Getting started with Xillinux for Zynq-7000 v2.0

Xillybus Ltd.

www.xillybus.com

Version 4.0

1

Introduction

1.1 The Xillinux distribution

Xillinux is a complete, graphical, Lubuntu 16.04-based Linux distribution for the Zynq-7000 device, intended as a platform for rapid development of mixed software / logic projects. The currently supported boards are Z-Turn Lite, Zedboard, MicroZed and Zybo.

Like any Linux distribution, Xillinux is a collection of software which supports roughly the same capabilities as a personal desktop computer running Linux. Unlike common Linux distributions, Xillinux also includes some of the hardware logic, in particular the VGA adapter.

With Z-Turn Lite, Zedboard and Zybo, the distribution is organized for a classic keyboard, mouse and monitor setting. It also allows command-line control from the USB UART port, but this feature is made available mostly for solving problems.

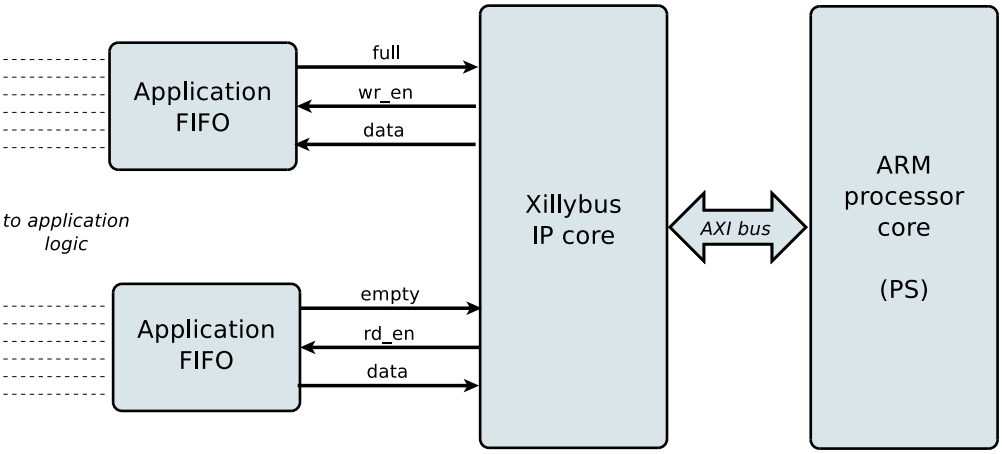
When used with MicroZed, which lacks a VGA/DVI output, only the USB UART is used as a console.

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 processor. With its included Xillybus IP core and driver, no more than basic skills in programming and logic design are needed to complete the design of an application where FPGA logic and Linux-based software work together.

The bundled Xillybus IP cores eliminate the need to deal with the low-level internals of kernel programming as well as the interface between application logic and the processor, by presenting a simple and yet efficient working environment to the application designers.

1.2 The Xillybus IP core

Xillybus is a DMA-based end-to-end solution for data transport between an FPGA and a host that runs Linux or Microsoft Windows. It offers a simple and intuitive interface to the designer of the FPGA logic as well as the programmer of the software. It's available for personal computers and embedded systems using the PCI Express bus as the underlying transport, as well as ARM-based processors, interfacing with the AMBA bus (AXI3/AXI4).



As shown above, the application logic on the FPGA only needs to interact with standard FIFOs.

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

On its other side, the Xillybus IP core implements the data flow utilizing the AXI bus, generating DMA requests on the processor core's bus.

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

The IP core is built instantly per customer's spec, using an online web application. The number of streams, their direction and other attributes are defined by customer to achieve an optimal balance between bandwidth performance, synchronization, and simplicity of design.

After going through the preparation as described in this guide, it's recommended to build and download your custom IP core at http://xillybus.com/custom-ip-factory.

This guide explains how to rapidly set up the Xillinux distribution, with a Xillybus IP core included. This IP core can be attached to user-supplied data sources and data consumers, for real application scenario testing. It's not a demonstration kit, but a fully functional starter design, which can perform useful tasks as is.

Replacing the existing IP core with one that is tailored for special applications is a quick process, and requires the replacement of one binary file and the instantiation of one single module.

More information about using the Xillybus IP core can be found in these documents:

- Getting started with Xillybus on a Linux host
- Xillybus host application programming guide for Linux
- The guide to defining a custom Xillybus IP core

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

2

Prerequisites

2.1 Hardware

Xillinux, which is the Linux distribution by Xillybus for Zynq, currently supports the following boards:

- Z-Turn Lite (along with any Zynq device it's available with)
- Zedboard
- 7010 MicroZed. 7020 MicroZed is also supported with a minor fix: See section 5.5.
- Zybo

If you're about to purchase a Z-Turn Lite board, it's recommended to visit this web page on this board for help with selecting the items.

Owners of boards not listed above may run the distribution on their own hardware, but certain changes, some of which may be nontrivial, may be necessary. More about this in section 5.2.

In order to use the board (MicroZed excluded) as a desktop computer with a monitor, keyboard and mouse, the following items are required:

- A monitor capable of displaying VESA-compliant 1024x768 @ 60Hz with an analog VGA or HDMI input, depending on the board's output (i.e. virtually any PC monitor).
- A VGA or HDMI cable for the monitor (as applicable)
- A USB keyboard

- A USB mouse
- A USB hub, if the keyboard and mouse are not combined in a single USB plug

Note that the Z-Turn Lite board doesn't have an output to a monitor. Therefore, the Z-Turn Lite IO Cape board, which has an HDMI port, must be attached to it for a desktop usage scenario.

When Zybo is used, the monitor can be connected to the HDMI port as well as the VGA port. Zedboard's HDMI output port is not supported.

A wireless keyboard / mouse combo is recommended, since it eliminates the need for a USB hub, and prevents possible physical damage to the USB port on the board, as a result of accidentally pulling the USB cables.

On the Zedboard and Z-Turn Lite, the connection of the keyboard and mouse is done through a Micro B to Type A female USB cable, which arrives with the Zedboard and possibly with Z-Turn Lite (depending on the purchased item).

On the other two boards, a standard USB type A female connector (like a PC's USB plug) is available for connection of peripherals.

Also required:

- A reliable SD card (for Zedboard) or MicroSD (for Z-Turn Lite, MicroZed and Zybo) with 4GB or more, most preferably a card manufactured by Sandisk. The card that (possibly) came with the board is not recommended, as problems have been reported using it with Xillinux.
- Recommended: A USB adapter between an (Micro)SD card and PC, for writing the image and boot file to the card. This may be unnecessary if the PC computer has a built-in slot for SD cards. The Zynq board itself can also be used to write to the SD card, but this is somewhat more difficult.

2.2 Downloading the distribution

The Xillinux distribution is available for download at Xillybus site's download page:

http://xillybus.com/xillinux/

The distribution consists of two parts, that are downloaded as two separate files:

- A raw image of the (Micro)SD card consisting of the file system to be seen by Linux at bootup

- The boot partition kit, which is a set of files for running the implementation with Xilinx' tools, in order to populate the boot partition.

More about this in section 3.

The distribution includes a demo of the Xillybus IP core for easy communication between the processor and logic fabric. The specific configuration of this demo bundle may perform relatively poorly on 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 application. Please visit http://xillybus.com/custom-ip-factory for using this tool.

Any downloaded bundle, including the Xillybus IP core, and the Xillinux distribution, is free for use, as long as this use reasonably matches the term "evaluation". This includes incorporating the IP core in end-user designs, running real-life data and field testing. There is no limitation on how the IP 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

Vivado 2014.4 **and later** can be used for compilation of the logic design in the Xillinux distribution.

If 7z007s or 7z014s devices are intended for use, a Vivado revision that supports these is required (e.g. Vivado 2016.4 and later).

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

Any of Vivado's editions is suitable, including

- the WebPack Edition, which can be downloaded and used with no license fee for an unlimited time, assuming that the intended device is covered. All devices that go with Z-Turn Lite, Zedboard, MicroZed and Zybo are covered by this edition.
- Design Edition, which requires a purchased license (but a 30-day trial is available).
- any edition that may have been licensed specifically with a purchased board, and is therefore limited to a certain Zynq device.
- the System and any other edition offering a superset of features of the WebPack Edition's, are fine as well.

All of these editions cover the Xilinx-supplied IP cores necessary to implement Xillybus for Zynq, with no extra licensing required.

2.4 Experience with FPGA design

When using Z-Turn Lite, Zedboard, MicroZed or Zybo, no previous experience with FPGA design is necessary to have the distribution running on the platform. Using another board requires some knowledge with using Xilinx' tools, and possibly some basic skills related to the Linux kernel.

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

3

Building Xillinux

3.1 Overview

The Xillinux distribution is intended as a development platform, and not just a demo: A ready-for-use environment for custom logic development and integration is built during its preparation for running on hardware. Accordingly, the preparation time for the first test run is a bit long (typically 30 minutes, most of which consist of waiting for Xilinx' tools). However this long preparation shortens the cycle for integrating custom logic.

For a successful boot process of the Xillinux distribution from an (Micro)SD card, it must have two components:

- A FAT32 filesystem in the boot partition, consisting of boot loaders, a configuration bitstream for the FPGA part (known as PL), and the binary files for a boot of the Linux kernel.
- An ext4 root file system mounted by Linux.

The downloaded raw image of Xillinux has almost everything set up already. There are just three files missing in its boot partition, one of which needs to be generated with Xilinx' tools, and two that are copied from the boot partition kit.

The various operations for preparing the (Micro)SD are detailed step by step in this section.

This procedure consists of the following steps, which must be done in the order outlined below.

- Unzipping the boot partition kit
- Running the implementation of the main PL (FPGA) project

- Writing the Xillinux image to the (Micro)SD card
- Copying three files into the (Micro)SD card's boot partition

How to work with other boards is discussed in paragraph 5.2.

3.2 Unzipping the boot partition kit

Unzip the previously downloaded xillinux-eval-*board-XXX*.zip file into a working directory.

IMPORTANT:

The path to the working directory must not include white spaces. In particular, the Windows Desktop is unsuitable, since its path includes "Documents and Settings".

The bundle consists of the following directories (or some of them):

- verilog – Contains the project file for the main logic and some sources in Verilog (in the 'src' subdirectory)
- vhdl – Contains the project file for the main logic and some sources files. The file in VHDL to edit is in the 'src' subdirectory
- blockdesign – This directory contains the files related to the Block Design Flow (see The guide to Xillybus Block Design Flow for non-HDL users).
- cores – Precompiled binaries of the Xillybus IP cores
- system – Directory for generating processor-related logic
- runonce – Directory for generating general-purpose logic. May not exist in some bundles.
- bootfiles – Contains two board-specific files, to be copied to the boot partition.
- vivado-essentials – Definition files and build directories for processor-related and general-purpose logic for use by Vivado.

There are bundles available for Z-Turn Lite, Zedboard, MicroZed and Zybo boards. If another board is used, the constraints file, vivado-essentials/xillydemo.xdc, must be edited accordingly, in addition to the issues listed in section 5.2.

Note that the vhdl directory contains Verilog files, but none of them should need editing by user.

The interface with the Xillybus IP core 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 data consumers.

3.3 Generating the bitstream file

3.3.1 Introduction

Vivado generates many intermediate files in a relatively complex structure, which makes it difficult to keep the project under 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 during the project's implementation.

The implementation of the project can be based upon Verilog, VHDL, or the Block Design Flow, (the latter explained in The guide to Xillybus Block Design Flow for non-HDL users).

3.3.2 Selecting the intended Zynq part (Z-Turn Lite only)

This step is required **only** with the demo bundle for Z-Turn Lite.

Open select_part.tcl in the bundle's root directory with a text editor. The four last lines of this file are Tcl commands for setting the Zynq part for which the Vivado project is created. These four lines are commented out with a "#" character.

Uncomment one "#" character from one of these lines in order to select the Zynq part on your Z-Turn Lite board.

