
Getting started with Xillybus on a Linux host

Xillybus Ltd.

www.xillybus.com

Version 2.0

1	Introduction	3
2	Installing the host driver	5
2.1	Overview	5
2.2	Checking for prerequisites	5
2.3	Unpacking the downloaded file	7
2.4	Compiling the kernel module	7
2.5	Installing the kernel module	7
2.6	Copying the udev rule file	8
2.7	Loading and unloading the module	8
3	Hello, world	10
3.1	The goal	10
3.2	Prerequisites	10
3.3	The trivial loopback test	11
4	Sample host applications	13
4.1	General	13
4.2	Compilation and editing	14
4.3	Execution	15

4.4	Memory interface	16
5	Xillinux notes	17
5.1	General	17
5.2	Compiling the Linux kernel	17
5.3	Compiling kernel modules	18
5.4	Sound support	19
5.4.1	General	19
5.4.2	Usage details	19
5.4.3	Related boot scripts	20
5.4.4	Accessing /dev/xillybus.audio directly	20
5.4.5	Pulseaudio details	21
5.5	The OLED utility	21
6	Troubleshooting	23
A	A short survival guide to Linux command line	24
A.1	Some keystrokes	24
A.2	Getting help	25
A.3	Showing and editing files	25
A.4	The root user	26
A.5	Selected commands	27

1

Introduction

This is a walkthrough guide for installing the driver for Xillybus on a Linux host, as well as trying out the basic functionality of the IP core. There is also a separate section for getting around in the Xilinx distribution for the Zedboard.

For the sake of simplicity, the host is assumed to be a fullblown computer with native compilation capabilities. The flow for embedded platforms is similar, with certain straightforward differences (e.g. cross compilation).

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 (except for Xilinx, which uses the AXI bus instead).

The steps for reaching this are outlined in one of these documents (depending on the chosen FPGA):

- [Getting started with the FPGA demo bundle for Xilinx](#)
- [Getting started with the FPGA demo bundle for Altera](#)
- [Getting started with Xilinx for Zynq-7000 EPP](#)

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 `/dev/xillybus_something`.

More in-depth information on topics related to the host can be found in [Xillybus host application programming guide for Linux](#).

2

Installing the host driver

2.1 Overview

Installing the Xillybus driver on a Linux machine consists of the following steps:

- Checking for prerequisites (the compiler and kernel headers are installed)
- Unpacking the downloaded file
- Compiling the kernel module
- Installing the kernel module
- Copying the udev configuration file to make the Xillybus device files accessible to any user

This is done using the command-line interface ("Terminal"). The short Linux survival guide in appendix [A](#) may be helpful for those less experienced with this interface.

IMPORTANT:

The Xillybus host driver is already installed in Xillinux, so Zedboard users may skip this section.

2.2 Checking for prerequisites

It's common that Linux installations lack the basic tools for compiling a kernel module. Since this task is common, how-to guides specific to each Linux distribution can be found on the web.

The simplest way to know if these tools exist is to attempt running them. For example, type "make coffee" on command prompt. This is the correct response:

```
$ make coffee

make: *** No rule to make target `coffee'.  Stop.
```

Even though this is an error, we can see that the "make" utility exists. This is what we expect if make needs to be installed:

```
$ make coffee
bash: make: command not found
```

The C compiler is needed as well. Type "gcc", with the desired response:

```
$ gcc
gcc: no input files
```

An error message again, but not "command not found".

On top of these two tools, the kernel headers need to be installed as well. This is a bit more difficult to check up. The common way to know these files are missing is when the kernel compilation fails with an error saying some header file is missing.

On Fedora, RHEL, CentOS and other Red Hat derivatives, the following command is likely to get the computer prepared:

```
# yum install gcc make kernel-devel-$(uname -r)
```

On Ubuntu and other Debian-based distributions:

```
# apt-get install gcc make kernel-devel-$(uname -r)c
```

IMPORTANT:

These installation commands must be issued as root. Those not familiar with this concept are urged to understand the meaning of this, among others in appendix paragraph [A.4](#)

2.3 Unpacking the downloaded file

After downloading the driver from Xillybus' site, change directory to where the downloaded file is. At command prompt, type (excluding the \$ sign)

```
$ tar -xzf xillybus.tar.gz
```

There should be no response, just a new command prompt.

