Getting started with Xillinux for Zynq-7000 EPP

Xillybus Ltd.

www.xillybus.com

Version 2.0

1

Introduction

1.1 The Xillinux distribution

Xillinux is a complete, graphical, Ubuntu 12.04 LTS-based Linux distribution for the Zedboard, intended as a platform for rapid development of mixed software / logic projects. 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.

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.

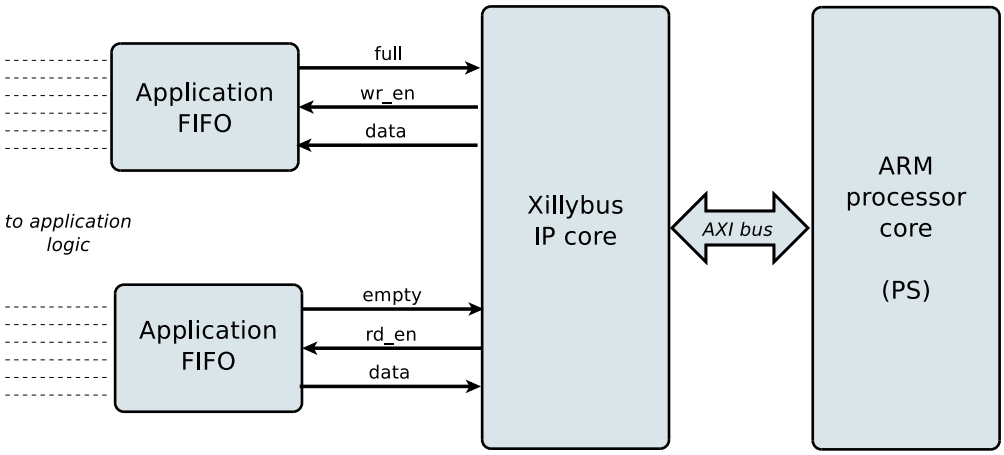
Xillinux is also a kickstart development platform for integration between the device's FPGA logic fabric and plain user space applications running on the ARM processors. With its included Xillybus IP core and driver, no more than basic programming skills and logic design capabilities are needed to complete the design of an application where FPGA logic and Linux-based software work together.

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

1.2 The Xillybus IP core

Xillybus is a straightforward, portable, intuitive, efficient DMA-based end-to-end turnkey solution for data transport between an FPGA and a host running Linux or Microsoft Windows. 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 Xillybus reads the data from the FIFO and sends it to the host, making it readable by the userspace software. The data transport mechanism is transparent to the application logic in the FPGA, which merely interacts with the FIFO.

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

The IP core is built instantly per customer's spec, using an online web interface. It's recommended to build and download your custom IP core at http://xillybus.com/custom-ip-factory after walking through the demo bundle flow described in this guide.

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

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

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

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

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

2

Prerequisites

2.1 Hardware

The Xillybus for Zynq Linux distribution (Xillinux) currently supports only the Zedboard.

Owners of other boards 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.

The following pieces of equipment are also required:

- A monitor capable of displaying VESA-compliant 1024x768 @ 60Hz with an analog VGA input (i.e. virtually any PC monitor).
- An analog VGA cable for the monitor
- A USB keyboard
- A USB mouse
- A USB hub recognized by Linux 3.3.0, if the keyboard and mouse are not combined in a single USB plug
- A USB cable to the Zedboard card (one of the two that came with the board).
- A reliable SD card with 2GB or more, preferably Sandisk. The card that came with the board is not recommended, as problems have been reported using it with Xillinux.
- Recommended: A USB adapter between an 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 Zedboard itself can also be used as a flash writer but this is somewhat trickier.

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.

2.2 Downloading the distribution

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

http://xillybus.com/zynq-download

The distribution consists of two parts: A raw image of the SD card consisting of the file system to be seen by Linux at bootup, and a set of files for implementation with the Xilinx tools to produce a first-stage boot image. More about this is 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 interface. 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 core in end-user designs, running real-life data and field testing. There is no limitation on how the core is used, as long as the sole purpose of this use is to evaluate its capabilities and fitness for a certain application.

2.3 Development software

The recommended tool for compiling the logic fabric parts of the Xillinux distribution is Xilinx ISE Design Suite release 14.2. Later releases should work properly as well.

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

Any of the design suite'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 target device is covered (e.g. XC7Z020, used on the Zedboard, is covered by this edition).

