Getting started with Xillybus on a Windows host

Xillybus Ltd.

www.xillybus.com

Version 2.0

1 Introduction

This is a walkthrough guide for installing the driver for Xillybus on a Windows host, as well as trying out the basic functionality of the IP core.

This guide assumes that the FPGA is loaded with the Xillybus demo bundle code, and has been recognized as a PCI Express peripheral by the host. The steps for reaching this are outlined in one of these two documents (depending on the chosen FPGA):

- Getting started with the FPGA demo bundle for Xilinx
- Getting started with the FPGA demo bundle for Altera

The host driver generates device files which behave like named pipes: They are opened, read from and written to just like any file, but behave much like pipes between processes or TCP/IP streams. To the program running on the host, the difference is that the other side of the stream is not another process (over the network or on the same computer), but a FIFO in the FPGA. Just like a TCP/IP stream, the Xillybus stream is designed to work well with high-rate data transfers as well single bytes arriving or sent occasionally.

One driver binary supports any Xillybus IP core configuration: The streams and their attributes are auto-detected by the driver as it's loaded into the host's operating system, and device files are created accordingly. These pipe-like device files appear as `\\.\`xillybus_*something*.

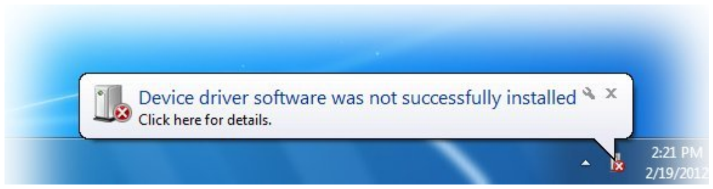
More in-depth information on topics related to the host can be found in Xillybus host application programming guide for Windows.

2 Installing the host driver

2.1 Installation procedure

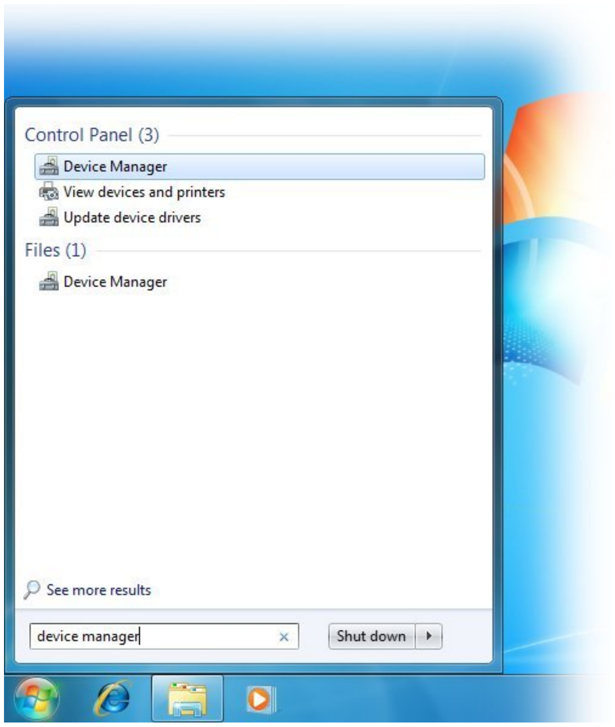
There is nothing special about installing the driver for Xillybus on Windows. The procedure described below is the straightforward flow for installing a device driver from a specific location on the disk.

When Windows finishes booting up the first time with the Xillybus IP core visible to the host, it's likely that a warning bubble will appear, saying that a piece of hardware was found, but wasn't installed:

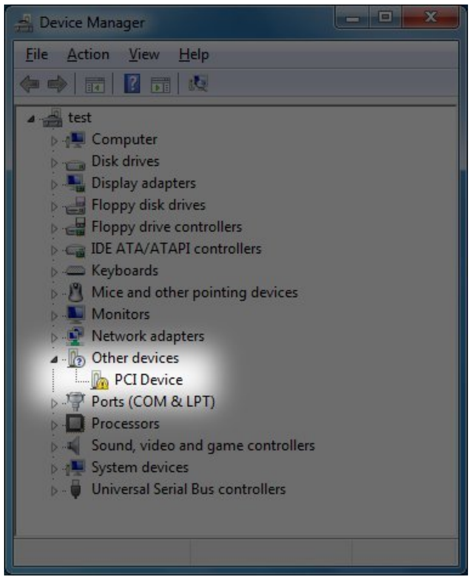

Device driver software was not successfully installed
Click here for details.

This is normal, and is a sign Windows has detected something it doesn't yet recognize.