2.4 Compiling the kernel module

Change directory to where the module's source is:

```
$ cd xillybus/module
```

and type "make" to compile the module. The session should look something like this:

```
$ make
make -C /lib/modules/2.6.35.4/build SUBDIRS=/home/myself/xillybus/module modules
make[1]: Entering directory `/usr/src/kernels/2.6.35.4'
  CC [M]  /home/myself/xillybus/module/xillybus.o
  Building modules, stage 2.
 MODPOST 1 modules
  CC      /home/myself/xillybus/module/xillybus.mod.o
  LD [M]  /home/myself/xillybus/module/xillybus.ko
make[1]: Leaving directory `/usr/src/kernels/2.6.35.4'
```

The details may vary slightly, but no errors or warnings should appear.

Note that the kernel module is compiled specifically for the kernel running during the compilation. If another kernel is targeted, type "make TARGET=kernel-version" where "kernel-version" is the targeted kernel version, as it appears in /lib/modules/.

2.5 Installing the kernel module

Staying in the same directory, become root and type "make install". This can take a few seconds, but shouldn't generate any errors. If this fails, copy the xillybus.ko directory to some existing kernel driver subdirectory, and run depmod as follows (assuming that the target kernel is 2.6.35.4):

```
# cp xillybus.ko /lib/modules/2.6.35.4/kernel/drivers/char/

# depmod -a
```

The installation does not load the module into the kernel immediately. It will do so on the next boot of the system if a Xillybus peripheral is discovered. How to load the module manually is shown in paragraph [2.7](#).

2.6 Copying the udev rule file

By default, Xillybus device files are accessible only by their owner, which is root. It makes a lot of sense to make these files accessible to any user, so that working as root can be avoided. The udev mechanism changes these files permissions when the device files are generated by following a set of rules.

So, in the same directory, and still as root, copy the udev rule file to where all rules are kept, most likely /etc/udev/rules.d/, with

```
# cp 10-xillybus.rules /etc/udev/rules.d/
```

The content of the file is simply:

```
SUBSYSTEM=="xillybus", MODE="666", OPTIONS="last_rule"
```

which means that any Xillybus device driver generated should be given permission mode 0666, which means read and write allowed to anyone.

Note that it's possible to keep the owner-only permissions and change the owner's ID instead, by altering the content of the rule file.

2.7 Loading and unloading the module

In order to load the module (and start working with Xillybus), type as root:

```
# modprobe xillybus
```

This will make the Xillybus device files appear.

Note that this should not be necessary if a Xillybus peripheral was present when the system was booted and the driver was already installed as described above.

To see a list of modules in the kernel, type "lsmod". To remove the module from the kernel, go


```
# rmmmod xillybus
```

This will make the device files vanish.

If something seems to have gone wrong, please check up the `/var/log/messages` log file for messages containing the word "xillybus". Valuable clues are often found in this log file.

If no `/var/log/messages` log file exists, it's probably `/var/log/syslog` instead.

3

Hello, world

3.1 The goal

Xillybus is a development kit. As such, its merits are best discovered by working with it in a real design. Accordingly, the initial FPGA code includes the simplest possible implementation, as a basis to work on. There are two FIFOs connected between two pairs of interfaces with the core. Any data sent to the core is stored in those FIFOs and looped back to the host. A RAM is also connected to the core, demonstrating addressed access.

The tests shown below merely trigger off communication with the FPGA. It's recommended to make slight changes in the FPGA as well as attach custom logic to the FIFOs, for a better experience of how the system works.

3.2 Prerequisites

- The FPGA loaded with the original evaluation code, as explained in [Getting started with the FPGA demo bundle for Xilinx](#) or [Getting started with the FPGA demo bundle for Altera](#). (this is implicitly the case on Zynq platforms loaded with Xilinx. See [Getting started with Xilinx for Zynq-7000 EPP](#)).
- Except for Zynq platforms: The computer booted properly and the PCIe interface detected (this can be checked with “lspci”).
- Being fairly comfortable with UNIX/Linux command-line interface. [Appendix A](#) may help a bit on this.