- the Logic Edition, which requires a purchased license (but a 30-day trial is available).
- any target-locked edition that may have been licensed specifically along with a purchased board.
- the DSP and Embedded Editions are fine as well, but have a higher license fee.

All of these design suite 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 targeting the Zedboard, no previous experience with FPGA design is necessary to have the distribution running on the platform. Targeting another board requires some knowledge with using Xilinx' tools, and possibly some basic capabilities 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. This makes the preparation for the first test run somewhat timely (typically 30 minutes, most of which consist of waiting for Xilinx' tools) but significantly shortens the cycle for integrating custom logic.

To boot the Xillinux distribution from an SD card, it must have two components:

- An initial boot image environment, consisting of boot loaders, a configuration bitstream for the logic fabric (known as PL), and the binaries for booting the Linux kernel.
- A root file system mounted by Linux.

The various operations for preparing the SD are detailed step by step in this section. Most of the time is spent on preparing the logic fabric's bitstream.

This flow assumes a Zedboard is targeted. It consists of the following steps, which must be done in the order outlined below:

- Unzipping the boot image kit
- Generating the processor netlist
- Generating Xilinx IP cores
- Implementing the main logic fabric project
- Producing a boot image file

- Writing the raw Xillinux image to the SD card
- Copying the boot image file into the SD card

How to target other boards is discussed in paragraph 5.2.

3.2 Unzipping the boot image kit

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

IMPORTANT:

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

The bundle consists of the following directories:

- 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, with the user-editable source file in VHDL (in the 'src' subdirectory)
- cores – Precompiled binaries of the Xillybus IP cores
- system – Directory for generating processor-related logic
- runonce – Directory for generating general-purpose logic (CoreGen FIFO IP cores).
- boot – Final stage assembly of the boot image file.

Note that both 'src' directories also contain the UCF file for the Zedboard. This file must be edited if another board is targeted.

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

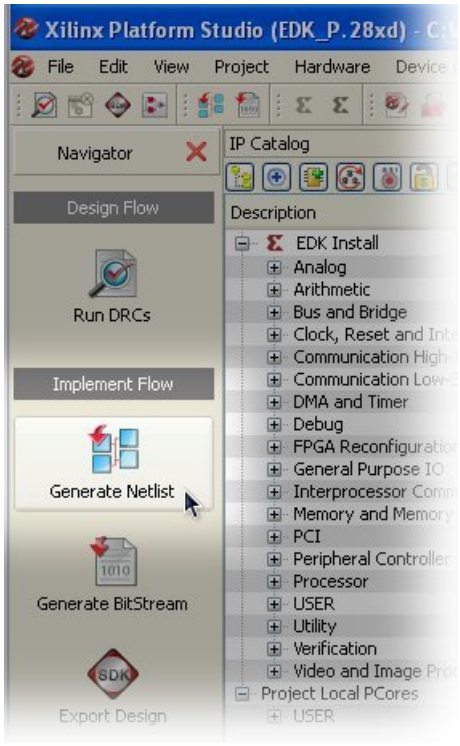
The interface with the Xillybus IP core takes place in the xillydemo.v or xillydemo.vhd files in the respective 'src' subdirectories. This is the file to edit in order to try Xillybus with your own data sources and sinks.

3.3 Generating the processor netlist

Within the boot image kit, double-click the system.xmp file in the "system" directory. This opens the Xilinx Platform Studio (XPS).

> **ISE Version 14.3 note:**
>
> In XPS' main window (the large one to the top right), choose the "Zynq" tab for the graphical representation of the processing system. Click on the I/O Peripherals block to the left to open the Zynq PS MIO Configurations window. Expand the GPIO Peripheral entry (click on the lowest "+" symbol to the left). Click the check box next to "EMIO GPIO (Width)" and set the width to 56.

Click "Generate Netlist" to the left (as shown in the image below).



The process takes up to 10 minutes, depending on the computer running it. Several warnings are generated, but no errors should be tolerated (which is unlikely to happen).

The console output says “XST completed” and “Done!” upon a successful completion of the process. At this point, close the XPS completely. There is no need to ever repeat this process in the normal use of Xillinux.