If you want to use another Zynq part later on, change the Vivado project's setting accordingly and reimplement the project. select_part.tcl is referred to only during the project's generation, so making changes to it afterwards has no effect.

3.3.3 Preparing xillydemo.vhd (VHDL project only)

This step is required only if VHDL is chosen, and the demo bundle is intended for any board **except** Z-Turn Lite.

vhdl/src/xillydemo.vhd must be edited to **remove** the following three lines in the beginning of Xillydemo's entity port list:

```
    PS_CLK : IN std_logic;
    PS_PORB : IN std_logic;
    PS_SRSTB : IN std_logic;
```

also, **uncomment** the following lines in the architecture definition (remove the "--" comment marks):

```
  -- signal PS_CLK :  std_logic;
  -- signal PS_PORB :  std_logic;
  -- signal PS_SRSTB :  std_logic;
```

There is no need to make changes in any Verilog source file.

3.3.4 Generating the Vivado project

Start **Vivado 2014.4 or later**.

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 the bottom of Vivado's window, and verify that it says

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

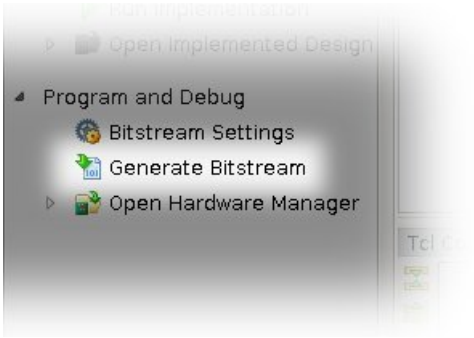
If this is not the last line of the Tcl console's output, something went wrong.

There can be Critical Warnings 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:
```

3.3.5 Implementation of the project

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, some of which may be classified critical. There should be no errors.

A popup window will appear, informing that the bitstream was generated successfully, 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 few error conditions worth mentioning:

- The Placer fails with “IO placement is infeasible” on a VHDL design. If this happens on an implementation with VHDL, please make sure that xillydemo.vhd was edited as required above.
- write_bitstream fails with DRC errors complaining about PS_CLK, PS_PORB and PS_SRSTB being unspecified, unrouted and unconstrained, then again – please make sure that xillydemo.vhd was edited as required above.
- Error saying “Timing constraints weren’t met”. This 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 run of route. 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.

- Any other error is most likely a result of changes made by the user, and should be handled per case.

3.4 Loading the (Micro)SD with the image

3.4.1 General

The purpose of this task is to write the image file to the (micro)SD card. This file's name is xillinux-2.0.img.gz, and it's downloaded as a compressed file (in gzip format).

This image of the (Micro)SD card, is ready for boot, except for three files, which are added after writing it to the card.

This image should be uncompressed and then written to the (Micro)SD card's first sector and on. There are several ways and tools to accomplish this. A few methods are suggested next.

The image contains a partition table, a partly populated FAT file system for placing initial boot files, and the Linux root file system of ext4 type. The second partition is ignored by almost all Windows computers, so the (Micro)SD card may appear to be very small in capacity (16 MB or so).

Writing a full disk image is not an operation intended for plain computer users, and therefore requires special software on Windows computers and extra care on Linux. The following paragraphs explain how to do this on either operating system.

If there's no USB adapter for the (Micro)SD card (or a dedicated slot on a computer), the board itself can be used to write the image, as described in paragraph 3.4.4.

IMPORTANT:

Writing an image to the (Micro)SD irrecoverably deletes any previous content it may contain. It's warmly recommended to make a copy of its existing content, possibly with the same tools used to write the image.

3.4.2 Loading the image (Windows)

On Windows, a special application is needed to copy the image, such as the USB Image Tool. This tool is suitable when a USB adapter is used to access the (Micro)SD card.

Some computers (laptops in particular) have an (Micro)SD slot built in, and may need to use another tool, e.g. Win32 Disk Imager. This may also be the case when running Windows 7.

Both tools are available free for download from various sites on the web. The following walkthrough assumes using the USB Image Tool.

For a Graphical interface, run "USB Image Tool.exe". When the main window shows up, plug in the USB adapter, select the device icon which appears at the top left. Make sure that you're in "Device Mode" (as opposed to "Volume Mode") on the top left drop down menu. Click Restore and set the file type to "Compressed (gzip) image files". Select the downloaded image file (xillinux.img.gz). The whole process should take about 4-5 minutes. When finished, unmount the device ("safely remove hardware") and unplug it.

On some machines, the GUI will fail to run with an error saying that the software initialization failed. In that case, the command line alternative can be used, or a Microsoft .NET framework component needs to be installed.

Alternatively, this can be done from the command line (which is a quick alternative if the attempt to run GUI fails). This is done in two stages. First, obtain the device's number. On a DOS Window, change directory to where the application was unzipped to and go (typical session follows):

```
C:\usbimage>usbitcmd l

USB Image Tool 1.57
COPYRIGHT 2006-2010 Alexander Beug
http://www.alexpage.de

  Device   | Friendly Name              | Volume Name | Volume Path | Size
```

```
-------------------------------------------------------------------------------
     2448 | USB Mass  Storage Device  |             | E:\         |   4024 MB
```

(Note that the character after ”usbitcmd” is the letter “l” and not the number “1”)

Now, when we have the device number, we can actually do the writing (”restore”):

```
C:\usbimage>usbitcmd r 2448 \path\to\xillinux.img.gz /d /g

USB Image Tool 1.57
COPYRIGHT 2006-2010 Alexander Beug
http://www.alexpage.de

Restoring backup to "USB Mass  Storage Device USB Device" (E:\)...ok
```

Again, this should take about 4-5 minutes. And of course, change the number 2448 to whatever device number you got in the first stage, and replace `\path\to` with the path to where the (Micro)SD card’s image is stored on your computer.

3.4.3 Loading the image (Linux)

IMPORTANT:

Raw copying to a device is a dangerous task: A trivial human error (typically choosing the wrong destination disk) can result in irrecoverable loss of all data in the computer’s hard disk. Think before pressing Enter, and consider doing this in Windows if you’re not used to Linux.

As just mentioned, it’s important to detect the correct device as the (Micro)SD card. This is best done by plugging in the USB connector, and looking for something like this in the main log file (/var/log/messages or /var/log/syslog):

```
Sep  5 10:30:59 kernel: sd 1:0:0:0: [sdc] 7813120 512-byte logical blocks
Sep  5 10:30:59 kernel: sd 1:0:0:0: [sdc] Write Protect is off
Sep  5 10:30:59 kernel: sd 1:0:0:0: [sdc] Assuming drive cache: write through
Sep  5 10:30:59 kernel: sd 1:0:0:0: [sdc] Assuming drive cache: write through
Sep  5 10:30:59 kernel:  sdc: sdc1
Sep  5 10:30:59 kernel: sd 1:0:0:0: [sdc] Assuming drive cache: write through
Sep  5 10:30:59 kernel: sd 1:0:0:0: [sdc] Attached SCSI removable disk
Sep  5 10:31:00 kernel: sd 1:0:0:0: Attached scsi generic sg0 type 0
```

The output may vary slightly, but the point here is to see what name the kernel gave the new disk. “sdc” in the example above.

Uncompress the image file:

```
# gunzip xillinux.img.gz
```

Copying the image to the (Micro)SD card is simply:

```
# dd if=xillinux.img of=/dev/sdc bs=512
```

You should point at the disk you found to be the flash drive, of course.

IMPORTANT:

/dev/sdc is given as an example. Don’t use this device unless it happens to match the device recognized on your computer.

And verify

```
# cmp xillinux.img /dev/sdc
cmp: EOF on xillinux.img
```

Note the response: The fact that EOF was reached on the image file means that everything else compared correctly, and that the flash has more space than actually used. If cmp says nothing (which would normally be considered good) it actually means something is wrong. Most likely, a regular file “/dev/sdc” was generated rather than writing to the device.

3.4.4 Using the Zynq board for loading the image

Paragraph 3.4.3 above described how to load the image with a Linux machine and a USB adapter. The Zynq board itself can be used, running the sample Linux system that came with the board. Basically, the same instructions can be followed, using /dev/mmcblk0 as the destination device (instead of /dev/sdc).

This works fine when the boot process is done from the QSPI flash, as well as with the sample Linux system on an SD card (if such arrived with the board). This is because unlike Xillinux, it runs completely from RAM, and doesn’t use the SD card after boot has finished. So if an SD card was used for boot, it’s possible to pull it out, and insert another for writing the image to.

How to give the Zynq board access to the image of the (Micro)SD and the boot partition files is a matter of preference and Linux knowledge. There are several ways to do this over the network, but the simplest way is to write these files to a disk-on-key, and connect it to the USB OTG port. Mount the disk-on-key with

```
> mkdir /mnt/usb
> mount /dev/sda1 /mnt/usb
```

The files on the disk-on-key can then be read at /mnt/usb/.

On Zedboard, make sure that the JP2 jumper is installed, so that the USB port is fed with 5V power supply.

3.5 Copying the files into the boot partition

This final stage places the necessary files for boot:

- Copy boot.bin and devicetree.dtb from the boot partition kit's bootfiles/ subdirectory, into the (Micro)SD card's boot partition (the first partition).
- Copy xillydemo.bit that was generated in section 3.3 (from the verilog/ or vhdl/ subdirectory, whichever was chosen).

Before copying these files: If the (Micro)SD's image was just written to the card, unplug the USB adapter and then connect it back to the computer. If the Zynq board was used for writing the raw image, pull the (Micro)SD card from its slot, and reinsert it.

This is necessary to make sure that the computer is up to date with the (Micro)SD card's partition table.

On Linux systems, it may be necessary to manually mount the first partition (e.g /dev/sdb1). Most computers will do this automatically.

For example, when the Zynq board itself is used for this purpose, type

```
> mkdir /mnt/sd
> mount /dev/mmcblk0p1 /mnt/sd
```

and copy the files to /mnt/sd/.

On Windows systems, plugging in the (micro)SD card will reveal a single "disk", with a single file, uImage. This is the correct destination to copy the files to.

When done, unmount the (Micro)SD card properly, and unplug it from the computer, e.g.