3.3 The trivial loopback test

The idea about this test is to verify that the loopback indeed works. The easiest way is using the UNIX command-line utility “cat”.

Open two terminal windows by e.g. double-clicking the “Terminal” desktop item, or look for it in the menus if it’s not present on the desktop.

On the first terminal window type at command prompt (don’t type the dollar sign, it’s the prompt):

```
$ cat /dev/xillybus_read_8
```

This makes the “cat” program print out anything it reads from the xillybus_read_8 device file. Nothing is expected to happen at this stage.

On the second terminal window, type

```
$ cat > /dev/xillybus_write_8
```

Notice the > redirection sign, which tells “cat” to send anything typed to xillybus_write_8.

Now type some text on the second terminal, and press ENTER. The same text will appear in the first terminal. Nothing is sent to xillybus_write_8 until ENTER is pressed by common UNIX convention.

If an error message was received while attempting these two “cat” commands, please check for typos. Otherwise, check /var/log/messages (or /var/log/syslog on Ubuntu) for messages containing the word “xillybus”, and look if any of these indicate a problem.

Either “cat” can be halted with a CTRL-C. Other trivial file operations will work likewise. For example, with the first terminal still reading, type

```
$ date > /dev/xillybus_write_8
```

in the second terminal.

This 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, `/dev/xillybus_read_32` and `/dev/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.

4

Sample host applications

4.1 General

There are five simple C programs in the Xillybus host driver bundle. The demoapps directory consists of the following files:

- Makefile – This file contains the rules used by the “make” utility to compile the 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 Linux](#).

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 UNIX file access practices, which is discussed further in the [Xillybus host application programming guide for Linux](#).

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 at the C library level.

4.2 Compilation and editing

Those familiar with compiling programs in the Linux environment may skip to the next paragraph: There is nothing unusual in the programs' setting.

First and foremost, change directory to where the C files are:

```
$ cd demoapps
```

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

```
$ make
gcc -g -Wall -O3 memwrite.c -o memwrite
gcc -g -Wall -O3 memread.c -o memread
gcc -g -Wall -O3 streamread.c -o streamread
gcc -g -Wall -O3 streamwrite.c -o streamwrite
gcc -g -Wall -O3 -pthread fifo.c -o fifo
```

The five lines starting with 'gcc' are the invocations of the compiler, generated by the "make" utility. These commands can be used to compile any of the programs individually (but there is no reason to do so. Just use "make").

On some systems, the fifth compilation (of fifo.c) may fail if the library for POSIX threads isn't installed (e.g. in some Cygwin installations). This can be ignored, unless fifo.c is the program of interest.

The "make" 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 "make 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 "make" compiles it as well.

For example, suppose that memwrite.c was copied to a new file, mygames.c. This can be done with the GUI interface or with command line:

```
$ cp memwrite.c mygames.c
```

Now to editing Makefile. There are numerous editors and numerous ways to invoke each of them. For a quick starter, use gedit, which can be invoked with its icon on the desktop, or directly on command prompt. To edit the Makefile, just go

```
$ gedit Makefile &
```

The ampersand ('&') in the end means no waiting until the application finishes to get another command prompt, which is suitable for launching GUI applications, among others.

In the Makefile, there's a line saying

```
APPLICATIONS=memwrite memread streamread streamwrite
```

It really just is a list of names with spaces between them. Add mygames (without the .c suffix) to the list.

4.3 Execution

The simple loopback example shown in paragraph 3.3 can be done with two of the applications. After compiling, type in the first terminal:

```
$ ./streamread /dev/xillybus_read_8
```

This is the part that reads from the device file. Note the need to point at the current directory explicitly with the “./” prefix when choosing the executable.

And then, in the second window (assuming a directory change is necessary):

```
$ cd ~/demoapps  
$ ./streamwrite /dev/xillybus_write_8
```

This will work more or less like with “cat”, only “streamwrite” attempts to work on a character-by-character basis, and not wait for ENTER. This is done in a function called config_console(), which can be disregarded: It's there merely for the immediate effect of typing.

IMPORTANT:

streamread and streamwrite perform their I/O in chunks of 128 bytes to keep the implementation simple. Larger buffers should be used when the data rate matters.