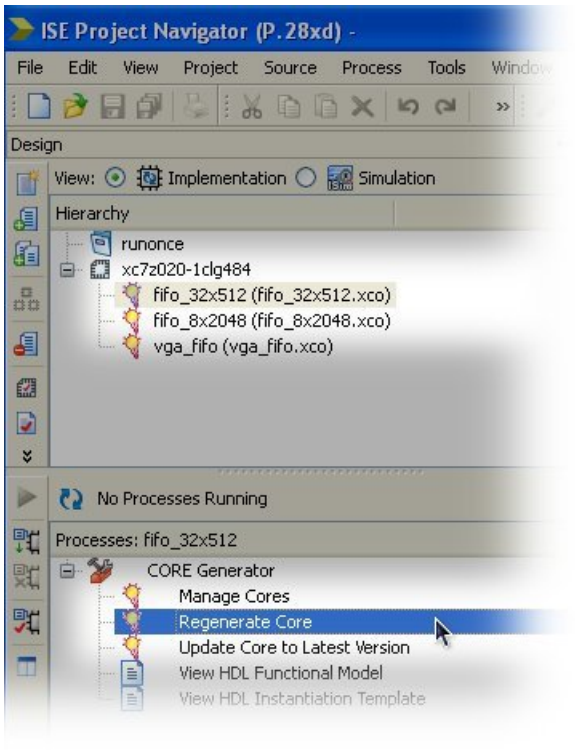
It’s possible that a Xilinx License Error dialog box followed by a Xilinx License Configuration Manager window will appear upon invocation of XPS. These should be ignored as a first measure (with “OK” and “Close”, respectively), since a license for XPS targeting certain Zynq devices is actually included in the WebPack license group. Some versions of XPS issue an error despite this.

If licensing errors continue to appear, preventing the generation of a netlist, make sure a WebPack (or broader) license is installed on the machine.

3.4 Generating Xilinx IP cores

Within the boot image kit, double-click the runonce.xise file in the “runonce” directory. This opens the Xilinx’ ISE Project Navigator.

On the opened windows top left, click on fifo_32x512, then expand the “CORE Generator” line in the process window below (clicking on the “+”), and finally, double click “Regenerate Core”, as shown in the image below.



The process produces a handful of warnings, but no errors, and ends with a message saying: Process “Regenerate Core” completed successfully.

Repeat this for the two other IP cores: fifo_8x2048 and vga_fifo.

At this point, close the ISE Project Navigator completely. There is no need to ever repeat this process.

3.5 Generating the bitstream file

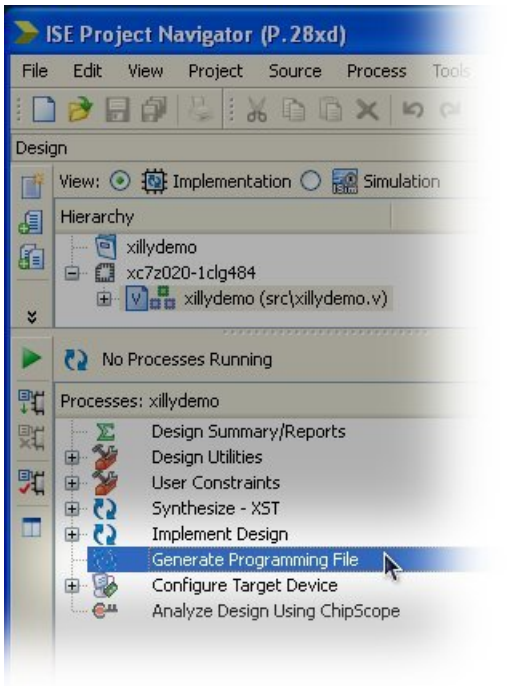
IMPORTANT:

*Please verify that the processes described in paragraphs 3.3 and 3.4 were indeed carried out, before moving on to generating the bitstream. In particular, please recall that the “Regenerate Core” process was run **three times** in paragraph 3.4, once for each core. There is no risk to get a faulty bitstream though, since the implementation will fail with an error if the preparations weren’t completed.*

Depending on your preference, double-click the ’xillydemo.xise’ file in either the ’ver-

ilog' or 'vhdl' subdirectory. Both directories are in the boot image kit. In the absence of preference, pick the 'verilog' subdirectory.

The Project Navigator will launch and open the project with the correct settings. Just click "Generate Programming File".



The procedure will produce several warnings (FPGA implementations always do) but should not present any errors. The process should end with the output "Process Generate Programming File completed successfully". The result can be found as xillydemo.bit in the 'verilog' or 'vhdl' directory (whatever was chosen) along with several other files.