```
> umount /mnt/sd
```

or “remove the disk safely” on Windows.

3.6 The files in the boot partition

Before attempting a boot, please verify that the boot partition is populated as follows.

In order for the boot to be successful, four files need to exist in the (micro)SD card’s first partition (the boot partition):

- uImage – The Linux kernel binary. This is the only file in the boot partition after writing Xillinux’ (Micro)SD image to the card. The kernel is board-independent.
- boot.bin – The initial bootloader. This file contains the initial processor initializations and the U-boot utility, and is significantly different from board to board.
- devicetree.dtb – The Device Tree Blob file, which contains hardware information for the Linux kernel.
- xillydemo.bit – The PL (FPGA) programming file, which was generated in section 3.3

4

Kick off boot

4.1 Jumper settings

For the board to perform a boot from the (Micro)SD card, modifications in the jumper settings need to be made. The settings are detailed for each of the boards below.

4.1.1 Zedboard

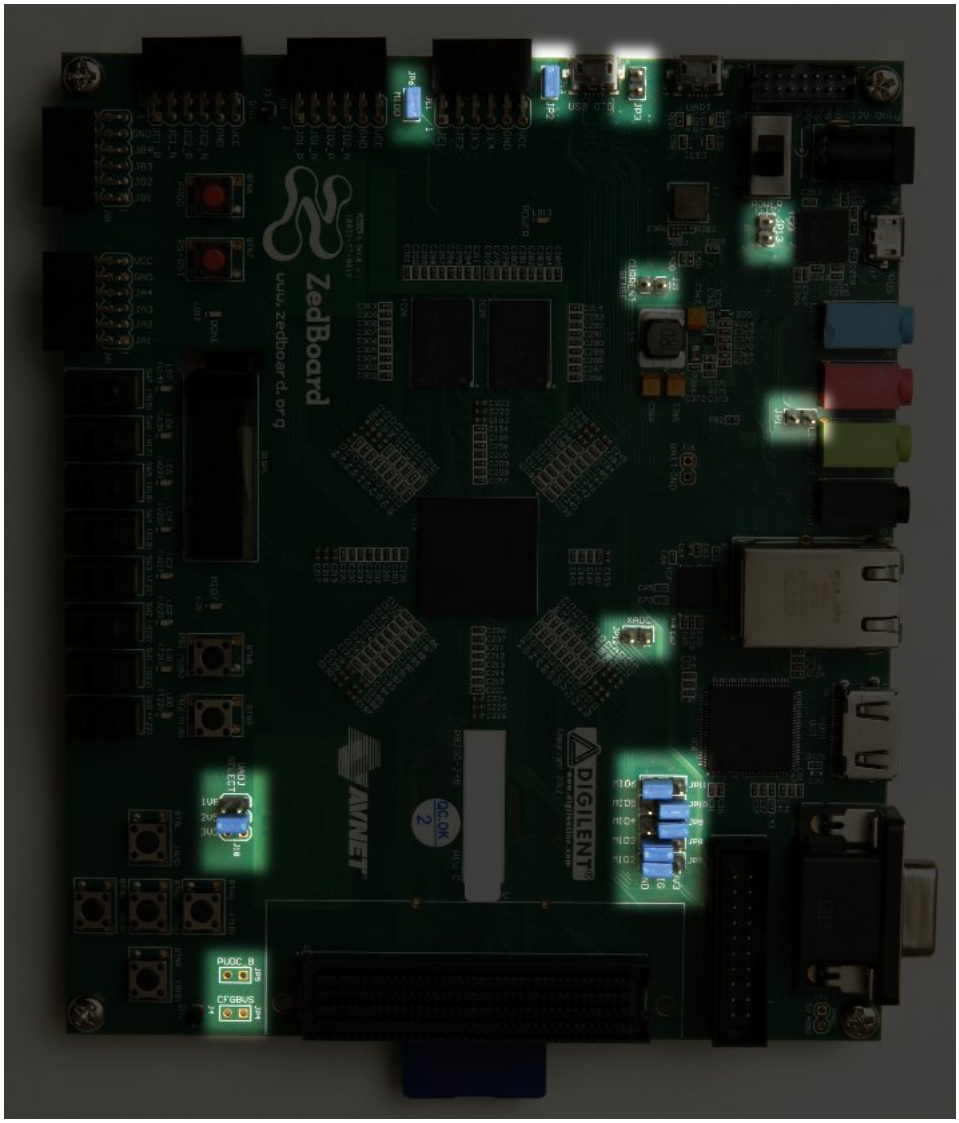
The correct setting is depicted in the image on the following page.

Typically, the following jumper changes are necessary (but your board may be set differently to begin with):

- Install a jumper for JP2 to supply 5V to USB device.
- JP10 and JP9 moved from GND to 3V3 position, the three others in that row are left connected to GND.
- Install a jumper for JP6 (required for CES silicon, see page 34 of the Zedboard's Hardware Guide).

IMPORTANT:

The required setting differs from the one detailed in the Zedboard Hardware User Guide in that JP2 is jumpered, so that the USB devices attached to the board (USB keyboard and mouse) receive their 5V power supply.



Jumper settings highlighted on the Zedboard

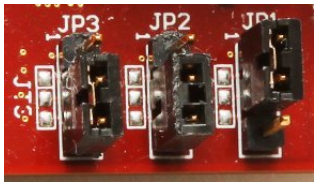
4.1.2 MicroZed

The proper jumper setting to perform boot of Xillinux from the MicroSD card is as follows:

- JP1: 1-2 (GND)
- JP2: 2-3 (VCC)
- JP3: 2-3 (VCC)

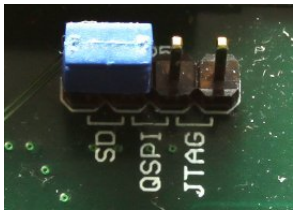
Given a board with the default setting, only JP2 needs to be moved.

The correct jumper setting is depicted here:



4.1.3 Zybo

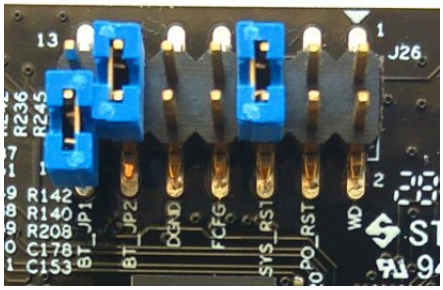
The boot mode is selected by a jumper near to the VGA connector, which should be set on the two pins marked with “SD”, as shown on this image:



The other jumpers are set depending on the desired operating mode. For example, the power supply jumper can be set to take power from an external 5V source, or from the UART USB jack – both are fine for a successful boot of Xillinux.

4.1.4 Z-Turn Lite

The jumpers next to the Ethernet connector (labeled J26) determine the boot mode, and should be set up as follows:



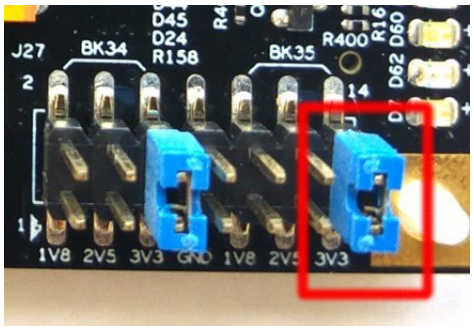
- The BT_JP1 jumper should not be placed (or placed as shown in the picture, i.e. attached to only one of the pins)
- The BT_JP2 jumper should be placed
- FCFG and PO_RST should not be placed

Unrelated to the boot operation, these two jumpers are of some importance:

- The SYS_RST jumper makes it possible to reset the Zynq's processor by pushing the board's K2 button.
- The WD (Watchdog) jumper allows enabling the on-board watchdog chip from the processor. Xillinux carries out boot properly regardless of whether it's placed or not.

The DGND pin pair is just two pins connected to ground. Placing a jumper on these has no effect.

Also, if the HDMI video output is used (via the Cape IO board), the Zynq device's bank 35 must be driven with a 3.3V voltage supply. This is established with the 3V3 jumper on the BK35 group of J27, at the board's corner close to the MicroSD card (see image below).



4.2 Attaching peripherals

The following general-purpose hardware can be attached the board:

- Z-Turn Lite with a Z-Turn Lite IO Cape board: A computer monitor to the Cape board's HDMI connector. Alternatively, A DVI input connected to the board's HDMI plug through an HDMI/DVI adapter or cable.
- Zedboard / Zybo: A computer monitor to the analog VGA connector.
- Zybo: A computer monitor with an HDMI input. Alternatively, A DVI input connected to the board's HDMI plug through an HDMI/DVI adapter or cable.
- Zedboard/Zybo/Z-Turn Lite: A mouse and keyboard to the USB (OTG) connector.

  On Zybo, there's a PC-like USB plug for this. On Zedboard and Z-Turn Lite, this goes through the USB female cable that comes with the Zedboard (which is the shorter one). This cable is also shipped with Z-Turn Lite when purchased in the "kit" configuration.

  The system will carry out a successful boot in the absence of these, and there is no problem connecting and disconnecting the keyboard and mouse as the system runs – the system detects and works with whatever keyboard and mouse it has connected at any given moment.

  Note that on Zedboard, JP2 must be installed for this USB port to function.
- The Ethernet port is optional for common network tasks. The Linux machine configures the network automatically if the attached network has a DHCP server.
- The UART USB port can be connected to a PC, but this is not necessary in most cases with Zedboard and Zybo. Some boot messages are sent there, and a shell prompt is issued on this interface when the boot completes.

  This is useful for debugging failures during boot, or when either a PC monitor or a keyboard is missing or don't work properly.

  For Z-Turn Lite, an external USB adapter is required for interfacing with the board's 3.3V UART signals. This adapter is possibly included with the board, as explained on Xillybus' web page regarding Z-Turn Lite.

4.3 Powering up the board

4.3.1 Initial diagnostics

Failures during boot are rare when the build instructions above have been followed. Among the common reasons are:

- Incorrect jumper settings (see paragraph 4.1).
- Using a (Micro)SD card not made by Sandisk. Even if the card seems to work fine, scattered data corruptions are easily overlooked, but result in errors that appear to have a completely different reason.
- Incomplete or faulty writing of the (Micro)SD image into the card
- Old and inadequate environment variables loaded by U-boot from the board's QSPI flash. See paragraph 4.3.3.
- Not following the instructions, usually due to attempts to tweak the system on the first attempt to build it.

The correct UART setting is 115200 baud, 8 data bits, 1 stop bits, and no flow control. Some text should appear no later than 4 seconds after the board is powered on. The only requirement for this to happen is that the boot.bin file is found in the first (FAT32) partition on the (Micro)SD card.

If nothing happens after powering on the board:

- Verify that the correct boot.bin has been copied from a boot partition kit that matches the board's type. The kit's file name indicates which board it should be used with.
- Verify that the UART to computer link works properly, possibly with the sample Linux on the QSPI flash or on the SD card that (possibly) arrived with the board. Note that Linux, as a host for the UART terminal application, may not work properly with some UART/USB converters, so trying Tera Term under Windows may be the only option.

If U-boot emits messages on the console, but the boot process fails, it may be helpful to compare with the transcript in paragraph 4.3.5. The rest of this section may also contain relevant information for understanding what's wrong.

4.3.2 When the boot process completes

At the end of the boot process, a shell prompt is given on the UART console, with no need to log in manually. Even so, the root user's password is set to nothing, so logging in as root, if ever needed, doesn't require a password.

IMPORTANT:

Unlike the Linux sample on the (Micro)SD card that came with the board, Xillinux' root file system permanently resides on the (Micro)SD card, and is written to while the system is up. The Linux system should be shut down properly before powering off the board to keep the system stable, just like any PC computer, even though a proper recovery is usually observed after sudden power losses.

Notes for Z-Turn Lite, Zedboard and Zybo:

- Type "startx" at shell prompt to launch a LXDE graphical desktop. The desktop takes some 15-30 seconds to initialize. If nothing appears to happen, monitoring the activity meter on the OLED display helps telling if something is going on (Only Zedboard has this OLED display).
- The Xillybus logo screensaver with a white background is present on the screen from the moment the logic fabric is loaded until the Linux kernel launches. It will also show when the operating system puts the screen in "blank" mode, which is a normal condition when the system is idle, or when the X-Windows system attempts to manipulate the graphic mode.
- A Xillybus screensaver on **blue** background, or random blue stripes on the screen indicate that the graphics interface suffers from data starvation. This is never expected to happen, and should be reported, unless an obvious reason is known.

4.3.3 U-boot environment variables

Xillinux relies on U-boot for loading xillydemo.bit, the kernel image and the device tree during the boot process. This utility offers a large variety of boot configurations, and has a simple command-line interface which allows experimentation and modification of the settings.

U-boot's shell is reached by typing any character on the UART console immediately after U-boot starts:

```
U-Boot 2013.07 (Mar 15 2014 - 22:59:21)