Start off by running the Device Manager. This is easiest done by clicking on the "Windows start" button and type "device manager" as follows, and then click on the topmost item:



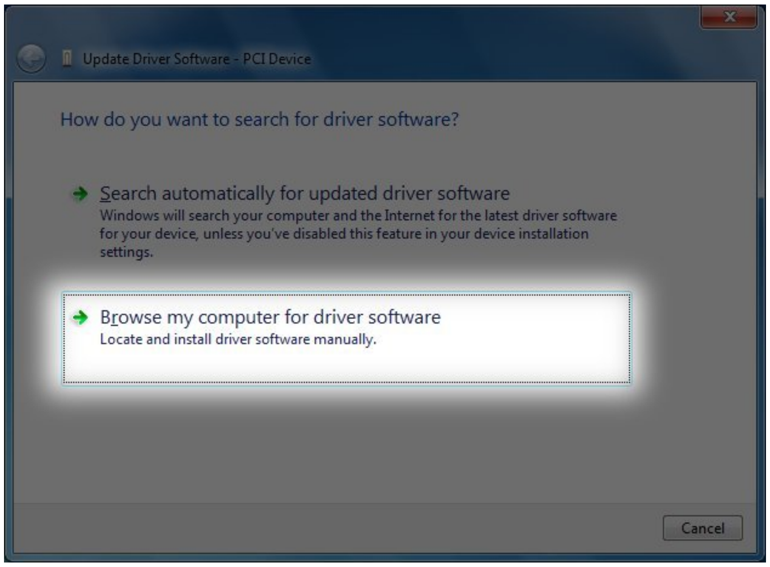
The opened device manager will look something like this (important part highlighted):



If this “PCI Device” entry doesn’t appear in the Device Manager, there are several possible causes for this:

- The Xilinx / Altera PCIe interface isn't detected, possibly because a board mis-configuration (jumpers etc.), a faulty pin assignment, wrong reference frequency etc. Note that the problem is Xilinx' / Altera's PCIe IP core not being detected, and has nothing to do with Xillybus, which uses the FPGA's native core.
- The PCIe interface wasn't detected by BIOS at bootup because of a slow FPGA configuration. Rebooting Windows (without powering off and on) is the safe way to fix this, even though Action > Scan for New Hardware may work too.
- The Xillybus driver is already installed. In that case the device manager should look like the example shown at the end of the installation procedure.

Right-click the "PCI Device" item and pick "Update Driver Software..." which will open the following window:

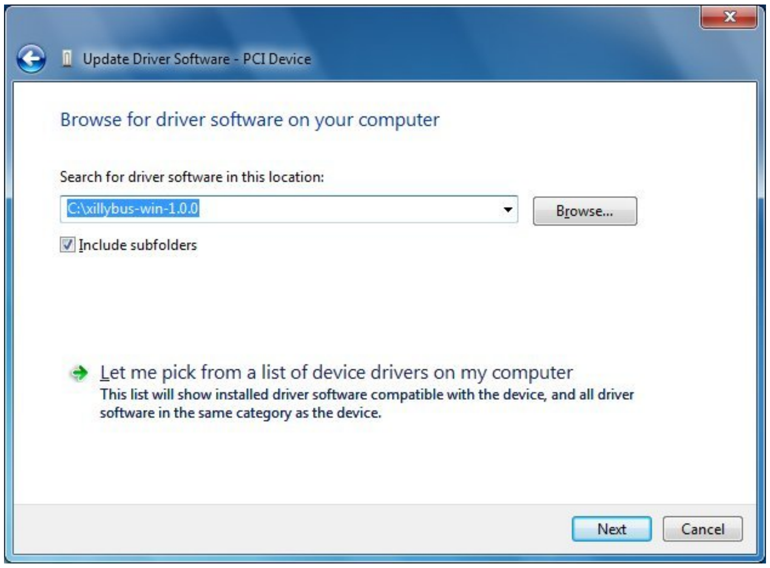


How do you want to search for driver software?

Search automatically for updated driver software
Windows will search your computer and the Internet for the latest driver software for your device, unless you've disabled this feature in your device installation settings.

Browse my computer for driver software
Locate and install driver software manually.

Choose "Browse my computer for driver software", which opens this window:



Update Driver Software - PCI Device

Browse for driver software on your computer

Search for driver software in this location:

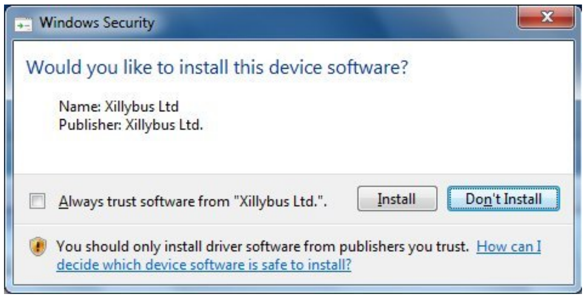
C:\xillybus-win-1.0.0

[✓] Include subfolders

Let me pick from a list of device drivers on my computer
This list will show installed driver software compatible with the device, and all driver software in the same category as the device.

Using the “Browse...” button, navigate to where the driver was unzipped to. The directory name can of course differ, depending on the driver’s version and unzip destination.

The next step is to confirm the installation:



Windows Security

Would you like to install this device software?