If an error occurs, please check that the steps of paragraphs 3.3 and 3.4 were carried out fully, in particular that the latter was carried out for all three IP cores. Most notably, if the bottom line at the console is "Process Translate failed", with a previous error message complaining about not being able to resolve some logical block, the proper completion of the the two previous steps should be verified.

Close the ISE Project Navigator completely when this task is completed successfully.

3.6 Creating the boot.bin image

Copy the xillydemo.bit file (from the 'verilog' or 'vhdl' subdirectory, whichever was chosen) into the 'boot' directory .

Next, make sure that the ISE tools' directory is in the execution path. Alternatively, makeboot.bat can be edited to call the "bootgen" utility with a full path.

On Windows machines, the said path is usually something like

```
{path-to-xilinx-installation}\14.2\ISE_DS\ISE\bin\nt\
```

With the path issue sorted out, double-click makeboot.bat.

The boot image is created as boot.bin in the same directory from the xillydemo.bit file and boot loader ELF files that are already in the 'boot' directory.

If a boot.bin file isn't generated, check the error message. To get a chance to see it, open a command prompt (or use a shell on Linux machines). Change to the 'boot' subdirectory in the boot image kit. Type "makeboot" at command prompt.

3.7 Loading the SD with the image

3.7.1 General

The purpose of this task is to write the downloaded SD card image file to the device. The image was downloaded as a file named xillinux.img.gz (or similar), and is a gzip-compressed image of the SD card (the boot.bin file, which was generated in paragraph 3.6, is missing in the image though).

This image should be uncompressed and then written to the 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 the boot image, and the Linux root file system of ext4 type. The second partition is ignored by almost all Windows computers, so the SD card will 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 SD card (or a dedicated slot on a computer), the board itself can be used to write the image, as described in paragraph 3.7.4.

IMPORTANT:

Writing an image to the 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.

3.7.2 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 SD card.