Memory: ECC disabled
DRAM:  512 MiB
MMC:   zynq_sdhci: 0
SF: Detected S25FL129P_64K/S25FL128S_64K with page size 64 KiB, total 16 MiB
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   Gem.e000b000
Hit any key to stop autoboot:  1
```

U-boot always attempts to retrieve saved environment variables from the QSPI flash. The warning that says“bad CRC” indicates that no valid data was found, so U-boot reverted to its hardcoded default environment. This is not an error for a boot of Xillinux from the (Micro)SD card, because the Xillinux' U-boot has all environment variables set correctly.

IMPORTANT:

Note that even when the system executes a boot from a (Micro)SD card, the environment variables are taken from the QSPI flash on the board itself. If the QSPI flash contains environment variables matching a different boot scenario, U-boot may fail the boot process, relying on inadequate variables.

When no key is pressed for one second, U-boot continues according to its environment variables (those loaded from the QSPI flash, or the hardcoded default settings). More precisely, it executes the content of the “bootcmd” variable, which says “run $modeboot” by default. “modeboot”, in turn, is set dynamically by U-boot, depending on where U-boot was loaded from, so it says “sdboot” on a regular Xillinux boot. The “sdboot” variable contains a series of commands for carrying out boot process of Xillinux.

U-boot's command-line shell has a “help” command, which lists all commands and their meaning. Some useful commands are

- help *command* – show help on *command*
- env print – print all environment variables' current values

- env set – set a certain environment variable's value
- env default -a – set all environment variables to their hardcoded defaults.
- saveenv – save the current environment variables to QSPI flash (*not* to MicroSD/SD card).

In particular, the two last commands in the list are important when U-boot fails the boot process. If the warning that says "bad CRC, using default environment" is **not** issued by U-boot, it's relying on stored variables. In order to use the default variables (which are correct for Xillinux), go

```
xillinux-uboot> env default -a
## Resetting to default environment
xillinux-uboot> saveenv
Saving Environment to SPI Flash...
SF: Detected S25FL129P_64K/S25FL128S_64K with page size 64 KiB, total 16 MiB
Erasing SPI flash...Writing to SPI flash...done
```

If there were any desirable changes in the stored environment variables, they are erased as well, of course.

4.3.4 Setting a custom Ethernet MAC address

Linux relies on the Ethernet MAC address that it finds on the hardware, which is set by U-boot, depending on a certain environment variable. Since the environment variables are stored on the QSPI flash, the MAC address is persistently bound to the hardware, so even if a specific (Micro)SD card is used on different boards, each board retains its own MAC address.

For example, on the U-boot's shell using the USB UART console:

```
xillinux-uboot> set ethaddr 00:11:22:33:44:55
xillinux-uboot> saveenv
Saving Environment to SPI Flash...
Erasing SPI flash...Writing to SPI flash...done
```

Later on, after recycling the board's power, and letting Linux perform boot automatically:

```
root@localhost:~# ifconfig
```

```
eth1      Link encap:Ethernet  HWaddr 00:11:22:33:44:55
          inet addr:10.1.1.164  Bcast:10.1.1.255  Mask:255.255.255.0
          inet6 addr: fe80::211:22ff:fe33:4455/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:16 errors:0 dropped:0 overruns:0 frame:0
          TX packets:50 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2720 (2.7 KB)  TX bytes:9230 (9.2 KB)
          Interrupt:54 Base address:0xb000
```

(The IP address was given by the DHCP server in the case above)

4.3.5 Sample transcript during boot

For reference, a typical UART transcript during boot is given below. The example is shown for Zedboard, but the differences between the boards are minor. If the boot process fails, an error message will probably indicate what stage went wrong, and possibly why.

```
U-Boot 2013.07 (Aug 10 2014 - 11:28:31)

Zynq PS_VERSION = 0
Memory: ECC disabled
DRAM:  512 MiB
MMC:   zynq_sdhci: 0
SF: Detected S25FL256S_64K with page size 64 KiB, total 32 MiB
In:    serial
Out:   serial
Err:   serial
Net:   Gem.e000b000
Hit any key to stop autoboot:  1  0
Device: zynq_sdhci
Manufacturer ID: 3
OEM: 5344
Name: SL08G
Tran Speed: 50000000
Rd Block Len: 512
SD version 3.0
High Capacity: Yes
Capacity: 7.4 GiB
Bus Width: 4-bit
Booting Xillinux...
reading xillydemo.bit
4045675 bytes read in 295 ms (13.1 MiB/s)
  design filename = "xillydemo.ncd;HW_TIMEOUT=FALSE;UserID=0xFFFFFFFF"
  part number = "7z020clg484"
  date = "2014/03/11"
  time = "22:22:00"
  bytes in bitstream = 4045564
zynq_load: Align buffer at 10006f to 100080(swap 1)
reading uImage
4487928 bytes read in 326 ms (13.1 MiB/s)
reading devicetree.dtb
9531 bytes read in 16 ms (581.1 KiB/s)
## Booting kernel from Legacy Image at 03000000 ...
   Image Name:   Linux-4.4.30-xillinux-2.0
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    4487864 Bytes = 4.3 MiB
   Load Address: 00008000
```

```
   Entry Point:  00008000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 02a00000
   Booting using the fdt blob at 0x2a00000
   Loading Kernel Image ... OK
   Loading Device Tree to 1fb4e000, end 1fb5353a ... OK

Starting kernel ...

Uncompressing Linux... done, booting the kernel.
[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Initializing cgroup subsys cpuset
[    0.000000] Initializing cgroup subsys cpu
[    0.000000] Initializing cgroup subsys cpuacct
[    0.000000] Linux version 4.4.30-xillinux-2.0 (eli@ocho.localdomain) (gcc version 4.7.3 (Sourcery CodeBench Lite 2013.05-40) ) #1 SMP PREEMPT Tue D
[    0.000000] CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=18c5387d
[    0.000000] CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
[    0.000000] Machine model: Xilinx Zynq
[    0.000000] bootconsole [earlycon0] enabled
[    0.000000] cma: Reserved 16 MiB at 0x1e800000
[    0.000000] Memory policy: Data cache writealloc
[    0.000000] PERCPU: Embedded 12 pages/cpu @dfb36000 s18880 r8192 d22080 u49152
[    0.000000] Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 129920
[    0.000000] Kernel command line: console=ttyPS0,115200n8 console=tty0 consoleblank=0 root=/dev/mmcblk0p2 rw rootwait earlyprintk
[    0.000000] PID hash table entries: 2048 (order: 1, 8192 bytes)
[    0.000000] Dentry cache hash table entries: 65536 (order: 6, 262144 bytes)
[    0.000000] Inode-cache hash table entries: 32768 (order: 5, 131072 bytes)
[    0.000000] Memory: 493232K/524288K available (6155K kernel code, 294K rwdata, 2192K rodata, 312K init, 472K bss, 14672K reserved, 16384K cma-reser
[    0.000000] Virtual kernel memory layout:
[    0.000000]     vector  : 0xffff0000 - 0xffff1000   (   4 kB)
[    0.000000]     fixmap  : 0xffc00000 - 0xfff00000   (3072 kB)
[    0.000000]     vmalloc : 0xe0800000 - 0xff800000   ( 496 MB)
[    0.000000]     lowmem  : 0xc0000000 - 0xe0000000   ( 512 MB)
[    0.000000]     pkmap   : 0xbfe00000 - 0xc0000000   (   2 MB)
[    0.000000]     modules : 0xbf000000 - 0xbfe00000   (  14 MB)
[    0.000000]       .text : 0xc0008000 - 0xc082f0c4   (8349 kB)
[    0.000000]       .init : 0xc0830000 - 0xc087e000   ( 312 kB)
[    0.000000]       .data : 0xc087e000 - 0xc08c7840   ( 295 kB)
[    0.000000]        .bss : 0xc08c7840 - 0xc093da38   ( 473 kB)
[    0.000000] Preemptible hierarchical RCU implementation.
[    0.000000]  Build-time adjustment of leaf fanout to 32.
[    0.000000]  RCU restricting CPUs from NR_CPUS=4 to nr_cpu_ids=2.
[    0.000000] RCU: Adjusting geometry for rcu_fanout_leaf=32, nr_cpu_ids=2
[    0.000000] NR_IRQS:16 nr_irqs:16 16
[    0.000000] slcr mapped to e0800000
[    0.000000] L2C: platform modifies aux control register: 0x72360000 -> 0x72760000
[    0.000000] L2C: DT/platform modifies aux control register: 0x72360000 -> 0x72760000
[    0.000000] L2C-310 erratum 769419 enabled
[    0.000000] L2C-310 enabling early BRESP for Cortex-A9
[    0.000000] L2C-310 full line of zeros enabled for Cortex-A9
[    0.000000] L2C-310 ID prefetch enabled, offset 1 lines
[    0.000000] L2C-310 dynamic clock gating enabled, standby mode enabled
[    0.000000] L2C-310 cache controller enabled, 8 ways, 512 kB
[    0.000000] L2C-310: CACHE_ID 0x410000c8, AUX_CTRL 0x76760001
[    0.000000] zynq_clock_init: clkc starts at e0800100
[    0.000000] Zynq clock init
[    0.000000] clocksource: ttc_clocksource: mask: 0xffff max_cycles: 0xffff, max_idle_ns: 537538477 ns
[    0.000018] sched_clock: 16 bits at 54kHz, resolution 18432ns, wraps every 603975816ns
[    0.007925] ps7-ttc #0 at e0808000, irq=17
[    0.012173] sched_clock: 64 bits at 333MHz, resolution 3ns, wraps every 4398046511103ns
[    0.020052] clocksource: arm_global_timer: mask: 0xffffffffffffffff max_cycles: 0x4ce07af025, max_idle_ns: 440795209040 ns
[    0.031309] Console: colour dummy device 80x30
[    0.035629] console [tty0] enabled
[    0.039067] bootconsole [earlycon0] disabled
[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Initializing cgroup subsys cpuset
[    0.000000] Initializing cgroup subsys cpu
[    0.000000] Initializing cgroup subsys cpuacct
[    0.000000] Linux version 4.4.30-xillinux-2.0 (eli@ocho.localdomain) (gcc version 4.7.3 (Sourcery CodeBench Lite 2013.05-40) ) #1 SMP PREEMPT Tue D
[    0.000000] CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=18c5387d
[    0.000000] CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
[    0.000000] Machine model: Xilinx Zynq
[    0.000000] bootconsole [earlycon0] enabled
[    0.000000] cma: Reserved 16 MiB at 0x1e800000
[    0.000000] Memory policy: Data cache writealloc
[    0.000000] PERCPU: Embedded 12 pages/cpu @dfb36000 s18880 r8192 d22080 u49152
```