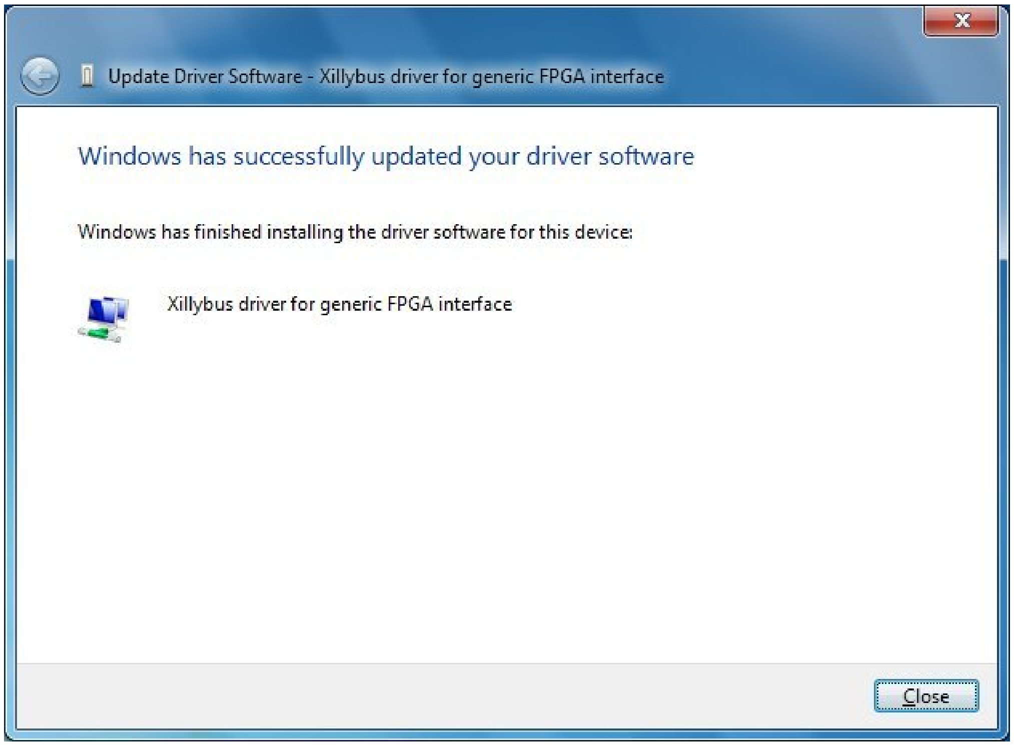
Name: Xillybus Ltd
Publisher: Xillybus Ltd.

[ ] Always trust software from "Xillybus Ltd.".

You should only install driver software from publishers you trust. How can I decide which device software is safe to install?

Click on “Install”.

The process of installing the driver takes 10-20 seconds, after which the following window announces the successful completion of the installation:



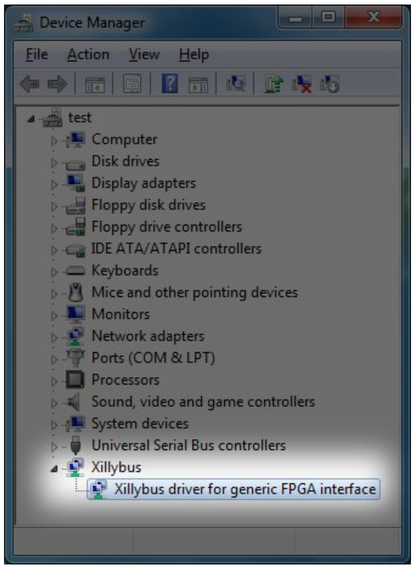
Update Driver Software - Xillybus driver for generic FPGA interface

Windows has successfully updated your driver software

Windows has finished installing the driver software for this device:

Xillybus driver for generic FPGA interface

The device manager will now show the newly installed device:



At this point, the driver is installed in the system, and has automatically loaded. It will reload every time the system is booted with the Xillybus IP core on the PCIe bus.

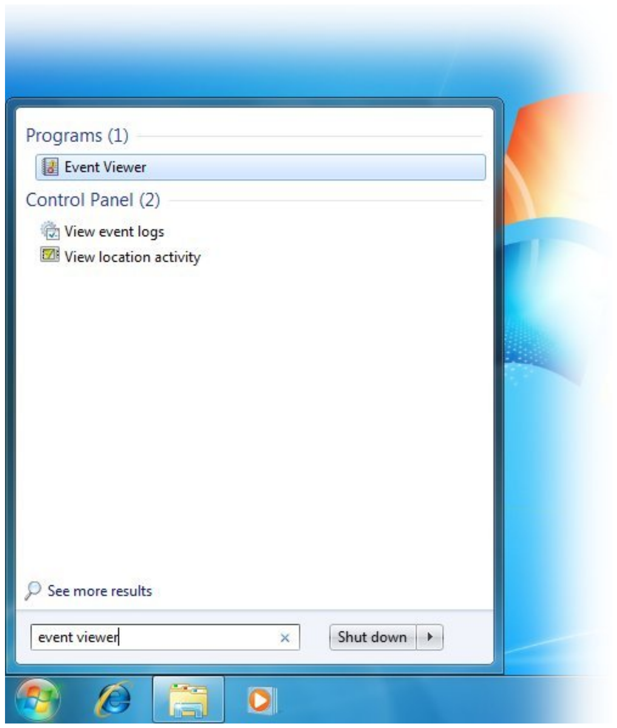
It's recommended to set up the Event Viewer to display Xillybus log messages, as explained next.

