made implicitly by placing the transceiver logic. The constraints defining the GTX placements in the UCF file force a certain pinout. Likewise, the placement of the input clock reference pins is implicitly set by constraining the position of the input clock buffer (pcieclk_ibuf). The UCF file in Xillybus' bundle contains guiding comments.

The UCF file must be edited so that the pin placements of these match those of the targeted board.

4.3.3 Targeting Virtex-5 boards

There are two groups of devices within the Virtex-5 family, each requiring a slightly different PCIe interface. To handle this simple, there are two different XCO files in the 'blockplus' subdirectory, only one of which should be used.

Accordingly, it's necessary to rename a file in that subdirectory as follows before building the PCIe core:

- For Virtex-5 LX or SX devices: Rename pcie_v5_gtp.xco to pcie_v5.xco
- For Virtex-5 FX or TX devices: Rename pcie_v5_gtx.xco to pcie_v5.xco

IMPORTANT:

The version of PCIe Block Plus generator should be 1.14, and definitely not 1.15. ISE 13.1 has the correct version for this purpose, but the one that arrives with ISE 13.2 will produce faulty code.

If a version other than ISE 13.1 is desired for the overall implementation, it's possible to generate the Verilog files with the correct version of PCIe Block Plus (included in ISE 13.1) but implement the entire project in the preferred version.

The UCF file has guiding comments regarding how the pins should be set up. The placement of the PCIe pins is implicit, and is forced by the constraint on the position of the GTP/GTX component.

A 100 MHz clock, is expected on the PCIE_REFCLK wire pair. If a different clock is applied, the Xilinx PCIe Coregen core (defined by pcie_v5.xco) must be reconfigured to expect the real clock frequency. Additionally, a timing constraint must be updated, so that the TS_MGTCLK time specification reflects the change.

4.3.4 Targeting Kintex-7, Virtex-7 and Artix-7 boards

All series-7 FPGA share the same PCIe interface.

- A reference clock differential wire pair PCIE_REFCLK_P and PCIE_REFCLK_N: A 100 MHz clock, which is derived from the PCIe bus clock (or connected directly), is expected on these wires. If a different clock is applied, the Xilinx PCIe

Coregen core (defined by pcie_k7_8x.xco or similar in the bundle) must be re-configured to expect the real clock frequency. Such a change may also involve changes in the constraints. Please refer to the example UCF file generated by Coregen.

- If Vivado is used, Xilinx' PCIe core is defined by e.g. pcie_k7_vivado.xci, which appears in the project's list of sources. The example XDC file should be referred to rather than a UCF file (Vivado doesn't relate to UCF files).
- The host's master bus reset on PCIE_PERST_B_LS
- Serial data input vector pair PCIE_RX_P and PCIE_RX_N (8 or 4 wires each)
- Serial data output vector pair PCIE_TX_P and PCIE_TX_N (8 or 4 wires each)

The pin assignment is made implicitly by placing the transceiver logic. The constraints defining the GTX placements in the UCF / XDC file force a certain pinout. Likewise, the placement of the input clock reference pins is implicitly set by constraining the position of the input clock buffer (pcieclk_ibuf). The UCF / XDC file in Xillybus' bundle contains guiding comments.

The UCF / XDC file must be edited so that the pin placements of these match those of the targeted board.

4.4 PRSNT pins for indicating PCIe lane settings

According to the PCIe spec, there are one or more pins on the connector, which indicate the presence and lane width of the physical interface (PRSNT pins). Most development boards have DIP switches for adjusting the number of lanes these pins inform the host about.

The typical default setting is the maximal possible lanes possible with the board, which is usually functional, even if less lanes are actually used: An initial negociation phase, which is required by the PCIe spec, takes care of the correct detection of the actual number of lanes.

Please refer to your board's reference manual about how to set these DIP switches. It's important not to set these DIP switches to less lanes that are actually used, since some hosts may ignore active physical data lanes as a result of a faulty setting.

4.5 Changing the number of PCIe lanes and/or link speed

4.5.1 Introduction

IMPORTANT:

Changing the link parameters may require adjustments in the timing constraints. Failing to pay attention to this issue can lead to a PCIe link that works, but in an unreliable manner.
Always be sure to have properly adjusted the timing constraints, if necessary, after making changes. This topic is detailed below.

Xillybus' FPGA bundles are typically set to the maximal number of lanes available on the board targeted, and a link rate of 2.5 GT/s per lane (Gen1). The rationale is that if an FPGA board fits the PCIe connector of a motherboard, one can expect all lanes to be supported by the latter. On the other hand, in almost all cases, the bandwidth achieved by the lane width is higher than the Xillybus IP core may utilize, even with 2.5 GT/s, and it's hence pointless to set a higher link rate.

As the PCIe specification requires a fallback capability to lower speeds from all bus components involved, picking 2.5GT/s ensures a uniform behavior on all motherboards.