Some computers (laptops in particular) have an 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 downloading 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 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:\         |   2014 MB
```

(That was the letter “l” as in “list” after ”usbitcmd”, not the figure “one”)

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 `\path\to` replaced with the path to where the SD card’s image is stored on your computer.

3.7.3 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 the data of an entire hard disk. The distance between the desired operation and a complete disaster is one character typed wrong. 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 SD card. This is best done by plugging in the USB connector, and looking for something like this is 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 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.

3.7.4 Using the Zedboard for loading the image

Paragraph 3.7.3 above described how to load the image with a Linux machine and a USB adapter. Instead, the Zedboard 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 target device (instead of /dev/sdc).

This works fine with the sample Linux system, since it runs completely from RAM, and doesn’t use the SD card after booting has finished (this is not the case with Xillinux, of course). So it’s possible to pull out the SD card used for booting, and insert another for writing the image to. In fact, it’s also possible to write on the card from which Linux was booted, but if something goes wrong, there will be nothing to boot from at

all. Besides, the SD card that came with the board isn't recommended for use with Xillinux, since it has been reported unreliable.

How to give the Zedboard access to the SD image and boot.bin 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 USB 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/.

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

3.8 Copying the boot.bin file into the SD card

Unplug the USB adapter and then connect it back to the computer. If the Zedboard was used for writing the raw image, pull the SD card from its slot, and reinsert it.

This is necessary to make sure that the computer is up to date with the SD card's partition table. If necessary, mount the first partition (e.g /dev/sdb1). Most computers will do this automatically.

If the Zedboard itself is used for this purpose, type

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

and access the file system at /mnt/sd/. Then copy the boot.bin (which was generated in paragraph 3.6) to the FAT file system on the SD card. On Windows systems, this is the only "disk" the system will display. On Linux systems, it's the first (and smaller) partition. Either way, the correct destination is easily recognized by its existing content, which is only two files: 'devicetree.dtb' and 'zImage'.

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

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

4

Booting up

4.1 Jumper settings

For the board to boot from the SD card, modifications in the jumper settings need to be made. 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, which is the case, see page 34 of the Zedboard's Hardware Guide).

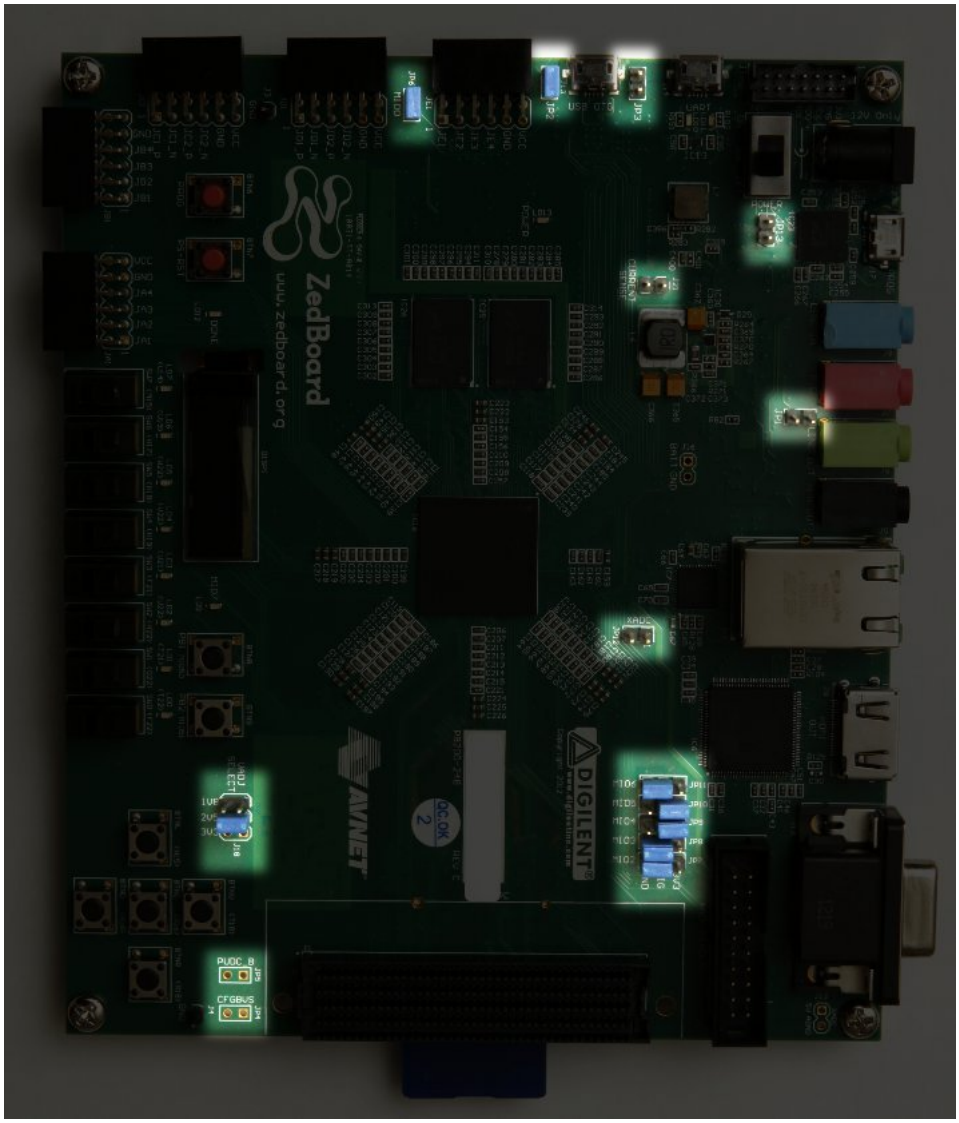
IMPORTANT:

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.

4.2 Attaching peripherals

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

- A computer monitor to the analog VGA connector. Since Xillinux produces a VESA-compliant 1024x768 @ 60Hz through the analog VGA plug, it's almost certain that any computer monitor will suffice.



Jumper settings highlighted on the Zedboard

- A mouse and keyboard to the USB OTG connector, through the USB female cable that came with the board (which is also the shorter one). The system will 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 JP2 on the board 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 is optionally connected to a PC, but is redundant in most cases. Some of the boot messages are sent there, and a shell prompt is issued on this interface when the boot completes.

  This is useful when either a PC monitor or a keyboard is missing or don't work properly.

4.3 Powering up the board

This paragraph describes what to expect on a proper cycle from powering up the system. The USB UART is disregarded, even though it should produce output, and present a shell prompt when the Linux system is up. However the output on the UART console is rarely informative for the purpose of solving possible problems, since most of the boot output goes to the VGA monitor.

IMPORTANT:

Unlike the Linux sample on the SD card that came with the Zedboard, Xillinux' root file system permanently resides on the 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 graceful recovery is usually observed on sudden power drops.

Plug the SD card into the Zedboard, and power it on. The following sequence is expected:

- Only the green "POWER" LED goes on. Nothing else happens.
- About 8 seconds later, the blue "DONE" LED goes on, and a red LED starts blinking. Other LEDs go on as well. The VGA monitor displays a screensaver

pattern with Xillybus' logo moving on white background. All these indicate that the logic fabric (PL, "FPGA") has been loaded properly with the bitstream (xillydemo.bit).

- After some additional 14 seconds (22 seconds from power-up), Linux bootup text appears rapidly on the VGA monitor. This looks exactly like a PC booting, with white text on black background.
- A login prompt should appear no later than 10 seconds after the bootup text was first seen on the VGA monitor. The system auto-logins as root, presenting a greeting message and a shell prompt. A similar shell prompt is also presented at the USB UART link, mostly for troubleshooting.

Type "startx" at command prompt to launch a Gnome 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.

Notes:

- The root user's password is set to nothing, so logging in as root, if ever needed, doesn't require a password.
- 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 driver 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.

4.4 To do soon after the first boot

4.4.1 Resize the file system

The root file system 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 SD card's full capacity.

IMPORTANT:

There's a significant risk of erasing the entire 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 SD card initialization (writing the image and boot.bin)

The starting point is typically as follows:

```
# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       1.7G  1.5G  125M  93% /
devtmpfs        243M  4.0K  243M   1% /dev
none             49M  664K   48M   2% /run
none            5.0M     0  5.0M   0% /run/lock
none            243M   76K  243M   1% /run/shm
```

So the root filesystem is 1.7 GB, with 125 MB free.

The first stage is to repartition the 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 SD card until the last step.

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

```
root@localhost:~# fdisk /dev/mmcblk0

Command (m for help): d
Partition number (1-4): 2

Command (m for help): n
Partition type:
   p   primary (1 primary, 0 extended, 3 free)
   e   extended
Select (default p):
Using default response p
Partition number (1-4, default 2):
Using default value 2
First sector (32130-15523839, default 32130):
Using default value 32130
Last sector, +sectors or +size{K,M,G} (32130-15523839, default 15523839):
Using default value 15523839

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.

WARNING: Re-reading the partition table failed with error 16:
         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)
Syncing disks.
```

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.

As the warning at the bottom says, Linux' view of the partition table can't be updated, because the root partition is in use. So a reboot is due:

```
# shutdown -h now
```

After there's a message saying "System halted." on the console, power the board off and on again. The system should boot up just like before, but the boot may fail at any

stage if something has been done incorrectly during the repartitioning.

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 (29-Nov-2011)
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 1936463 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 SD card:

```
# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       7.4G  1.6G  5.5G  23% /
devtmpfs        243M  4.0K  243M   1% /dev
none             49M  664K   48M   2% /run
none            5.0M     0  5.0M   0% /run/lock
none            243M   68K  243M   1% /run/shm
```

Note that the sizes given by the "df -h" utility are with 1 GiB = 2^{30} bytes, which is 7.3% larger than a "commercial" Gigabyte of 10^9 bytes. That's why an 8 GB card appears as 7.4 GiB above.

4.4.2 Allow remote SSH access

To install an ssh server on the board, connect the board to the internet and type

```
# apt-get install ssh-server
```

at shell prompt. Please note that the root password is none by default, and ssh rightfully refuses to login someone without a password.

To rectify this, set the root password with

```
# passwd root
```

at shell prompt.

4.5 Using the desktop

The Xillinux desktop is just like any Ubuntu desktop. Due to the SD card's relatively low data bandwidth, applications will load somewhat slowly, but the desktop itself is fairly responsive.

To run applications in the desktop environment, click the top-left Ubuntu icon on the desktop ("Dash home") and type the name of the desired application, e.g. "terminal" for a shell prompt terminal window, or "edit" for the gedit text editor.

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

4.6 Shutting down

To power down the system, pick the top-right icon on the desktop, and click "Shut Down...". Alternatively, type