2.2 Retrieving diagnostic information (Windows)

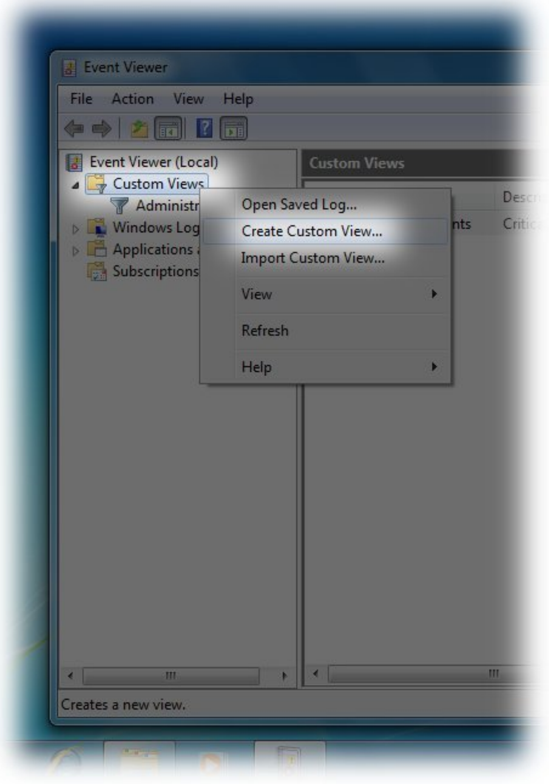
The Windows driver for Xillybus sends diagnostic messages to the operating system's main event logger. These messages include information about what went wrong when the driver fails to initialize (e.g. there wasn't enough memory for DMA buffers) and other messages which can be helpful in understanding unexpected behavior.

The sequence outlined next shows how to create a custom view in the event viewer, so only Xillybus-related messages are displayed.

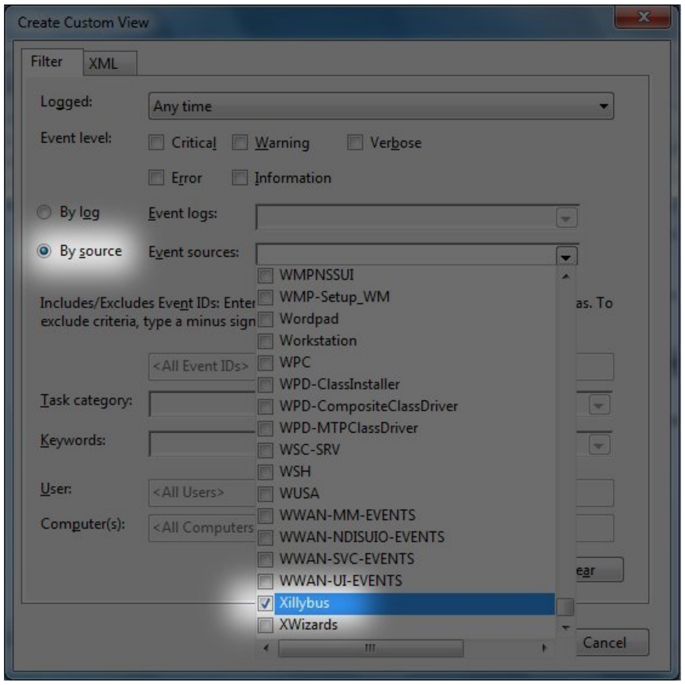
First, open the Event Viewer. This is easiest done by clicking on the "Windows start" button and type "event viewer" as follows, and then click on the topmost item:



The Event Viewer will open. To create a custom view for Xillybus messages only, right-click "Custom Views" and choose "Create Custom View..." from the menu.



A window for defining the custom view's filtering will open. Pick "By source" and select Xillybus in the drop-down menu. The defaults should be left for the other options.



Logged: Any time

Event level: [ ] Critical [ ] Warning [ ] Verbose [ ] Error [ ] Information

( ) By log Event logs:

(●) By source Event sources:

[ ] WMPNSSUI
[ ] WMP-Setup_WM
[ ] Wordpad
[ ] Workstation
[ ] WPC
[ ] WPD-ClassInstaller
[ ] WPD-CompositeClassDriver
[ ] WPD-MTPClassDriver
[ ] WSC-SRV
[ ] WSH
[ ] WUSA
[ ] WWAN-MM-EVENTS
[ ] WWAN-NDISUIO-EVENTS
[ ] WWAN-SVC-EVENTS
[ ] WWAN-UI-EVENTS
[✓] Xillybus
[ ] XWizards

Includes/Excludes Event IDs: Enter … exclude criteria, type a minus sign … as. To