It's however often desired to change the lane width and speed, in particular when using the Xillybus IP core on a custom board. Less lanes with higher link speed is a common requirement.

The Xillybus IP core relies on Xilinx' PCIe block for the low-level interface with the PCIe bus. Accordingly, the IP core works properly as long as Xilinx' PCIe block operates properly, regardless of the lane number or link speed.

If the PCIe block is configured with a low number of lane count combined with a low speed link, such that its bandwidth capability is lower than Xillybus IP core's, it will still work properly. In this case, the aggregate bandwidth offered by Xillybus' streams approximately equals the bandwidth limit imposed by the PCIe block's setting – it's a plain bottleneck situation.

4.5.2 The workflow

In principle, changing the lane width and/or link speed merely consists of making changes in the PCIe block's configuration as desired. However there are a few issues to pay attention to:

- The modification may influence other parameters of the PCIe block, to the extent that causes a failure to operate. Among others, there's a bug in Xilinx' GUI tool to watch out for, as detailed below.
- The modification may change the frequencies of the clocks driving the PCIe block (the PIPE clock), and hence requires changes in timing constraints.
- The said change in clock frequencies may also require changes in Verilog code that supports the PCIe block (the instantiation of the pipe_clock module, where applicable), or the PCIe block will not function.

The stages are hence as follows:

1. Make a copy of the XCO or XCI file on an active project (i.e. after Vivado or ISE has upgraded the IP as necessary). This will allow comparing the changes with a diff tool afterwards, and spot possibly unwanted changes.
2. Open the PCIe IP block in Vivado (or ISE) for configuration.
3. Change the number of lanes and/or Maximum Link Speed as desired, while paying attention not to change the (AXI) interface width. If it's possible to avoid changing the (AXI) Interface Frequency by choosing a functionally acceptable combination of lanes and speed, that is preferable.
4. After making the desired changes, verify that the Vendor and Device IDs haven't changed (neither the Subsystem counterparts). Some revisions of Vivado may reset some parameters to their defaults as a result of unrelated modifications (this is a bug).
5. Confirm the changes (typically click "OK" at the bottom of the dialog box). There is no need to generate output products, if this is suggested following the confirmation.
6. Compare the updated and previous XCO or XCI files with a textual diff tool, and verify that only relevant parameters have changed. More on this below.
7. Adjust the PIPE clock module's instantiation if necessary, as described in paragraph 4.5.3 below.
8. Adjust timing constraints if necessary, as described in paragraph 4.5.4 below.

When comparing new and old XCI files, `PARAM_VALUE.Device_ID` should get special attention, as it's often changed accidentally.

The differences in the parameters of the XCI files should match those desired. This is a short list of possible parameters for which changes are acceptable in accordance with the changes made in Vivado. The names of the parameters should be taken with a grain of salt, as different revisions of Vivado (and hence different revisions of the PCIe block) may represent the PCIe block's attributes with different XML parameters.

- Related to the number of lanes:
  - `PARAM_VALUE.Maximum_Link_Width`
  - `MODELPARAM_VALUE.max_lnk_wdt`
- Related to the link speed:
  - `PARAM_VALUE.Link_Speed`
  - `PARAM_VALUE.Trgt_Link_Speed`
  - `MODELPARAM_VALUE.c_gen1`
  - `MODELPARAM_VALUE.max_lnk_spd`
- Related to the the interface frequency. A change in these parameters is a strong indication that the stages in paragraphs 4.5.3 and 4.5.4 are necessary.
  - `PARAM_VALUE.User_Clk_Freq`
  - `MODELPARAM_VALUE.pci_exp_int_freq`

4.5.3 Has the PIPE frequency changed?

When targeting Ultrascale FPGAs and later, the considerations and actions below are unnecessary, as their PCIe blocks supply the timing constraints and the PIPE module as an integral part of the IP itself.

As for other FPGA families, it's important to verify that the PIPE clock settings are correct as follows:

Generate an example project for the PCIe block **after the changes**, and synthesize the project. In Vivado, this is typically done by right-clicking the PCIe block in the project's source hierarchy and select "Open IP Example Design...". After selecting a location for the design, and it has been generated, synthesize the project: Click "Run Synthesis" at the left column.