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 FPGA demo bundle 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:

```
$ hexdump -C -v -n 32 /dev/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 /dev/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 /dev/xillybus_mem_8 3 170
```

And read the entire array again:

```
$ hexdump -C -v -n 32 /dev/xillybus_mem_8
00000000  00 00 00 aa 00 00 00 00 00 00 00 00 00 00 00 00 |...1.....|
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

Xillinux notes

5.1 General

Xillinux is a graphical Ubuntu 12.04 LTS-based Linux distribution for the Zedboard (a development board for Xilinx' Zynq-7000 EPP family). Like any Linux distribution, it's mostly a collection of software packages. It also includes some of the hardware logic, in particular the VGA adapter, giving the development board the feel of a desktop computer running Linux.

The distribution is organized for a classic keyboard, mouse and monitor setting. Xillinux is also a kickstart development platform for integration between the device's FPGA logic fabric and plain user space applications running on the ARM processors.

This section contains Linux-related topics that are specific to Xillinux.

5.2 Compiling the Linux kernel

Due to its size, the full Linux kernel is not included in the Xillinux distribution. It can be re-established fairly easily by acquiring the sources on which it is based, and apply a set of patches.

The sources can be downloaded with

```
$ git clone https://github.com/Digilent/linux-digilent.git
```

Xillinux' kernel is based upon the commit tagged v3.3.0-digilent-12.07-zed-beta, but if more recent commits are in place, it's possibly beneficial to go for a later version.

The adjustments for Xillinux are given in `/usr/src/xillinux/kernel-patches` as git patches. They will apply cleanly on Linux version represented by the commit tag mentioned

above, and may work well with other versions as well.

Note that the first patch in the kernel-patches directory adds a .config file, and is there for convenience, even though it's not common practice to add the configuration file with a patch.

The same configuration file, as well as the device tree used, can be found in the /boot directory.

Once the kernel sources are set up, standard kernel compilation practices apply. Please refer to the guide relevant to your working environment. Cross compilation of the kernel is recommended to save time.

5.3 Compiling kernel modules

The Xillinux distribution comes with the running kernel's compilation headers. This is not enough to compile the kernel itself, but allows kernel modules to be compiled natively on the platform.

The standard way to compile a "private" (out-of-tree) kernel module is using a Makefile which invokes the kernel's own build environment, after setting up an environment variable, telling it to compile a specific module.

The minimal Makefile for a *native* compilation (performed by the Zynq processor itself) of a kernel module consisting of a single source file, sample.c, is as follows:

```
ifneq ($(KERNELRELEASE),)
obj-m := sample.o
else
TARGET := $(shell uname -r)
PWD := $(shell pwd)
KDIR := /lib/modules/$(TARGET)/build

default:
    @echo $(TARGET) > module.target
    $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
endif
```

Note that the name of the module, "sample", is mentioned only in the "obj-m" line. This is the only thing that differs from Makefile to Makefile.

Using this Makefile typically results in compilation session as follows (run on the platform itself; this is not a cross compilation):

```
# make
make -C /lib/modules/3.3.0-xillinux-1.1+/build SUBDIRS=/root/sample modules
make[1]: Entering directory `/usr/src/kernels/3.3.0-xillinux-1.1+'
  CC [M]  /root/sample/sample.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /root/sample/sample.mod.o
  LD [M]  /root/sample/sample.ko
make[1]: Leaving directory `/usr/src/kernels/3.3.0-xillinux-1.1+'
```

5.4 Sound support

5.4.1 General

The Zedboard supports sound recording and playback by virtue of an Analog Devices' ADAU1761 chip. The chip is connected to the Zynq device's logic fabric (PL) pins only. Except for its obvious functionality, the sound support package also demonstrates how the Xillybus IP core can be used to transport data, as well as to program a chip through SMBus / I²C.

Xillinux supports sound natively by interfacing dedicated Xillybus streams with the most common toolkit for sound in Linux today, Pulseaudio. As a result, virtually any application requiring a sound card will properly use the board's sound chip as the system's default input and output.

It's also possible to turn the Pulseaudio daemon off, and work directly with the Xillybus streams (/dev/xillybus_audio). This presents a simpler programming interface of just opening a device file, at the cost of losing the native functionality of other applications.

Unlike the common approach for Linux sound cards, there is no dedicated kernel driver (e.g. ALSA) for the sound interface, since the Xillybus host driver handles the data transport anyhow. This has no significance, even not to programs expecting to work with /dev/dsp, since Pulseaudio has the capability of faking this interface with the "padsp" utility.

5.4.2 Usage details

By default, sound is played back to the headphones jack (black) as well as Line Out (green). For recording, only the microphone input (pink) is used, but this can be changed, as described next.

5.4.3 Related boot scripts

The sound-related tasks at system boot are defined in `/etc/init/zedboard-sound.conf` as follows:

```
/usr/bin/pulseaudio -D || true
exec /usr/local/bin/zedboard_sound_setup.pl
```

The first command launches the Pulseaudio daemon as root. This is not recommended practice on a multi-user computer, but since Xillinux runs with root as the default user, there is no point in protecting against escalation of permissions.

This command should be disabled if direct access to `/dev/xillybus_audio` is needed.

`zedboard_sound_setup.pl` is a Perl script, which sets up the audio chip's registers for proper operation. It's fairly straightforward, even for non-Perl programmers, using the `/dev/xillybus_smbus` device file to initiate transactions on the chip's I²C bus.

The script can be edited to achieve a different setup of the audio chip. In particular, if the Line In input is the desired source for recording, the lines

```
write_i2c(0x400a, 0x0b, 0x08);
write_i2c(0x400c, 0x0b, 0x08);
```

should be replaced with

```
write_i2c(0x400a, 0x01, 0x05);
write_i2c(0x400c, 0x01, 0x05);
```

5.4.4 Accessing `/dev/xillybus_audio` directly

The `/dev/xillybus_audio` can be written to directly for playback, or read from for recording. The sample format is 32 bit per audio sample, divided into two 16-bit signed integers in little Endian format. The most significant word corresponds to the left channel.

The sampling rate is fixed at 48000 Hz

A Windows WAV file with this sampling rate is likely to play correctly if written directly to the `/dev/xillybus_audio` (the header will be played back as well for about 1 ms) e.g. with

```
# cat song.wav > /dev/xillybus_audio
```

If the response is

```
-bash: /dev/xillybus_audio: Device or resource busy
```

another process is having the device file open for write, possibly the Pulseaudio daemon. There is no problem having the device file opened for read by one process and for write by another.

5.4.5 Pulseaudio details

Pulseaudio interacts with the `/dev/xillybus_audio` device file through a couple of dedicated Pulseaudio modules, `module-file-sink` and `module-file-source`. Their sources can be found in Xillinux' file system at `/usr/src/xillinux/pulseaudio/`.

These are slight modifications of standard Pulseaudio modules for using UNIX pipes as data sinks and sources (`module-pipe-sink` and `module-pipe-sink`).

The modules are automatically loaded when Pulseaudio starts, by virtue of the two following lines in `/etc/pulse/default.pa`:

```
load-module module-file-sink file=/dev/xillybus_audio rate=48000
load-module module-file-source file=/dev/xillybus_audio rate=48000
```

These modules are selected as the system's sound interface automatically, as there are no other alternatives.

5.5 The OLED utility

By default, Xillinux starts an activity meter utility on boot, displaying approximate CPU usage percentage and an indication of the I/O rate on the SD flash disk.

The CPU percentage is based upon `/proc/stat`, where all time not spent idle is considered used CPU time.

An estimation of the SDIO traffic is made based upon the rate at which interrupts are sent to the respective driver. There is no known figure for a full utilization of this resource. Rather, the utility displays 100% for an interrupt rate that appears to be maximal according to measurements.

To display the graphical output on the board's OLED, the bitmap is sent to `/dev/zed.oled`, which is created by Digilent's driver. It's worth to mention that this driver sends SPI data to the OLED device with a bit-banging mechanism, toggling the clock and data in

software. Hence there is a slight CPU consumption sending 512 bytes this way several times per second, but the impact to the overall system performance is minimal.

To change the parameters of this utility, edit `/etc/init/zedboard.oled.conf` or delete this file to disable its launch at system boot altogether.

This file boils down to the following command