<All Event IDs>

Task category:

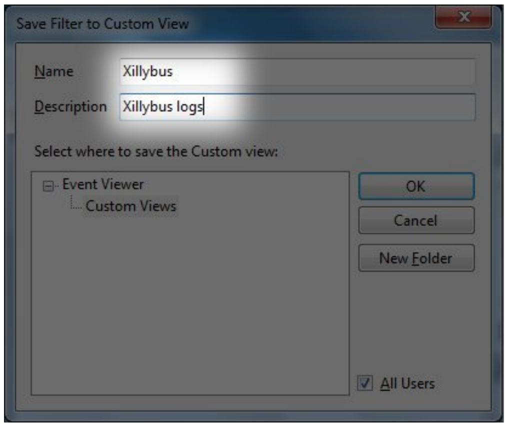
Keywords:

User: <All Users>

Computer(s): <All Computers

If the Xillybus entry in the drop-down list is not found, check if the driver is properly installed.

After clicking “OK” a window will open for assigning this custom view a name and description. This is open for personal choice:



Name: Xillybus

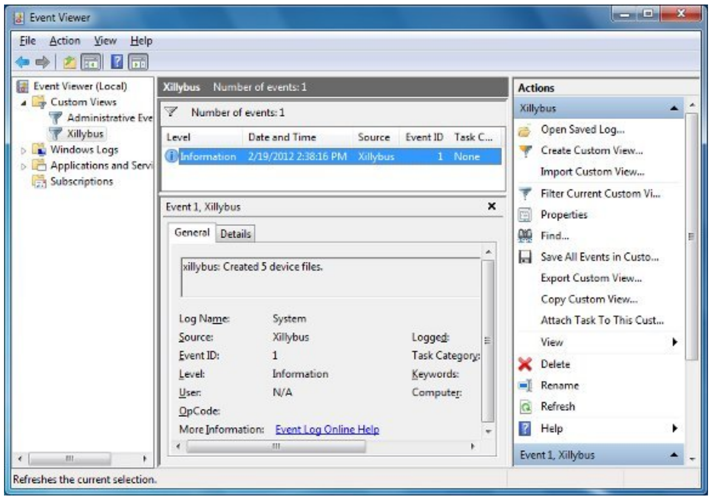
Description: Xillybus logs

Select where to save the Custom view:

Event Viewer
Custom Views

[✓] All Users

After clicking “OK”, the event viewer will look something like this:



Xillybus Number of events: 1

Number of events: 1

Level	Date and Time	Source	Event ID	Task C...
Information	2/19/2012 2:38:16 PM	Xillybus	1	None

Event 1, Xillybus

xillybus: Created 5 device files.

Log Name:	System		
Source:	Xillybus	Logged:	
Event ID:	1	Task Category:	
Level:	Information	Keywords:	
User:	N/A	Computer:	
OpCode:			
More Information:	Event Log Online Help		

In the example above there is one log entry in this view, informing that Xillybus has started properly and created 5 device files. This is what should be expected immediately after successfully installing the driver, and the FPGA is loaded with the demo bundle.

The log entries are not deleted when rebooting the computer, so this custom view shows the history since the driver was installed (unless the log is cleaned up).

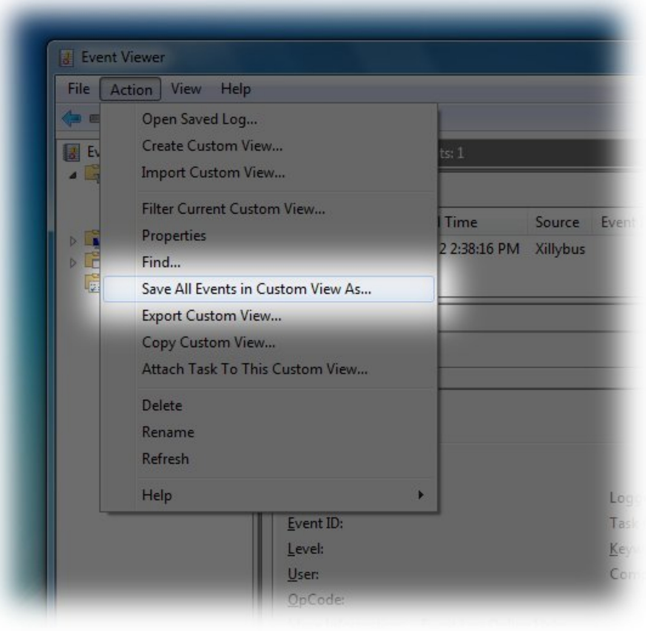
There are explanations for many of the log messages at

http://xillybus.com/doc/list-of-kernel-messages

It may however be easier to find a specific message by using Google on the message text itself.

Either way, please don't hesitate sending an email reporting problems that may occur, if they're not resolved quickly.

It's also possible to export log messages to a file. When asking for support, it's always a good idea to attach the log message file which contains valuable information. To do this, choose the "Action" menu item and "Save All Events in a Custom View As..."