```
[    0.000000] Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 129920
[    0.000000] Kernel command line: console=ttyPS0,115200n8 console=tty0 consoleblank=0 root=/dev/mmcblk0p2 rw rootwait earlyprintk
[    0.000000] PID hash table entries: 2048 (order: 1, 8192 bytes)
[    0.000000] Dentry cache hash table entries: 65536 (order: 6, 262144 bytes)
[    0.000000] Inode-cache hash table entries: 32768 (order: 5, 131072 bytes)
[    0.000000] Memory: 493232K/524288K available (6155K kernel code, 294K rwdata, 2192K rodata, 312K init, 472K bss, 14672K reserved, 16384K cma-rese
[    0.000000] Virtual kernel memory layout:
[    0.000000]     vector  : 0xffff0000 - 0xffff1000   (   4 kB)
[    0.000000]     fixmap  : 0xffc00000 - 0xfff00000   (3072 kB)
[    0.000000]     vmalloc : 0xe0800000 - 0xff800000   ( 496 MB)
[    0.000000]     lowmem  : 0xc0000000 - 0xe0000000   ( 512 MB)
[    0.000000]     pkmap   : 0xbfe00000 - 0xc0000000   (   2 MB)
[    0.000000]     modules : 0xbf000000 - 0xbfe00000   (  14 MB)
[    0.000000]       .text : 0xc0008000 - 0xc082f0c4   (8349 kB)
[    0.000000]       .init : 0xc0830000 - 0xc087e000   ( 312 kB)
[    0.000000]       .data : 0xc087e000 - 0xc08c7840   ( 295 kB)
[    0.000000]        .bss : 0xc08c7840 - 0xc093da38   ( 473 kB)
[    0.000000] Preemptible hierarchical RCU implementation.
[    0.000000]  Build-time adjustment of leaf fanout to 32.
[    0.000000]  RCU restricting CPUs from NR_CPUS=4 to nr_cpu_ids=2.
[    0.000000] RCU: Adjusting geometry for rcu_fanout_leaf=32, nr_cpu_ids=2
[    0.000000] NR_IRQS:16 nr_irqs:16 16
[    0.000000] slcr mapped to e0800000
[    0.000000] L2C: platform modifies aux control register: 0x72360000 -> 0x72760000
[    0.000000] L2C: DT/platform modifies aux control register: 0x72360000 -> 0x72760000
[    0.000000] L2C-310 erratum 769419 enabled
[    0.000000] L2C-310 enabling early BRESP for Cortex-A9
[    0.000000] L2C-310 full line of zeros enabled for Cortex-A9
[    0.000000] L2C-310 ID prefetch enabled, offset 1 lines
[    0.000000] L2C-310 dynamic clock gating enabled, standby mode enabled
[    0.000000] L2C-310 cache controller enabled, 8 ways, 512 kB
[    0.000000] L2C-310: CACHE_ID 0x410000c8, AUX_CTRL 0x76760001
[    0.000000] zynq_clock_init: clkc starts at e0800100
[    0.000000] Zynq clock init
[    0.000000] clocksource: ttc_clocksource: mask: 0xffff max_cycles: 0xffff, max_idle_ns: 537538477 ns
[    0.000018] sched_clock: 16 bits at 54kHz, resolution 18432ns, wraps every 603975816ns
[    0.007925] ps7-ttc #0 at e0808000, irq=17
[    0.012173] sched_clock: 64 bits at 333MHz, resolution 3ns, wraps every 4398046511103ns
[    0.020052] clocksource: arm_global_timer: mask: 0xffffffffffffffff max_cycles: 0x4ce07af025, max_idle_ns: 440795209040 ns
[    0.031309] Console: colour dummy device 80x30
[    0.035629] console [tty0] enabled
[    0.039067] bootconsole [earlycon0] disabled
[    0.043389] Calibrating delay loop... 1332.01 BogoMIPS (lpj=6660096)
[    0.130990] pid_max: default: 32768 minimum: 301
[    0.131116] Security Framework initialized
[    0.131135] Yama: becoming mindful.
[    0.131211] AppArmor: AppArmor initialized
[    0.131270] Mount-cache hash table entries: 1024 (order: 0, 4096 bytes)
[    0.131295] Mountpoint-cache hash table entries: 1024 (order: 0, 4096 bytes)
[    0.132028] Initializing cgroup subsys io
[    0.132059] Initializing cgroup subsys memory
[    0.132104] Initializing cgroup subsys devices
[    0.132133] Initializing cgroup subsys freezer
[    0.132156] Initializing cgroup subsys net_cls
[    0.132177] Initializing cgroup subsys perf_event
[    0.132200] Initializing cgroup subsys net_prio
[    0.132222] Initializing cgroup subsys pids
[    0.132274] CPU: Testing write buffer coherency: ok
[    0.132537] CPU0: thread -1, cpu 0, socket 0, mpidr 80000000
[    0.132602] Setting up static identity map for 0x82c0 - 0x82f4
[    0.310974] CPU1: thread -1, cpu 1, socket 0, mpidr 80000001
[    0.311078] Brought up 2 CPUs
[    0.311115] SMP: Total of 2 processors activated (2664.03 BogoMIPS).
[    0.311133] CPU: All CPU(s) started in SVC mode.
[    0.312116] devtmpfs: initialized
[    0.314713] evm: security.selinux
[    0.314734] evm: security.SMACK64
[    0.314748] evm: security.SMACK64EXEC
[    0.314761] evm: security.SMACK64TRANSMUTE
[    0.314775] evm: security.SMACK64MMAP
[    0.314804] evm: security.ima
[    0.314824] evm: security.capability
[    0.315239] VFP support v0.3: implementor 41 architecture 3 part 30 variant 9 rev 4
[    0.315600] clocksource: jiffies: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 19112604462750000 ns
[    0.316774] pinctrl core: initialized pinctrl subsystem
[    0.318050] NET: Registered protocol family 16
```

```
[    0.320031] DMA: preallocated 256 KiB pool for atomic coherent allocations
[    0.323590] zynq_gpio e000a000.ps7-gpio: This is the Xillinux-1.3 compliant legacy GPIO driver.
[    0.324184] zynq_gpio e000a000.ps7-gpio: gpio at 0xe000a000 mapped to 0xe0814000
[    0.329041] hw-breakpoint: found 5 (+1 reserved) breakpoint and 1 watchpoint registers.
[    0.329099] hw-breakpoint: maximum watchpoint size is 4 bytes.
[    0.375012] vgaarb: loaded
[    0.377592] SCSI subsystem initialized
[    0.378094] usbcore: registered new interface driver usbfs
[    0.378220] usbcore: registered new interface driver hub
[    0.378369] usbcore: registered new device driver usb
[    0.378741] media: Linux media interface: v0.10
[    0.378849] Linux video capture interface: v2.00
[    0.379225] pps_core: LinuxPPS API ver. 1 registered
[    0.379263] pps_core: Software ver. 5.3.6 - Copyright 2005-2007 Rodolfo Giometti <giometti@linux.it>
[    0.379346] PTP clock support registered
[    0.379675] EDAC MC: Ver: 3.0.0
[    0.383475] NetLabel: Initializing
[    0.383515] NetLabel:  domain hash size = 128
[    0.383538] NetLabel:  protocols = UNLABELED CIPSOv4
[    0.383614] NetLabel:  unlabeled traffic allowed by default
[    0.384003] clocksource: Switched to clocksource arm_global_timer
[    0.384740] AppArmor: AppArmor Filesystem Enabled
[    0.399611] NET: Registered protocol family 2
[    0.400379] TCP established hash table entries: 4096 (order: 2, 16384 bytes)
[    0.400470] TCP bind hash table entries: 4096 (order: 3, 32768 bytes)
[    0.400579] TCP: Hash tables configured (established 4096 bind 4096)
[    0.400652] UDP hash table entries: 256 (order: 1, 8192 bytes)
[    0.400701] UDP-Lite hash table entries: 256 (order: 1, 8192 bytes)
[    0.400961] NET: Registered protocol family 1
[    0.402023] RPC: Registered named UNIX socket transport module.
[    0.402065] RPC: Registered udp transport module.
[    0.402090] RPC: Registered tcp transport module.
[    0.402114] RPC: Registered tcp NFSv4.1 backchannel transport module.
[    0.402789] hw perfevents: enabled with armv7_cortex_a9 PMU driver, 7 counters available
[    0.404341] futex hash table entries: 512 (order: 3, 32768 bytes)
[    0.404505] audit: initializing netlink subsys (disabled)
[    0.404585] audit: type=2000 audit(0.379:1): initialized
[    0.405111] Initialise system trusted keyring
[    0.405881] VFS: Disk quotas dquot_6.6.0
[    0.405984] VFS: Dquot-cache hash table entries: 1024 (order 0, 4096 bytes)
[    0.406373] squashfs: version 4.0 (2009/01/31) Phillip Lougher
[    0.407215] NFS: Registering the id_resolver key type
[    0.407288] Key type id_resolver registered
[    0.407315] Key type id_legacy registered
[    0.407361] nfs4filelayout_init: NFSv4 File Layout Driver Registering...
[    0.407468] jffs2: version 2.2. (NAND) (SUMMARY)   2001-2006 Red Hat, Inc.
[    0.407949] Allocating IMA MOK and blacklist keyrings.
[    0.409634] Key type asymmetric registered
[    0.409681] Asymmetric key parser 'x509' registered
[    0.409832] Block layer SCSI generic (bsg) driver version 0.4 loaded (major 248)
[    0.410040] io scheduler noop registered
[    0.410079] io scheduler deadline registered (default)
[    0.410137] io scheduler cfq registered
[    0.440104] Console: switching to colour frame buffer device 128x48
[    0.468985] xuartps e0001000.serial: clock name 'aper_clk' is deprecated.
[    0.469289] xuartps e0001000.serial: clock name 'ref_clk' is deprecated.
[    0.469614] e0001000.serial: ttyPS0 at MMIO 0xe0001000 (irq = 158, base_baud = 3125000) is a xuartps
[    1.278046] console [ttyPS0] enabled
[    1.282451] xdevcfg f8007000.ps7-dev-cfg: ioremap 0xf8007000 to e0872000
[    1.305388] brd: module loaded
[    1.315816] loop: module loaded
[    1.337113] libphy: Fixed MDIO Bus: probed
[    1.343139] libphy: XEMACPS mii bus: probed
[    1.348856] xemacps e000b000.ps7-ethernet: pdev->id -1, baseaddr 0xe000b000, irq 31
[    1.357688] ehci_hcd: USB 2.0 'Enhanced' Host Controller (EHCI) Driver
[    1.364433] ehci-pci: EHCI PCI platform driver
[    1.369044] ehci-platform: EHCI generic platform driver
[    1.381383] ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver
[    1.394522] ohci-pci: OHCI PCI platform driver
[    1.405966] ohci-platform: OHCI generic platform driver
[    1.418231] uhci_hcd: USB Universal Host Controller Interface driver
[    1.431756] usbcore: registered new interface driver usb-storage
[    1.445354] mousedev: PS/2 mouse device common for all mice
[    1.458696] i2c /dev entries driver
[    1.470432] device-mapper: uevent: version 1.0.3
[    1.482312] device-mapper: ioctl: 4.34.0-ioctl (2015-10-28) initialised: dm-devel@redhat.com
```

```
[    1.497966] sdhci: Secure Digital Host Controller Interface driver
[    1.511276] sdhci: Copyright(c) Pierre Ossman
[    1.522726] sdhci-pltfm: SDHCI platform and OF driver helper
[    1.537085] sdhci-arasan e0100000.ps7-sdio: No vmmc regulator found
[    1.550571] sdhci-arasan e0100000.ps7-sdio: No vqmmc regulator found
[    1.564036] mmc0: Invalid maximum block size, assuming 512 bytes
[    1.614085] mmc0: SDHCI controller on e0100000.ps7-sdio [e0100000.ps7-sdio] using ADMA
[    1.629895] ledtrig-cpu: registered to indicate activity on CPUs
[    1.644279] Key type dns_resolver registered
[    1.656171] Registering SWP/SWPB emulation handler
[    1.666380] mmc0: new high speed SDHC card at address aaaa
[    1.677100] mmcblk0: mmc0:aaaa SL08G 7.40 GiB
[    1.678486]  mmcblk0: p1 p2
[    1.703249] registered taskstats version 1
[    1.714453] Loading compiled-in X.509 certificates
[    1.727453] Key type encrypted registered
[    1.738464] AppArmor: AppArmor sha1 policy hashing enabled
[    1.750995] ima: No TPM chip found, activating TPM-bypass!
[    1.763635] evm: HMAC attrs: 0x1
[    1.774166] hctosys: unable to open rtc device (rtc0)
[    1.791487] md: Waiting for all devices to be available before autodetect
[    1.805427] md: If you don't use raid, use raid=noautodetect
[    1.819245] md: Autodetecting RAID arrays.
[    1.830359] md: Scanned 0 and added 0 devices.
[    1.841633] md: autorun ...
[    1.851105] md: ... autorun DONE.
[    1.861835] EXT4-fs (mmcblk0p2): couldn't mount as ext3 due to feature incompatibilities
[    1.877515] EXT4-fs (mmcblk0p2): couldn't mount as ext2 due to feature incompatibilities
[    1.906578] EXT4-fs (mmcblk0p2): mounted filesystem with ordered data mode. Opts: (null)
[    1.921636] VFS: Mounted root (ext4 filesystem) on device 179:2.
[    1.942290] devtmpfs: mounted
[    1.952285] Freeing unused kernel memory: 312K (c0830000 - c087e000)
[    2.204187] systemd[1]: System time before build time, advancing clock.
[    2.323264] NET: Registered protocol family 10
[    2.372338] random: systemd: uninitialized urandom read (16 bytes read, 6 bits of entropy available)
[    2.390906] random: systemd: uninitialized urandom read (16 bytes read, 6 bits of entropy available)
[    2.414805] systemd[1]: systemd 229 running in system mode. (+PAM +AUDIT +SELINUX +IMA +APPARMOR +SMACK +SYSVINIT +UTMP +LIBCRYPTSETUP +GCRYPT +GNU
[    2.447961] systemd[1]: Detected architecture arm.
[    2.491022] systemd[1]: Set hostname to <localhost.localdomain>.
```

And it goes on and with systemd initialization. After no more than 30 seconds, a shell prompt appears as follows. There's possibly a several seconds' pause before this final piece of output:

```
Ubuntu 16.04 LTS localhost.localdomain ttyPS0

localhost login: root (automatic login)

Last login: Thu Feb 11 16:28:21 UTC 2016 on ttyPS0
Welcome to the Xillinux-2.0 distribution for Xilinx Zynq.