```
/usr/local/bin/zedboard_oled /proc/irq/$irqnum/spurious 4 800
```

The application, `zedboard.oled`, takes three arguments:

- The `/proc` file to monitor for SDIO-related interrupts. `$irqnum` is the IRQ number of the `mmc0` device.
- The rate at which the OLED display is updated, in times per second.
- What is considered to be 100% SDIO interrupt rate, in interrupts per second. The current figure was found by trial and error.

6

Troubleshooting

The driver for Xillybus was designed to produce meaningful log messages. It is therefore recommended to look for messages including the word “xillybus” in `/var/log/messages` (or `/var/log/syslog` in some systems) 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.

A list of messages and their explanation can be found 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.

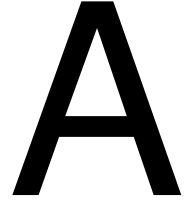
Please don't hesitate seeking assistance through email, if a problem is not resolved quickly.

There is a known issue with old host motherboards, or otherwise immature hardware with respect to the PCI Express bus. There have been reported problems with activating the MSI (Message Signaled Interrupts) on such hardware, because of silicon bugs, or the Linux driver refusing to activate this feature as a precaution against buggy hardware.

Some Linux distributions, in particular from around 2009, have the MSI feature turned off by default, to avoid problems with buggy hardware.

In most cases, MSI-related problems were solved by instructing the kernel to activate MSI interrupts by adding “`pci=msi`” to the kernel boot parameters.

MSI-related problems are becoming increasingly rare as PCI Express evolves to be the next standard for peripherals.



A short survival guide to Linux command line

People who are not used to the command line interface may find it a bit difficult to get things done on a Linux machine. Since the basic setting has been the same for more than 20 years, there are plenty of online tutorials about how to use each and every command. This short guide is only a starter.

A.1 Some keystrokes

This is a summary of the most commonly used keystrokes.

- CTRL-C: Stop the currently running application
- CTRL-D: Finish this session (close terminal window)
- CTRL-L: Clear screen
- TAB : At command prompt, attempt to autocomplete the currently written item. Useful with long file names: Type the beginning of the name, and then [TAB].
- Up and down arrows: At command prompt, suggests previous commands from the command history. Useful for repeating something just done. Editing of previous commands is possible as well, so it's also good for doing almost the same as a previous command.
- space: When computer displays something with a terminal pager, [space] means "page down".
- q: "Quit". In page-by-page display, "q" is used to quit this mode.

A.2 Getting help

Nobody really remembers all the flags and options. There are two common ways to get some more help. One is the “man” command, and the second is the help flag.

For example, to know more about the “ls” command (list files in current directory):

```
$ man ls
```

Note that the ‘\$’ sign is the command prompt, which the computer prints out to say it’s ready for a command. In most cases, the prompt is longer, and includes some information about the user and current directory.

The manual page is shown with a terminal pager. Use [space], arrow keys, Page Up and Page Down to navigate, ‘q’ to quit.

For a shorter command summary, run the command with the `--help` flag. Some commands respond to `-h` or `-help` (with a single dash). Other print the help information any time the syntax isn’t proper. It’s trial and error. For the `ls` command:

```
$ ls --help
Usage: ls [OPTION]... [FILE]...
List information about the FILES (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort.

Mandatory arguments to long options are mandatory for short options too.
  -a, --all                do not ignore entries starting with .
  -A, --almost-all        do not list implied . and ..
      --author              with -l, print the author of each file
...
(and it goes on)
```

A.3 Showing and editing files

If the file is expected to be short (or if it’s OK to use the terminal window’s scrollbar), have its content displayed on the console with

```
$ cat filename
```

Longer files require a terminal pager:

```
$ less filename
```

As for editing text files, there are a lot of editors to choose from. The most popular (but not necessarily easy to start off with) are emacs (and Xemacs) as well as vi. The vi editor is known to be very difficult to learn, but due to its bare-bones simplicity, it's always available and always works.

The recommended simple GUI editor is gedit. It can be started through the desktop menus or from the command line:

```
$ gedit filename &
```

The ampersand at the end ('&') means that the command should be executed "in the background". Or simply put, the next command prompt appears before the command has completed. GUI applications are best started like this.

The 'filename' in these examples can be a path as well, of course. For example, viewing the system's main log file is always:

```
# less /var/log/messages
```

A capital G within "less" jumps to the end of the file.

Note that the log files are accessible only by the root user.

A.4 The root user

All UNIX machines, Linux included, have a user with ID 0 and name "root", also known as the superuser. The special thing about this user is that it's allowed everything. The common limitations on accessing files, resources and certain operations, which are enforced based upon which user you are, are not imposed on the root user.

This is not just an issue of privacy on a multi-user computer. Being allowed everything includes wiping the hard disk in the raw data level by a simple command at shell prompt. It includes several other ways to mistakenly delete data, trash the system in general, or make the system vulnerable to attacks. The basic assumption in any UNIX system is that whoever has root access, knows what he or she is doing. The computer doesn't ask root are-you-sure questions.

Working as root is necessary for system maintenance, including software installation. The key to not messing things up is thinking before pressing ENTER, and being sure the command was typed exactly as required. Following installation instructions exactly

is usually safe. Don't make any modifications without understanding what they're doing exactly. If the same command can be repeated as a non-root user (possibly involving other files), try it out to see what happens as non-root.

Because of the dangers of being root, the common way to run a command as root is with the `sudo` command. For example, view the main log file:

```
$ sudo less /var/log/messages
```

This doesn't always work, because the system needs to be set up to allow the certain user this special mode. The second method is to use "su" for a single command (note the single quotes)

```
$ su -c 'less /var/log/messages'
Password:
```

The system requires the root user's password. In the UNIX world, it's the computer's holy grail due to the permissions it grants to whoever knows it.

And finally, one can simply type "su" and start a session in which every command is given as root. This is convenient when several tasks need to be done as root, but it also means there is a bigger chance of forgetting being root, and write the wrong thing without thinking. Keep root sessions short, or discover why other people do so the hard way.

```
$ su
Password:
# less /var/log/messages
```

The change in the command prompt indicates the change of identity from a regular user to root. In case of doubt, type "whoami" to obtain your current user name.

A.5 Selected commands

And finally, here are a few commonly used UNIX commands.

A few file manipulation commands (it's possibly better to use GUI tools for this):

- `cp` – Copy file(s)
- `rm` – Remove file. Or files. Or whatever.

- mv – Move file(s)
- rmdir – Remove directory. This one is actually safe.

And some which are recommended to begin with

- ls – List all files in the current directory (or another one, when specified). 'ls -l' lists them with their attributes.
- lspci – List all PCI (and PCIe) devices on the bus. Useful for telling if Xillybus is on board. Also try lspci -v, lspci -vv and lspci -n.
- cd – Change directory
- pwd – Show current directory
- cat – Send the file(s) in the argument(s) to standard output. Or use standard input if no argument is given. The original purpose of this command was to concatenate files, but it ended up as the Swiss knife for plain input-output from and to files.
- man – Show manual page on a certain command. Also try “man -a” (sometimes there’s more than one manual page for a command).
- less – Terminal pager. Shows a file or data from standard input page by page. See above. Also used to page the output of a command. For example,

```
$ ls -l | less
```

- head – Show the beginning of a file
- tail – Show the end of a file. Or even better, with the -f flag: show the end + new lines as they arrive. Good for log files, for example (as root):

```
# tail -f /var/log/messages
```

- diff – compare two text files. If it says nothing, files are identical.
- cmp – compare two binary files. If it says nothing, files are identical.
- hexdump – Show the content of a file in a neat hex format. Flags -v and -C are preferred.

- `df` – Show the mounted disks and how much space there is left in each. Even better, “`df -h`”
- `make` – Attempt to build (compile) a project, following the rules in the Makefile
- `gcc` – The GNU C compiler.
- `ps` – Get a list of running processes. “`ps a`”, “`ps au`” and “`ps aux`” will supply different amounts of information. Usually too much of it.

And a couple of advanced commands:

- `grep` – Search for a pattern in a file or standard input. The pattern is a regular expression, but if it's just text, then it searches for the string. For example, search for the word “xillybus”, case insensitive in the main log file, and paginate the output:

```
# grep -i xillybus /var/log/messages | less
```

- `find` – Find a file. It has a complicated argument syntax, but it can find files depending on their name, age, type or anything you can think of. See the man page.