Next, obtain the PIPE clock module's instantiation parameters in the synthesis report (in Vivado, it's found as something like pcie_example/pcie_example.runs/synth_1/runme.log). In this report, search for a segment like:

```
INFO: [Synth 8-638] synthesizing module 'example_pipe_clock' [...]
Parameter PCIE_ASYNC_EN bound to: FALSE - type: string
Parameter PCIE_TXBUF_EN bound to: FALSE - type: string
Parameter PCIE_CLK_SHARING_EN bound to: FALSE - type: string
Parameter PCIE_LANE bound to: 4 - type: integer
Parameter PCIE_LINK_SPEED bound to: 3 - type: integer
Parameter PCIE_REFCLK_FREQ bound to: 0 - type: integer
Parameter PCIE_USERCLK1_FREQ bound to: 4 - type: integer
Parameter PCIE_USERCLK2_FREQ bound to: 4 - type: integer
Parameter PCIE_OOBCLK_MODE bound to: 1 - type: integer
Parameter PCIE_DEBUG_MODE bound to: 0 - type: integer
```

The parameters in this report must match those in the instantiation of pipe_clock in xillybus.v, which is of the form:

```
pcie_[...]_pipe_clock #
   (
    .PCIE_ASYNC_EN                  ( "FALSE" ),
    .PCIE_TXBUF_EN                  ( "FALSE" ),
    .PCIE_LANE                      ( 6'h08 ),
    .PCIE_LINK_SPEED               ( 3 ),
    .PCIE_REFCLK_FREQ               ( 0 ),
    .PCIE_USERCLK1_FREQ             ( 4 ),
    .PCIE_USERCLK2_FREQ             ( 4 ),
    .PCIE_DEBUG_MODE                ( 0 )
    )
   pipe_clock
     (
       [ ... ]
      );
```

The three parameters to compare are `PCIE_LINK_SPEED`, `PCIE_USERCLK1_FREQ` and `PCIE_USERCLK2_FREQ`, which must match. If they do (as shown in the example), all is set correctly, including the timing constraints. If not, two actions must be taken:

- The instantiation parameters in xillybus.v must be updated to match those in the example design's synthesis report.
- The timing constraints must be adapted to the example design's. This is more difficult, because failing to do this correctly doesn't necessary cause a problem immediately, but may impact the design's reliability.

If the `PCIE_LANE` parameter in xillybus.v is larger than the example design's, there is

no problem leaving it that way, and it's often easier to do so.

4.5.4 Adapting the timing constraints

If the PIPE clock frequency has changed, it's mandatory to adjust the timing constraints to reflect the PCIe block's updated clocking. This requires some understanding on what the constraints are intended for before and after the change, so the PCIe link's reliability can be be ensured by virtue of proper constraining.

As the constraining requirements depend on the FPGA target as well as Vivado's revision, it may be somewhat difficult to get this done correctly. Avoiding this adjustment is the main motivation for attempting to keep the PIPE clock frequency unchanged by selecting a combination of link speed and lane width – when possible.

Once again, there is no need to deal with timing constraints when Ultrascale FPGAs and later are targeted, as their PCIe blocks handle this internally.

In order to match the timing constraints, find the constraints of the example design. With Vivado, it's typically as a file of the form `example.srcs/constrs_1/imports/example_design/xilinx_*.xdc`.

Compare the constraint file's "Timing constraints" section with those in xillydemo.xdc. The example design references logic elements by their absolute position in the logic hierarchy, so some editing is necessary. For example, the constraint

```
set_false_path -to [get_pins {pcie_vivado_support_i/pipe_clock_i/
pclk_i1_bufgctrl.pclk_i1/S0}]
```

appears in xillydemo.xdc as

```
set_false_path -to [get_pins -match_style ucf */pipe_clock/
pclk_i1_bufgctrl.pclk_i1/S0]
```

The main differences are with the relative paths used in Xillybus' constraints. There might also be slight differences, as some constraints are necessary on earlier revisions of Xilinx' tools, and then become superfluous.

After making the changes in the timing constraints, it's important to verify that they took effect on logic by reviewing the timing reports after the design's implementation.

Finally, it's worth explaining the following two constraints, which may be encountered in xillydemo.xdc (before any changes):

```
set_case_analysis 1 [get_pins -match_style ucf */pipe_clock/
```

```
pclk_i1_bufgctrl.pclk_i1/S0]
set_case_analysis 0 [get_pins -match_style ucf */pipe_clock/
pclk_i1_bufgctrl.pclk_i1/S1]
```

These constraints are required for Gen1 PCIe blocks, when using quite old revisions of Vivado, as explained in Xilinx AR #62296. Hence they may be omitted with recent Xilinx tools.

5

Troubleshooting

5.1 Implementation errors

Slight differences between releases of Xilinx' tools sometimes result in failures to run the implementation chain for creating a bitfile.

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 ISE tools was used (or if it was Vivado).

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 10EE 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 Xilinx' 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 (possibly looking at one of the xillydemo_pad files in the implementation directories)
- 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 Xilinx' PCIe cores by invoking Coregen for each (double-clicking the XCO element in Project Navigator with each of the projects open). On Vivado, right-click the XCI file in the sources list that represents Xilinx' PCIe core, and choose "Re-customize IP..." to view the core's attributes.

The following configuration elements may need adjustment:

- The frequency of the reference clock (may appear as "Interface Frequency" in the GUI).
- 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.