You may communicate data with standard FPGA FIFOs in the logic fabric by
writing to or reading from the /dev/xillybus_* device files. Additional
pipe files of that sort can be set up with a custom Xillybus IP core.

For more information: http://www.xillybus.com.

To start a graphical X-Windows session, type "startx" at shell prompt.
```

```
root@localhost:~#
```

4.4 To do soon after the first boot

4.4.1 Resize the file system

The root file system's image is kept small so that writing it to the device is as fast as possible. On the other hand, there is no reason not to use the (Micro)SD card's full capacity.

IMPORTANT:

There's a significant risk of erasing the entire (Micro)SD card's content while attempting to resize the file system. It's therefore recommended to do this as early as possible, while the cost of such a mishap is merely to repeat the (Micro)SD card initialization (writing the image and populating the boot partition)

The starting point is typically as follows:

```
# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       3.4G  2.8G  388M  89% /
devtmpfs        241M     0  241M   0% /dev
tmpfs           249M   72K  249M   1% /dev/shm
tmpfs           249M  7.2M  242M   3% /run
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           249M     0  249M   0% /sys/fs/cgroup
tmpfs            50M  4.0K   50M   1% /run/user/0
```

So the root filesystem is 2.8 GB, with 388 MB free.

The first stage is to repartition the (Micro)SD card. At shell prompt, type

```
# fdisk /dev/mmcblk0
```

and then type as following (also see a session transcript below):

- d [ENTER] – Delete partition
- 2 [ENTER] – Choose partition number 2

- n [ENTER] – Create a new partition
- Press [ENTER] 4 times to accept the defaults: A primary partition, number 2, starting at the lowest possible sector and ending on the highest possible one.
- w [ENTER] – Save and quit.

If something goes wrong in the middle of this sequence, just press CTRL-C (or q [ENTER]) to quit fdisk without saving the changes. Nothing changes on the (Micro)SD card until the last step.

A typical session looks as follows. Note that the sector numbers may vary.

```
# fdisk /dev/mmcblk0

Welcome to fdisk (util-linux 2.27.1).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.


Command (m for help): d
Partition number (1,2, default 2): 2

Partition 2 has been deleted.

Command (m for help): n
Partition type
   p   primary (1 primary, 0 extended, 3 free)
   e   extended (container for logical partitions)
Select (default p):

Using default response p.
Partition number (2-4, default 2):
First sector (32768-15523839, default 32768):
Last sector, +sectors or +size{K,M,G,T,P} (32768-15523839, default 15523839):

Created a new partition 2 of type 'Linux' and of size 7.4 GiB.

Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Re-reading the partition table failed.: Device or resource busy
```

```
The kernel still uses the old table. The new table will be used at
the next reboot or after you run partprobe(8) or kpartx(8).
```

If the default first sector displayed on your system is different from the one above, pick your system's default, and not the one shown here.

The only place in this sequence, where it might make sense to divert from fdisk's defaults, is the last sector, in order to make a file system smaller than the maximum possible (but there's no need to do this).

As the warning at the bottom says, Linux' view of the partition table isn't updated. Following the suggestion, type:

```
# partprobe
```

This should produce no output to the console. If it does complain about not being able to inform the kernel of the change in partition 2, odds are it's because partprobe couldn't find it. In other words, the root partition isn't where the partition table says it should be. Most likely, something was done wrong with fdisk, and should be fixed, or the Linux will not be able to execute a boot again.

If partprobe was silent, the partition table is fine, but the file system has not been resized yet; it has only been given room to resize. So at shell prompt, type

```
# resize2fs /dev/mmcblk0p2
```

to which the following response is expected:

```
resize2fs 1.42.13 (17-May-2015)
Filesystem at /dev/mmcblk0p2 is mounted on /; on-line resizing required
old_desc_blocks = 1, new_desc_blocks = 1
The filesystem on /dev/mmcblk0p2 is now 1936384 (4k) blocks long.
```

The block count depends on the size of the partition, so it may vary.

As the utility says, the resizing takes place on a file system that is actively used. This is safe as long as power isn't lost in the middle.

The result is effective immediately: There is no need to reboot.

A typical session using an 8 GB (Micro)SD card:

```
# df -h
Filesystem      Size  Used Avail Use% Mounted on
```

```
/dev/root       7.1G  2.8G  4.0G  42% /
devtmpfs        241M     0  241M   0% /dev
tmpfs           249M   72K  249M   1% /dev/shm
tmpfs           249M  7.2M  242M   3% /run
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           249M     0  249M   0% /sys/fs/cgroup
tmpfs            50M  4.0K   50M   1% /run/user/0
```

Note that the sizes given by the “df -h” utility are with 1 GiB = 2^{30} bytes, which is 7.1% larger than a Gigabyte of 10^9 bytes. That’s why an 8 GB card appears as 7.1 GiB above.

4.4.2 Allow remote SSH access

The root password is none by default, allowing any user to login as root with no password at all. Consequently, ssh refuses to login root.

To rectify this, set the root password with the following command at shell prompt:

```
# passwd
```

4.4.3 Compilation of locale definitions (if needed)

In certain situations, application software requires knowledge on how to display characters depending on locale settings. The most common reason is when connecting with ssh to the board, because ssh copies the client’s locale settings to the server’s environment when setting up a shell session.

This may lead to warning messages like

```
bash: warning: setlocale: LC_CTYPE: cannot change locale (en_US.UTF-8)
```

When such error message appears, it’s quite obvious which locale is missing. To resolve this before an error occurs, check which locales are required:

```
# locale
LANG=en_US.UTF-8
LANGUAGE=
LC_CTYPE="en_US.UTF-8"
LC_NUMERIC="en_US.UTF-8"
```

```
LC_TIME="en_US.UTF-8"
LC_COLLATE="en_US.UTF-8"
LC_MONETARY="en_US.UTF-8"
LC_MESSAGES="en_US.UTF-8"
LC_PAPER="en_US.UTF-8"
LC_NAME="en_US.UTF-8"
LC_ADDRESS="en_US.UTF-8"
LC_TELEPHONE="en_US.UTF-8"
LC_MEASUREMENT="en_US.UTF-8"
LC_IDENTIFICATION="en_US.UTF-8"
LC_ALL=
```

Compare with the available locales:

```
# locale -a
C
C.UTF-8
POSIX
```

In this example, it's quite obvious which locale is missing, so let's add it:

```
# locale-gen en_US.UTF-8
Generating locales...
  en_US.UTF-8... done
```

Note that the necessary locale depends on the computer from which the ssh connection is made. Users from different places in the world need to install different locales on their boards, to achieve smooth ssh sessions. The shell on the UART port is based upon POSIX locale, which is included by default.

As en_US.UTF-8 is rather ubiquitous, it's already installed in the distribution (even though the example session above shows the opposite).

4.5 Using the desktop

The Xillinux desktop (on Z-Turn Lite, Zedboard and Zybo) is just like any Lubuntu desktop. Due to the (Micro)SD card's relatively low data bandwidth, applications may load somewhat slowly, but the desktop itself is fairly responsive.

Additional packages can be installed with "apt-get" like with any Ubuntu distribution.

Upgrading the entire Ubuntu operating system with apt is not possible, and will fail, leaving the system without possibility to perform boot again.

4.6 Shutting down / rebooting

To power down the system, pick the bottom-right icon on the desktop (if available), and click “Shutdown”, or pick any of the other options as suitable. “Lock Screen” does nothing.

Alternatively, type the following at shell prompt:

```
# halt
```

When a textual message saying “System Halted” appears on the UART console, (and on the screen, when present) it’s safe to power the board off.

For a reboot, which includes reloading the bitstream to the FPGA (PL) part, pick the reboot option on the desktop menu, or type:

```
# reboot
```

Note that this doesn’t necessarily reset the external hardware components, e.g. the sound chip.

4.7 What to do from here

The Zynq board has now become a computer running Linux for all purposes. The basic steps for interaction with the logic fabric through the Xillybus IP core can be found in Getting started with Xillybus on a Linux host. Note that the driver for Xillybus is already installed in the Xillinux distribution, so the part in the guide dealing with installation can be skipped.

Paragraph 5.1 refers to integrating application-specific logic with the Linux operating system.

Alternatively, users who want to stay away from Verilog / VHDL programming may choose the Block Design Flow, as outlined in The guide to Xillybus Block Design Flow for non-HDL users. Note however that this method utilizes a a weaker set of Xillybus’ features compared with Verilog / VHDL, so this method should be avoided unless the desired application fits a block design naturally (e.g. in conjunction with Vivado’s High Level Synthesis, HLS).

Note that Xillinux includes the gcc compiler and GNU make, so it’s possible to run a compilation of a regular computer program directly on the board’s processors. Additional packages may be added to the distribution with apt-get as well.

5

Making modifications

5.1 Integration with custom logic

This part relates to working with Verilog / VHDL. Users who have chosen the Block Design Flow should follow the instructions in The guide to Xillybus Block Design Flow for non-HDL users.

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

Additional HDL files with custom logic designs may be added to the project presented in paragraph 3.3, and then rebuilt the same way it was done in the beginning. To execute a boot of the system with the updated logic, copy the new xillydemo.bit into the (Micro)SD card's boot partition, overwriting the existing one. Note that it's possible to use the Zynq board itself for copying xillydemo.bit into the boot partition, as shown in paragraph 3.5.

There is no need to repeat the other steps of the initial distribution deployment, so the development cycle for logic is fairly quick and simple.

Programming the PL part through JTAG is not supported.

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 a FIFO's behavior with logic, at least not in the first stage.

An exception for this is when connecting Xillybus with a block RAM or with registers, in which case the method shown in the xillydemo module should be followed.

In the xillydemo module, FIFOs are used to perform a loopback of data arriving from the host and 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 data consumer.

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

The FIFOs used in the xillydemo module accept only one common clock for both sides, as both sides are driven by 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 data consumers 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 (as opposed to First Word Fall Through), for streams from the FPGA to host.

The following documents are related to integrating custom logic:

- The API for logic design: Xillybus FPGA designer's guide
- Basic concepts with the Linux host: Getting started with Xillybus on a Linux host
- Programming applications: Xillybus host application programming guide for Linux
- Requesting a custom Xillybus IP core: The guide to defining a custom Xillybus IP core

5.2 Using other boards

Before attempting to run Xillinux on a board other than Z-Turn Lite, Zedboard, MicroZed or Zybo, certain modifications may be necessary.

It's however not recommended to attempt adapting Xillinux to other hardware, as the procedure is difficult. Experience shows, that if the purpose of adapting Xillinux is other than to use the Xillybus IP core, it's easier to start from scratch.

This is a partial list of issues to pay attention to.

- A purchased board should have an XML file as a reference (for use as ps7_system_prj.xml). This file contains the processor's settings, including de-facto use of the MIO pins and the electrical parameters of the DDR pins. The recommended practice is adopting the reference file, at least as a starting point.

- If an XML file is adopted as reference, the FPGA CLK1 (FCLK_CLK1) must be set to 100 MHz, regardless of what the reference file says.
- If changes are made manually, attention should be paid to the processor core's MIO assignments: The ARM core has 54 I/O pins which are routed to physical pins on the chip with a fixed placement. The ARM core is configured in the project's block design to assign specific roles to these pins (e.g. USB interface, Ethernet etc.), which must match what these pins are wired to on the board.
- If changes are made in the processor's configuration (i.e. in the XML file), the boot.bin must be rebuilt, based upon an FSBL (First Stage Boot Loader) that is derived from the new XML file, and a U-boot binary. The changes made in Vivado's block design tool take effect through the initialization routine that is part of the FSBL. This routine writes to registers in the ARM processor, with values that reflect the settings made in Vivado, and exported to the SDK. Note that the parameters of the processor in the Vivado project may not be accurate, so the FSBL should be generated based upon the XPS project available in the bundle. To set up the sources for U-boot, please refer to the README file in /usr/src/xillinux/uboot-patches/.
- Alternatively, in some cases, rebuilding boot.bin can be avoided by pinpointing the changes in the registers' settings, by virtue of the "poke" feature. See paragraph 5.6.
- It may also be necessary to make changes in devicetree.dtb, in order to reflect the new setting. The sources of the existing DTB (in DTS format) can be found in the sources of the Linux kernel (see paragraph 6.2).
- The VGA/DVI outputs (if applicable) need to be matched to the intended board. This is done by editing the xillybus.v file in the src/ subdirectory. Note that the signals arriving from the "system" module are 8 bits wide, and the truncation to 4 bits takes place in xillybus.v. Hence it's fairly easy to connect these signal to any encoder chip for VGA/DVI.

5.3 Changing the frequencies of clocks in the system

The ARM processor's core supplies four clocks for use by the logic fabric, commonly referred to as FCLK_CLKn. It's important to note, that their frequencies are set by the FSBL (First Stage Boot Loader), before U-boot is loaded.

Accordingly, even though the clocks' frequencies are set in Vivado, these frequencies are effective only for propagating timing constraints and initialization by bare-metal applications that are compiled on the SDK.

If the hardware application requires different frequencies, the following series of actions is suggested:

- Update the frequencies of the clock in Vivado.
- Rebuild the netlist (this is necessary for updating the timing constraints in the .ncf files)
- Export the project to SDK, and create an FSBL application project based upon this.
- Learn the registers' settings required for the desired setting from Vivado's reports.
- Adjust as necessary, by virtue of the "poke" feature, as described in paragraph 5.6.

Please refer to Xilinx' guides for the details of how to perform each step (except for the last).

5.4 Taking over GPIO I/O pins for PL logic

5.4.1 Z-Turn Lite

While the Z-Turn Lite board itself supplies no convenient way to access the I/O pins for lab purposes, attaching it to the Z-Turn Lite IO Cape board exposes 68 I/O pins and a push button through standard connectors, in addition to the HDMI interface.

All of these 68 pins are wired to two 40-pin connectors, J3 and J8, to which standard flat ribbon cables can be attached. The IO Cape board has a few additional connectors, which share pins with J3 and J8. Since all of the additional connectors' pins are available on J3 and J8, Xillydemo's top-level module's ports for these pins are vectors named J3 and J8, with pin placement constraints routing them to the corresponding connectors.

The vectors in the ports in Verilog / VHDL of both J3 and J8 correspond to the pin numbers of the connectors minus 3: The signal J3[0] in Verilog / VHDL goes to physical pin J3/3. J3[1] goes to J3/4 etc. up to J3[33] going to J3/36. Same goes with J8.

For the sake of simplicity, all pins belonging to J8 are connected in xillydemo.v and xillydemo.vhd to the processor's GPIO pins, and can be controlled directly from software running on the processor. All pins of J3 are driven low by the xillydemo.v and xillydemo.vhd, and can easily be utilized by application logic by modifying the relevant xillydemo module file.

This division of the pins to GPIO and pins driven by application logic is also easy to change, by altering their wiring in the xillydemo module. If the number of pins that are used as GPIO is changed, the gpio_width instantiation parameter (generic) of the xillybus module should be altered accordingly. It currently stands at 35, which accounts for 34 I/O pins going to the J8 connector, plus one GPIO for the Cape Board's pushbutton.

And as mentioned before, there are additional connectors on the board which share their pins with J3 and J8, which can still be used. This requires however looking up which pins goes where in the Cape Board's schematics.

One side effect of this pin sharing is that some pins, which are used as I^2C by their alternative connectors, have pull-ups on the Cape board: J3[19], J3[18], J8[28] and J8[31] (using Verilog / VHDL vector signal notation). As these are pull-ups with resistors on the board, they are in effect even when using these pins on the J3 and J8 connectors.

The HDMI connector is independent, and shares no pin with neither J3, J8 nor any of the other connectors.

5.4.2 Zedboard and Zybo

On the Zedboard and Zybo boards, many physical I/O pins are connected to the ARM processor's GPIO ports (PS), which allows controlling and monitoring these pins directly from Linux. It's however often desired to connect these physical pins to FPGA logic instead (i.e. to PL).

The technique for using Zynq's PL pins for I/O is exactly the same as with any Xilinx FPGA: The signals are exposed in the toplevel module (xillydemo.v or xillydemo.vhd) as inputs, outputs or inouts. The assignment of physical pins to these signals takes place in xillydemo.xdc.

Since the pins are used as GPIO signals, they are taken away from the processor and given to the PL part. For example, the following line in the XDC file,

```
set_property -dict "PACKAGE_PIN U5 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[55]"]
```

can be replaced with

```
set_property -dict "PACKAGE_PIN U5 IOSTANDARD LVCMOS33" [get_ports "my_output"]
```

if we want my_output to appear on pin U5.

But doing this replacement causes PS_GPIO[55] to lack a pin assignment. Even though there is a chance that Xilinx' tools will place this port automatically during implementation, it's recommended to assign any evicted PS_GPIO with an I/O pin. The alternative is to eliminate the signal, as explained below.

So there are two solutions for these evicted PS_GPIO signals:

- The easy way: Finding unused pins on the device, and assign these pins to the evicted PS_GPIO signals. Even though it's not a very clean solution (GPIO pins are connected to just anything on the board), it's practically harmless, because GPIOs are inputs by default. The electrical condition on these pins remains, unless the GPIO gets driven by software accidentally (which isn't likely). For example, on Zedboard, the FMC connector often supplies many unused pins.
- The harder way: Reducing the number of PS_GPIO pins. This may be necessary on Zybo, which doesn't have many vacant pins.

In what follows, the second solution is discussed. For example, let's assume that PS_GPIO[55:48] were removed from the XDC file, for the sake of replacing their pins with signals from the PL. Note that if the pins from lower PS_GPIO indexes were needed, the evicted PS_GPIO signals should take over the pins of those with the highest indexes, and the latter are then eliminated. There is no possibility to eliminate a certain range of PS_GPIO indexes, only reduce the maximal index.

The width of PS_GPIO should be reduced in xillydemo.v/vhd to reflect those that have pin assignments in the XDC file.

This is not enough however. Attempting to build the project at this state, critical warnings will be issued for these pins (possibly claiming that these are multiple driven, with high-Z and GND).

To resolve this, edit vivado-essentials/system.v in the following part:

```
generate
  for (i=0; i<56; i=i+1)
    begin: gpio
      assign gpio_tri_i[i] = processing_system7_0_GPIO[i];
      assign processing_system7_0_GPIO[i] = gpio_tri_t[i] ? 1'bz :
                                            gpio_tri_o[i];
    end
endgenerate
```

Reduce the index range (i.e. the `i<56` part) to the number of used GPIOs (48 in the example).

Depending on Vivado's revision, it might also be necessary to adjust the widths of the signals.

It case the GPIO width needs to be corrected in the block design: In Vivado's main window, click "Open Block Design" on the left column. Right-click the processor block (processing_system_7_0, with the ZYNQ marking) and select "Customize block". Select "MIO Configuration" on the left column, expand the "I/O Peripherals" hierarchy, and expand the GPIO hierarchy (at the bottom). The EMIO GPIO (Width) parameters currently stands at 56, the number of GPIO pins. Reduce it to the desired number (48 in this example).

5.5 Working with 7020 MicroZed

The boot partition kit available for MicroZed is intended for 7010 MicroZed boards by default. It's however possible to work with 7020 MicroZed using Vivado, after making a minor change in the xillydemo-vivado.tcl file used to create the Vivado project (that is, verilog/xillydemo-vivado.tcl or vhdl/xillydemo-vivado.tcl in the bundle, depending on the language chosen).

Just after unzipping the kit (and before using it in Vivado), the file should be edited, changing the line saying

```
set thepart "xc7z010clg400-1"
```

(around line 11) to

```
set thepart "xc7z020clg400-1"
```

The rest of the build process is exactly the same.

5.6 Pre-boot manipulation of hardware registers (“poke”)

It’s often desirable to make slight changes in the ARM processor’s hardware setup without rebuilding the boot.bin file (paragraph 5.3 discusses a typical rebuild sequence).

For example, slight changes in the processor’s MIO/EMIO configuration result in a few changes in the registers’ settings, which can be deduced quite easily by looking for differences in the reports that are generated when the system’s settings are exported to the software tools.

The processor’s hardware registers are documented in Xilinx’ Zynq-7000 AP SoC Technical Reference Manual, also known as the TRM or ug585.

In order to manipulate registers, an entry is added to the kernel’s device tree (typically by editing the respective DTS file given in the Linux kernel sources, see paragraph 6.2).

An entry like the following example is added anywhere in the device tree’s hierarchy (preferably after the “chosen” entry):

```
poke {
        compatible = "xillybus,poke-1.0";
        sequence = < 0 0xf8002000 0
                     0 0xf800200c 0
                     0 0xf8002018 0
                     1 0xf800200c 0x20
                     0 0xf8002018 0
                     0 0xf8002018 0
                     1 0xf800200c 0x21
                     0 0xf8002018 0
                     0 0xf8002018 0
                   >;
};
```

The “sequence” part should be altered to set up the desired sequence of register reads and writes. Each operation is defined by three values in the “sequence” array of elements. There is no restriction on the number of triplets (and hence operations). The formatting of the “sequence” entry above, with tabs and three values per row, has no syntactic significance – what matters is that each triplet represents an operation as follows:

- First element: Read or write. A value of 0 means read, otherwise write.

- Second element: The address. Must be 32-bit aligned (2 LSBs of the address must be zero).
- Third element: The value to write. Ignored on read operations.

The operations are carried out in the order listed in the device tree entry, with unpredictable delays between each operation.

In the example above, which has no practical significance, the registers of the processor's ttc2 (Triple Timer Counter 2) are manipulated: The first three operations read registers for the sake of demonstration. Then the counter is enabled briefly, its counter value is read twice to show that it's changing, and then the counter is disabled. After this, the counter value is read twice again, to show that is has stopped.

The result of these operations can be found in the kernel's message log, which is available at the serial console (UART), and/or with the dmesg command at shell prompt:

```
[    0.000000] poke read addr=f8002000: value=00000000
[    0.000000] poke read addr=f800200c: value=00000021
[    0.000000] poke read addr=f8002018: value=00000000
[    0.000000] poke write addr=f800200c: value=00000020
[    0.000000] poke read addr=f8002018: value=00000009
[    0.000000] poke read addr=f8002018: value=00004f68
[    0.000000] poke write addr=f800200c: value=00000021
[    0.000000] poke read addr=f8002018: value=000013ec
[    0.000000] poke read addr=f8002018: value=000013ec
```

The values read from 0xf8002018 vary, of course, as they're read from a running counter.

The register modification takes place early in the kernel's boot process, before any device driver is loaded. Note however that U-boot sets up a few of the ARM processor's hardware peripherals before Linux starts its boot process, so they are already active. Also note that modifying registers that relate to the ARM processor's basic functionality (e.g. clocks and interrupts) may disrupt the proper functionality of the processor itself. This can happen even though the interrupts are disabled by the kernel at the time the "poke" is executed.

Attempting to access addresses that are disallowed, either by the kernel's memory management and/or by the hardware itself, leads to a kernel Oops, kernel panic and possibly a complete freeze. The two latter result in a boot failure. A kernel panic on grounds of "imprecise external abort (0x406)" is probably due to an attempt to access an hardware-wise illegal address.

Moreover, since the early kernel console messages are stored in an internal memory buffer at the stage they're generated, and written to the console only at a later stage (when the serial port is set up), an early freeze is likely to result in no output to the console at all – nothing appearing after U-boot's "done, booting the kernel" message.

Hence when no kernel messages appear on the console, it doesn't necessarily mean that the kernel didn't kick off. It could be a result of a freeze before the kernel messages, that were stored in memory, were written to the console. The reason can be an illegal modification of a register.

The "poke" feature was added by patching the kernel of Xillinux-2.0 specifically, and is not part of mainline Linux kernels.

6

Linux notes

6.1 General

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

A wider view on Xillybus and Linux can be found in these documents:

- Getting started with Xillybus on a Linux host
- Xillybus host application programming guide for Linux

6.2 Compilation of the Linux kernel

Due to its size, the full Linux kernel is not included in the Xillinux distribution, but can be downloaded from Github with

```
$ git clone https://github.com/xillybus/xillinux-kernel.git
```

For an exact replication of the kernel used with Xillinux-2.0, check out the “xillinux-2.0” tag. Inform the kernel build environment on the desired cross compiler with something of the form of

```
$ export CROSS_COMPILE=/path/to/crosscompiler/arm-xilinx-linux-gnueabi-
```

Then obtain the .config file, that is used for compilation of the kernel which comes with Xillinux-2.0:

```
$ make ARCH=arm xillinux_defconfig
```

And then run the compilation of the kernel:

```
$ make ARCH=arm -j 8 uImage modules LOADADDR=0x8000
```

In order to build the Device Tree Blobs, the following command can be used, after setting the cross compiler and the .config file as said above:

```
$ make ARCH=arm dtbs
[ ... ]
  DTC     arch/arm/boot/dts/xillinux-microzed.dtb
  DTC     arch/arm/boot/dts/xillinux-zedboard.dtb
  DTC     arch/arm/boot/dts/xillinux-zybo.dtb
  DTC     arch/arm/boot/dts/xillinux-zturn-lite.dtb
```

The DTB files can be found in arch/arm/boot/dts/, as suggested by the command's response.

6.3 Compilation of kernel modules

The Xillinux distribution comes with the running kernel's compilation headers. This is not enough to perform a compilation of the kernel itself, but allows the compilation of kernel modules directly on the platform.

The standard way for compilation of an out-of-tree kernel module is using a Makefile which invokes the kernel's own build environment, after setting up an environment variable, which tells it to perform compilation of 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, example.c, is as follows:

```
ifneq ($(KERNELRELEASE),)
obj-m := example.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, “example”, is mentioned only in the “obj-m” line. This is the only thing that differs from one Makefile to another.

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

```
# make
make -C /lib/modules/4.4.30-xillinux-2.0/build SUBDIRS=/root/example modules
make[1]: Entering directory '/usr/src/linux-headers-4.4.30-xillinux-2.0'
  CC [M]  /root/example/example.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /root/example/example.mod.o
  LD [M]  /root/example/example.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.4.30-xillinux-2.0'
```

6.4 Sound support

6.4.1 General

The Zedboard and Zybo boards support sound recording and playback by virtue of Analog Devices’ ADAU1761 and SSM2603 chipsets, respectively. These are 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 / I^2C.

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.

6.4.2 Usage details

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

6.4.3 Related boot scripts

There are two tasks related to setting up sound on Zedboard and Zybo: Programming the audio chip and launching the Pulseaudio daemon.

For the first task, two systemd unit files are in place at /etc/systemd/system/:

- xillinux_sound.service: Launches /usr/local/bin/xillinux-sound when started.
- xillinux_sound.path: Instructs systemd to wait for /dev/xillybus_smb to appear, and when that happens, start the service just mentioned. Note that this is not based upon udev, but on inotify, which is monitoring /dev.

/usr/local/bin/xillinux-sound identifies which board it runs on, and runs the script /usr/local/bin/zybo_sound_setup.pl or /usr/local/bin/zedboard_sound_setup.pl, as applicable.

The Pulseaudio daemon is started when any user creates its first login session on the system, by virtue of the user service unit file xillinux_pulseaudio.service, which resides in /etc/systemd/user/.

Since Xillinux auto-logins the root user on the serial console as well as the screen console, Pulseaudio is started soon after the system's boot is completed.

Running Pulseaudio as root is not a recommended on a multi-user computer, but since Xillinux runs with root as the default user, this is the least problematic option.

The service should be disabled if direct access to /dev/xillybus_audio is needed, with e.g.

```
# systemctl --global disable xillinux_pulseaudio
Removed symlink /etc/systemd/user/default.target.wants/
xillinux_pulseaudio.service.
```

zybo_sound_setup.pl and zedboard_sound_setup.pl are Perl scripts, which set up the audio chip's registers for proper operation. They are fairly straightforward, even for programmers who are not familiar with Perl. The script uses the /dev/xillybus_smbus device file to initiate transactions on the chip's I^2C bus.

zedboard_sound_setup.pl 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);
```

In a similar manner, zybo_sound_setup.pl can be edited, replacing

```
write_i2c(0x04, 0x14);
```

with

```
write_i2c(0x04, 0x10);
```

to use Line In for recording.

6.4.4 Accessing /dev/xillybus_audio directly

/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 /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
```

it's likely that 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.

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

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.

6.5 The OLED utility (Zedboard only)

By default, Xillinux starts an activity meter utility during 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 previous 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 /usr/local/bin/start_zedboard_oled, which 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% of the SDIO interrupt rate, in interrupts per second. The current figure was found by trial and error.

To prevent this utility from being launched during boot:

```
# systemctl disable zedboard_oled.path
Removed symlink /etc/systemd/system/paths.target.wants/zedboard_oled.path.
```

6.6 Other notes

- Even though there have been changes in the Linux GPIO driver since kernel 3.12, the same GPIO driver as Xillinux-1.3 is used in Xillinux-2.0, retaining the behavior and numbering of Xillinux-1.3.

  In order to use the new GPIO driver, alter the device tree entry with compatible = “xlnx,ps7-gpio-1.00.a”, so it says “xlnx,zynq-gpio-1.0”.
- The Quad SPI flash is accessed by the Linux kernel with one bit wide bus, even though the driver supports a quad bit interface. The relevant device tree entry can be changed for enabling the quad bit interface. This is not the default setting to allow the same device tree to be used with kernel 3.12 (i.e. Xillinux-1.3).

7

Troubleshooting

7.1 Errors during implementation

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

If the problem isn't solved fairly quickly, please seek assistance at Xillybus' forum:

http://forum.xillybus.com

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, please detail these changes. Also please state which version of the ISE / Vivado tools was used.

If this error is issued by Vivado on an implementation for VHDL:

```
ERROR: [Place 30-58] IO placement is infeasible. Number of unplaced terminals (3) is greater than number of available sites (2).
The following Groups of I/O terminals have not sufficient capacity:
Bank: 35:
 The following table lists all user constrained IO terminals
 Please analyze any user constraints (PACKAGE_PIN, LOC, IOSTANDARD ) which may cause a feasible placement to be impossible.
 The following table uses the following notations:
 c1 - is IOStandard compatible with bank? 1 - compatible, 0 is not
 c2 - is IO VREF compatible with INTERNAL_VREF bank? 1 - compatible, 0 is not
 c3 - is IO with DriveStrength compatible with bank? 1 - compatible, 0 is not
+--------+------------+----+----+----+-----------------------------------------------------+
| BankId | IOStandard | c1 | c2 | c3 | Terminal Name                                       |
+--------+------------+----+----+----+-----------------------------------------------------+
|   35   |LVCMOS18    |  1 |  1 |  1 |   PS_CLK                                            |
|   35   |LVCMOS18    |  1 |  1 |  1 |   PS_PORB                                           |
|   35   |LVCMOS18    |  1 |  1 |  1 |   PS_SRSTB                                          |
+--------+------------+----+----+----+-----------------------------------------------------+
```

Please edit xillydemo.vhd as mentioned in paragraph 3.3.

Another possible error resulting from the same problem:

```
ERROR: [Drc 23-20] Rule violation (NSTD-1) Unspecified I/O Standard
```

```
        ...
        Problem ports: PS_CLK, PS_PORB, PS_SRSTB.
ERROR: [Drc 23-20] Rule violation (RTSTAT-1) Unrouted net ...
ERROR: [Drc 23-20] Rule violation (UCIO-1) Unconstrained Logical Port ...
```

7.2 Problems with USB keyboard and mouse

Almost all USB keyboards and mice meet a standard specification for compatible behavior, so it's unlikely to face problems with devices that aren't recognized. The first things to check if something goes wrong are:

- Zedboard only: Are you using the correct USB plug? It should be the one marked "USB OTG", farther away from the power switch.
- Zedboard only: Is there any 5V supply to the devices? Is the JP2 jumper installed? With the Zedboard powered on, connect an optical USB mouse, and verify that the LED goes on.
- If a USB hub is used, attempt to connect only a keyboard or mouse directly to the board.

Helpful information may be present in the general system log file, /var/log/syslog. Viewing its content with "less /var/log/syslog" can be helpful at times. Even better, typing "tail -f /var/log/syslog" will dump new messages to the console as they arrive. This is useful in particular, as events on the USB bus are always noted in this log, including a detailed description on what was detected and how the event was handled.

Note that a shell prompt is also accessible through the USB UART, so the log can be viewed with a serial terminal if connecting a keyboard fails.

7.3 Issues with file system mount

Experience shows that if a proper (Micro)SD card is used, and the system is shut down properly before powering off the board, there are no issues at all with the data in the (Micro)SD card.

Powering off the board without unmounting of the root file system is unlikely to cause permanent inconsistencies in the file system itself, since the ext4 file system repairs itself with the journal on the next mount. There is however an accumulating damage in the operating system's functionality, since files that were opened for write when the

power went off may be left with false content or deleted altogether. This holds true for any computer being powered off suddenly.

If the root file system fails to perform a mount (resulting in a kernel panic during boot) or performs the mount only as read-only, the most likely cause is a low quality (Micro)SD card. It's quite typical for such storage to function properly for a while, after which random error messages begin to appear. If /var/log/syslog contains messages such as this one, the (Micro)SD card is most likely the reason:

```
EXT4-fs (mmcblk0p2): warning: mounting fs with errors, running ec2fsck
   is recommended
```

To avoid these problems, please insist on a Sandisk device.

7.4 "startx" fails (Graphical desktop won't start)

Even though not directly related, this problem is reported quite frequently when the (Micro)SD card is not made by Sandisk. The graphical software reads a large amount of data from the card when starting up, and is therefore likely to be the notable victim of an (Micro)SD card that generates read errors.

The obvious solution is using a Sandisk (Micro)SD card.

7.5 Black screen after screensaver with X desktop

If the /root directory has been cleared, or if a new user is using the desktop, or if the power saving settings have been changed, the desktop may not recover from screensaver mode, leaving a black screen when attempting to resume.

Fix: On the XFCE desktop, go to the Power Manager, and make the following settings:

- Set Display $>$ "Put to sleep after" to "Never"
- Set Display $>$ "Put to sleep after" to "Never"
- Set Display $>$ "Switch off after" to "Never".
- Set Security $>$ "Automatically lock the session" to "Never".

This problem should not occur on a fresh Xillinux distribution, since these settings are already set up for the root user.