```
# shutdown -h now
```

at shell prompt.

When a textual message saying "System Halted" appears, it's safe to power the board off.

4.7 Taking it from here

The Zedboard 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.

Note that Xillinux includes the gcc compiler and GNU make, so host applications can be compiled natively 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

The Xillinux distribution is set up for easy integration with application logic. The front end for connecting data sources and sinks is the xillydemo.v or xillydemo.vhd file (depending on the preferred language). All other HDL files in the boot image 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 handled in paragraph 3.5, and then rebuilt the same way it was done the first place. To boot the system with the updated logic, the boot.bin needs to be regenerated as described in paragraph 3.6 and written to the SD file as described in paragraph 3.8. 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.

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 memories or register arrays to Xillybus, in which case the schema shown in the xillydemo module should be followed.

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

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

The FIFOs used in the xillydemo module accept only one common clock for both

sides, as both sides are driven Xillybus' main clock. In a real-life application, it may be desirable to replace them with FIFOs having separate clocks for reading and writing, allowing data sources and sinks to be driven by a clock other than the bus clock. By doing this, the FIFOs serve not just as mediators, but also for proper clock domain crossing.

Note that the Xillybus IP core expects a *plain* FIFO interface, (as opposed to First Word Fall Through) for FPGA to host streams.

The following documents are related to integrating custom logic:

- The API for logic design: Xillybus FPGA designer's guide
- 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 the Xillinux on a board other than Zedboard, certain modifications may be necessary. There are two main issues to address:

- The VGA output needs to be matched to the target 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 DVI/VGA encoder chip.
- 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 Xilinx Platform Studio 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.

The latter issue is important, not only because crucial hardware functions may fail, but illegal physical signal conditions on the board may result in damage to the hardware (even though actual damage is very rare).

Inconsistencies in MIO assignments are rectified in two steps:

- Correcting these assignments in the project opened with system.xmp, using Xilinx Platform Studio's GUI.
- Updating the DTS file to match the device assignments, and compiling this file, producing the DTB file to boot the system with.

5.3 Changing the system's clock frequencies

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 software writing to the system's registers, and not by some direct hardware setting.

Hence even though the clocks' frequencies are set in the Xilinx Platform Studio, these frequencies are effective only for propagating timing constraints and initialization by bare-metal applications compiled on the SDK. The Linux kernel overrides the clock setting at bootup with hardcoded defaults (as of version 3.3.0 of the Linux kernel).

The defaults chosen for Xillinux' are FCLK0 and FCLK1 at 100 MHz, and FLKC2 and FCLK3 at 50 MHz.

If the hardware application requires different clock frequencies, the respective registers need to be changed as its driver loads. The default settings are hardcoded in arch/arm/mach-zynq/slcr.c (relative to the kernel tree's root), in the xslcr_probe() function.

To set the registers to other values, use the xslcr_write() and xslcr_read() functions exported by this driver.

6

Troubleshooting

6.1 Implementation errors

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

If the problem isn't solved fairly quickly, please seek assistance through the email address given at the company's web site. Please attach the output log of the process that failed, in particular around the first error reported by the tool. Also, if custom changes were made in the design, please detail these changes. Also please state which version of the ISE tools was used.

6.2 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:

- Are you using the correct USB plug? It should be the one marked "USB OTG", farther away from the power switch.
- 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 USB cable that came with the Zedboard.

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. Please refer to Zedboard’s documentation regarding how to set up a UART link.

6.3 File system mount issues

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

Powering off the board without unmounting 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 mount (resulting in a kernel panic during boot) or mounts read-only, the most likely cause is a low quality SD card. It’s quite typical for such storage to function properly for a while, after which random error messages surface. If /var/log/syslog contains messages such as this one,

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

the SD card is most likely the reason.

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

6.4 “startx” fails (Graphical desktop won’t start)

Even though not directly related, this problem is reported quite frequently when the SD card is a non-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 SD card that generates read errors.

The obvious solution is using a Sandisk SD card.

6.5 Errors on “Generate Netlist”

If the following (or similar) error appears

```
ERROR:EDK:4073 - INSTANCE: processing_system7_0, PORT: GPIO_I, CONNECTOR:
 processing_system7_0_GPIO_I - 56 bit-width connector assigned to 64
 bit-width port - C:\builds\xillinux-eval-zedboard-1.0\system\system.mhs
 line 104
```

when attempting to generate the netlist (as instructed in paragraph 3.3) with ISE 14.3, please verify that the specific instructions for ISE 14.3 in that paragraph have been followed.