This will open a file selection window. The preferred output format is CSV.

3 Command line tests

3.1 Introduction

The Windows driver exposes its interfaces through device files with a `\\.\xillybus_` prefix. These operating system objects behave like files, and can be accessed with the same interface. Those who are not familiar with Windows internals, may mistake this for a shaky hack, but the truth is that much of the hardware access on a Windows computer is made through device files like this.

It's indeed uncommon that these device files are exposed directly to user space applications. Rather, they're usually accessed by a vendor-supplied DLL, which supplies an API, wrapping the direct access to the hardware driver. As Xillybus' interface is so simple, this DLL API was omitted.

Since Windows' philosophy is that device files aren't accessed directly, there's no immediate way to get a listing of those. This is the list of the device files generated by the demo bundle:

- `\\.\xillybus_read_8`
- `\\.\xillybus_write_8`
- `\\.\xillybus_read_32`
- `\\.\xillybus_write_32`
- `\\.\xillybus_mem_8`

The list of device files of custom IP cores, generated by the IP Core Factory, are listed in the README file, included in the downloaded zip file.

Note that the backslashes need to be escaped when used in most programming languages (C/C++ included), so it's `\\\\.\\xillybus_read_8` when used as an argument to a function in C.

The WinObj utility (available for download at Microsoft's site) allows browsing through Window's internal object structure. The Xillybus device files appear as symbolic links in the GLOBAL?? "subdirectory", along with the well-known C:, COM1: etc.

For output of the Xillybus device files to standard output (command-line use), download the accesschk utility from Microsoft's site, and use

```
> accesschk -o \\GLOBAL\?\?
```

Note that many other global device files are listed with this operation.

3.2 Using command line interface

Xillybus device files are designed to work well with any user space application in Windows. Command-line tools are however great for the first steps with the interfaces, as well as obtaining data dumps easily.

Command line tests can be done with the sample host applications, as explained in paragraph 4. Alternatively, general-purpose command-line utilities can be used, as detailed next.

Those who prefer to jump to programming right away, may want to consult the Xillybus host application programming guide for Windows.

The command-line tools for Windows are unfortunately not nearly as rich as those available in every UNIX-based system. The DOS command prompt's built-in commands render useless when used not exactly as expected, and the standard set of shell programs supplied with Windows were clearly not meant for something useful. Luckily, the UNIX commands have been ported to Windows in many ways. Using these ported utilities, the "Hello world" usage examples shown in Getting started with Xillybus on a Linux host can be run on a Windows machine.

A few things to be observant about:

- Windows keeps its global device files in the `\\.\` path rather than the well-known UNIX `/dev/` directory. For example, when a device file is given in a Linux-oriented context as `/dev/xillybus_read_8`, Windows users should refer to `\\.\xillybus_read_8`

- Since the backslash is commonly interpreted as an escape character, the `\\.\` path needs to be written as `\\\\.\\` when used in Cygwin and in most programming languages. In other words, `/dev/xillybus_read_8` is reached as `\\\\.\\xillybus_read_8` in Cygwin and in literal strings in languages such as C and Perl. See paragraph 3.4.
- The backslash escaping is **not** used in the DOS command window when passing the device file as an argument to a command-line binary program, since the DOS environment doesn't consider the backslash as an escape character. Different levels of escaping may be necessary for scripts, depending on how they juggle their arguments internally.

3.3 UNIX port alternatives

Please note that the sample host applications for Windows in section 4 are based upon Microsoft's compiler, which is free for download. The choice of UNIX utilities is not related to these sample applications, but only to running standalone utilities.

There are several possibilities for running UNIX utilities on a Windows machine. These are a few suggestions:

- Install Cygwin. This is the heavy-weight choice, and may include a complete GNU C compiler and development environment. The definite choice if a Linux feel is desired, and also if only command-line tools based upon plain system interface are to be developed.
- Download and install the Gnuwin32 port of UNIX utilities for Windows. In particular, installing only the Coreutils and Util-Linux-NG packages covers the utilities used in this site's examples (when installing both). Note that Gnuwin32's setup utilities **don't** change the DOS Command Window's execution path.
- Use the utilities supplied in the Xillybus Windows package (available for download), in the unixutils subdirectory. These are just the set of Gnuwin32 utilities, handpicked to support the examples in this site. This is the preferred choice for minimal use of command line utilities.

3.4 Cygwin Notes

Xillybus has been extensively tested with Cygwin. Please note that the following session in a Cygwin bash terminal is **correct** :

```
$ cat \\\\.\\xillybus_read_8
cygwin warning:
 MS-DOS style path detected: \\.\xillybus_read_8
 Preferred POSIX equivalent is: //./xillybus_read_8
 CYGWIN environment variable option "nodosfilewarning" turns off this warning.
 Consult the user's guide for more details about POSIX paths:
 \url{http://cygwin.com/cygwin-ug-net/using.html#using-pathnames}
```

- The warning message appears on the first usage of backslashes in a path name, and should be **ignored** . Possibly set the environment variable as suggested to avoid it. For normal file names, it's indeed preferred to use forward slashes, but Cygwin **doesn't** translate `//./` into `\\.\` and therefore the backslash version is mandatory.
- Note that the backslashes are doubled, since Cygwin treats the single backslash as an escape character

4 Sample host applications

4.1 General

There are six simple C command-line programs in the Xillybus package for Windows, under the demoapps subdirectory. Their compiled executables can be found in precompiled-demoapps. These C sources were written for compilation with Microsoft's Visual C++ compiler, which can be downloaded free as part of the Windows SDK. Adopting these for Visual Studio should not be difficult.

Users who prefer to compile with Cygwin can follow the instructions for Linux, substituting the /dev/ prefixes with `\\\\.\\` when running the programs. A similar flow is possible when using the MinGW toolset. See the command-line notes for more about this.

Anyhow, for those wishing to stay with Microsoft's compiler, the demoapps directory consists of the following files:

- Makefile – This file contains the rules used by the "nmake" utility to compile the five programs.
- streamread.c – Read from a file, send data to standard output
- streamwrite.c – Read data from standard input, send to file
- memread.c – Read data after seeking. Demonstrates how to access a memory interface in the FPGA
- memwrite.c – Write data after seeking. Also demonstrates how to access a memory interface in the FPGA

- fifo.c – Demonstrate the implementation of a userspace RAM FIFO for continuous data streaming. Referred to in the Xillybus host application programming guide for Windows.
- winstreamread.c – Read from a file, send data to standard output, demonstrating direct use of Windows file API

Despite targeting Microsoft's compiler, all programs except the last are written in classic UNIX style. The purpose of these programs is to show the correct coding style and serve as basis for writing custom applications. They are not individually documented here, as they are very simple and all follow common file access practice, which is discussed further in the Xillybus host application programming guide for Windows.

Note that these programs use the plain open(), read(), write() etc. functions rather than fopen(), fread(), fwrite() set, since the latter may cause unexpected behavior due to data caching in the C library level.

4.2 Compilation

Even though there are precompiled binaries in the Xillybus Windows package, the compilation of the demo applications is obviously necessary to make changes in the code.

Microsoft offers a software development kit (SDK) for free download. This kit includes, among others, a C compiler and a build utility, and is the assumed choice of toolset in the compilation procedure described below. It is however expected that compilation will be straightforward with other compilers as well, in particular those from Microsoft.

After installing the Windows SDK, open the Program Files entry in the "Start menu" and pick Microsoft Windows SDK v7.1 > Windows SDK 7.1 Command Prompt (other versions apply as well). This opens a DOS command window with certain environment variables set (and a yellow text font).

Change directory to where the C files are:

```
> cd \path\to\demoapps
```

To compile all programs, just type "nmake" at prompt. The following session is expected:

```
> nmake
```

```
Microsoft (R) Program Maintenance Utility Version 10.00.30319.01
Copyright (C) Microsoft Corporation. All rights reserved.

 if not exist "XP32_DEBUG/" mkdir XP32_DEBUG
 cl -D_CRT_SECURE_NO_WARNINGS -c -DCRTAPI1=_cdecl -DCRTAPI2=_cdecl -nologo  [ ... ]
 link /INCREMENTAL:NO /NOLOGO -subsystem:console,5.01 -out:XP32_DEBUG\ [ ... ]
 cl -D_CRT_SECURE_NO_WARNINGS -c -DCRTAPI1=_cdecl -DCRTAPI2=_cdecl -nologo  [ ... ]
 link /INCREMENTAL:NO /NOLOGO -subsystem:console,5.01 -out:XP32_DEBUG\ [ ... ]
 cl -D_CRT_SECURE_NO_WARNINGS -c -DCRTAPI1=_cdecl -DCRTAPI2=_cdecl -nologo  [ ... ]
 link /INCREMENTAL:NO /NOLOGO -subsystem:console,5.01 -out:XP32_DEBUG\ [ ... ]
 cl -D_CRT_SECURE_NO_WARNINGS -c -DCRTAPI1=_cdecl -DCRTAPI2=_cdecl -nologo  [ ... ]
 link /INCREMENTAL:NO /NOLOGO -subsystem:console,5.01 -out:XP32_DEBUG\ [ ... ]
 cl -D_CRT_SECURE_NO_WARNINGS -c -DCRTAPI1=_cdecl -DCRTAPI2=_cdecl -nologo  [ ... ]
 link /INCREMENTAL:NO /NOLOGO -subsystem:console,5.01 -out:XP32_DEBUG\ [ ... ]
 cl -D_CRT_SECURE_NO_WARNINGS -c -DCRTAPI1=_cdecl -DCRTAPI2=_cdecl -nologo  [ ... ]
 link /INCREMENTAL:NO /NOLOGO -subsystem:console,5.01 -out:XP32_DEBUG\ [ ... ]
```

The six lines starting with “cl” are the invocations of the compiler, generated by the “nmake” utility. These commands can be used to compile any of the programs individually (but there is no reason to do so. Just use “nmake”). Same goes for the “link” commands, which link the object files with libraries and create executables.

The “nmake” utility compiles what is necessary to compile. If only one of the source files is changed, only that file will be compiled in a subsequent call to “make”. So the normal flow of work is to edit whatever source file, and then call “make” to recompile as necessary.

To remove the executables generated by compilation, type “nmake clean”.

As mentioned above, the compilation rules are in the Makefile. Its syntax can be somewhat cryptic, but fortunately it’s one of those files which can be edited without understanding them exactly.

The given Makefile relates to files only in the current directory. That means that it’s possible to make a copy of the entire directory, and work on the copy without collisions. It’s also possible to copy a single C file, and easily change the rules so that “nmake” compiles it as well.

The executables (and the object files) can be found in the XP32_DEBUG subdirectory, which is generated during the build process. As the directory’s name implies, these files are targeted to 32-bit Windows XP, but run on 32/64-bit Windows 7 platforms as well.

4.3 Execution

With the executables just derived from compilation, or using the precompiled binaries downloaded, a simple loopback example with two terminals is shown next. After compiling, change directory to XP32_DEBUG or skip the compilation and use the precompiled-demoapps subdirectory. Type in the first DOS command prompt:

```
> streamread \\.\xillybus_read_8
```

Note that there is no escaping of the backslashes. If this was run with Cygwin rather than in a plain DOS command window, it would be `\\\\.\\xillybus_read_8`.

This is the part that reads from the device file. In the second DOS window, after changing to the correct directory:

```
> streamwrite \\.\xillybus_write_8
```

Now type some text on the second DOS window, and press ENTER. The same text will appear in the first DOS window. The reason nothing is sent until ENTER is pressed, is that the standard input is designed not to bother applications with every character.

If an error message was received while attempting these two commands, please check for typos. Otherwise, check for errors in the Event Viewer, as explained in paragraph 2.2.

Either of these two commands can be halted with a CTRL-C.

This loopback test works as long as both ends of the FIFO are connected to the core. Changes in the FPGA code will of course change the outcome of the operations described above.

Note that there is no risk for data overflow or underflow by the core on the FIFOs, since the core respects the hardware 'empty' and 'full' signals. When necessary, the Xillybus driver forces the application to wait until the FIFO is ready for I/O (by classic blocking = forcing the application to sleep).

There is another pair of device files with a FIFO inbetween, \\.\xillybus_read_32 and \\.\xillybus_write_32. These device files work with a 32-bit word granularity, and so does the FIFO in the FPGA. Attempting the same test as above will result in similar behavior, with one difference: All hardware I/O runs in chunks of 4 bytes, so when the input hasn't reached a round boundary of 4-byte chunks, the last byte(s) will remain untransmitted.

Note that unlike the Linux version, streamwrite for Windows doesn't attempt to alter the console settings, so it works on a line-by-line basis.

4.4 Memory interface

The memread and memwrite are more interesting, because they demonstrate something which can't be shown with simple command-line utilities: Accessing memory on the FPGA by seeking the device file. Note that in the evaluation kit, only xillybus_mem_8 is seekable. It's also the only device file which can be opened for both read and for write.

Before writing to the memory, the current situation is observed by using the hexdump utility (can be found in the Windows package, in the unixutils directory):

```
> hexdump -C -v -n 32 \\.\xillybus_mem_8
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000020
```

The output may vary: It reflects the FPGA's RAM which may contain other values to begin with (most likely because of previous games with it).

The -C and -v flag told hexdump to use the output format shown. The "-n 32" part asked for the first 32 bytes only. The memory array is just 32 bytes long, so there is no point reading more.

A word about what happened: hexdump opened `\\.\xillybus\mem_8` and read 32 bytes from it. Every seekable file starts at position zero by default when opened, so the output is the first 32 bytes in the memory array.

Changing the value of the memory at address 3 to 170 (0xaa in hex):

```
> memwrite \\.\xillybus_mem_8 3 170
```

And read the entire array again:

```
> hexdump -C -v -n 32 \\.\xillybus_mem_8
00000000  00 00 00 aa 00 00 00 00  00 00 00 00 00 00 00 00  |...ł............|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000020
```

So quite evidently, it worked. The memread programs demonstrates how to read individual bytes from the RAM. Its main interest is how it's written, though.

The magic in memwrite.c happens where it says "lseek(fd, address, SEEK_SET)". With this command, the FPGA's address is set, causing any subsequent reads or writes start from that position (and increment as the data flows).

5 Troubleshooting

The Windows driver for Xillybus was designed to produce meaningful log messages in the system's event log. It is therefore recommended to look for messages by filtering the log for Xillybus-related events, as described in paragraph 2.2, when something appears to be wrong.

It's in fact advisable to keep an eye on the log even when everything appears to work fine.

Some messages are listed and explained at

http://xillybus.com/doc/list-of-kernel-messages

It may however be easier to find a specific message by using Google on the message text itself.

Please don't hesitate seeking assistance through email, if a problem is not resolved quickly.