OSELAS.Support
OSELAS.Training
OSELAS.Development
OSELAS.Services

Quickstart Manual
OSELAS.BSP( )
Phytec phyCORE-Z500PT

PHYTEC

Pengutronix.

Pengutronix e. K.
Peiner Straße 6–8
31137 Hildesheim

+49 (0)51 21 / 20 69 17 – 0 (Fon)
+49 (0)51 21 / 20 69 17 – 55 55 (Fax)

info@pengutronix.de

Contents

Part I

OSELAS Quickstart for Phytec phyCORE-Z500PT

1 Getting a working Environment

1.1 Download Software Components

In order to follow this manual, some software archives are needed. There are several possibilities how to get these: either as part of an evaluation board package or by downloading them from the Pengutronix web site.

The central place for OSELAS related documentation is http://www.oselas.com. This website provides all required packages and documentation (at least for software components which are available to the public).

To build OSELAS.BSP-Phytec-phyCORE-12, the following archives have to be available on the development host:

- ptxdist-1.99.12.tgz
- ptxdist-1.99.12-patches.tgz
- OSELAS.BSP-Phytec-phyCORE-12.tar.gz
- OSELAS.Toolchain-1.99.3.2.tar.bz2

If they are not available on the development system yet, it is necessary to get them.

1.2 PTXdist Installation

The PTXdist build system can be used to create a root filesystem for embedded Linux devices. In order to start development with PTXdist it is necessary to install the software on the development system.

This chapter provides information about how to install and configure PTXdist on the development host.

1.2.1 Main Parts of PTXdist

The most important software component which is necessary to build an OSELAS.BSP( ) board support package is the `ptxdist` tool. So before starting any work we'll have to install PTXdist on the development host.

PTXdist consists of the following parts:

The `ptxdist` Program: `ptxdist` is installed on the development host during the installation process. `ptxdist` is called to trigger any action, like building a software packet, cleaning up the tree etc. Usually the `ptxdist` program is used in a *workspace* directory, which contains all project relevant files.

A Configuration System: The config system is used to customize a *configuration*, which contains information about which packages have to be built and which options are selected.

Patches: Due to the fact that some upstream packages are not bug free – especially with regard to cross compilation – it is often necessary to patch the original software. PTXdist contains a mechanism to automatically apply patches to packages. The patches are bundled into a separate archive. Nevertheless, they are necessary to build a working system.

Package Descriptions: For each software component there is a "recipe" file, specifying which actions have to be done to prepare and compile the software. Additionally, packages contain their configuration sniplet for the config system.

Toolchains: PTXdist does not come with a pre-built binary toolchain. Nevertheless, PTXdist itself is able to build toolchains, which are provided by the OSELAS.Toolchain() project. More in-deep information about the OSELAS.Toolchain() project can be found here: http://www.pengutronix.de/oselas/toolchain/index_en.html

Board Support Package This is an optional component, mostly shipped aside with a piece of hardware. There are various BSP available, some are generic, some are intended for a specific hardware.

1.2.2 Extracting the Sources

To install PTXdist, at least two archives have to be extracted:

ptxdist-1.99.12.tgz The PTXdist software itself.

ptxdist-1.99.12-patches.tgz All patches against upstream software packets (known as the 'patch repository').

ptxdist-1.99.12-projects.tgz Generic projects (optional), can be used as a starting point for self-built projects.

The PTXdist and patches packets have to be extracted into some temporary directory in order to be built before the installation, for example the `local/` directory in the user's home. If this directory does not exist, we have to create it and change into it:

```
~# cd
~# mkdir local
~# cd local
```

Next steps are to extract the archives:

```
~/local# tar -zxf ptxdist-1.99.12.tgz
~/local# tar -zxf ptxdist-1.99.12-patches.tgz
```

and if required the generic projects:

```
~/local# tar -zxf ptxdist-1.99.12-projects.tgz
```

If everything goes well, we now have a PTXdist-1.99.12 directory, so we can change into it:

```
~/local# cd ptxdist-1.99.12
~/local/ptxdist-1.99.12# ls -l
total 487
drwxr-xr-x  13 jb users   1024 Mar 23 13:25 ./
drwxr-xr-x  22 jb users   3072 Mar 23 13:25 ../
-rw-r--r--   1 jb users    377 Feb 23 22:23 .gitignore
-rw-r--r--   1 jb users  18361 Apr 24  2003 COPYING
-rw-r--r--   1 jb users   3731 Mar 11 18:09 CREDITS
-rw-r--r--   1 jb users 115540 Mar  7 15:25 ChangeLog
-rw-r--r--   1 jb users     58 Apr 24  2003 INSTALL
-rw-r--r--   1 jb users   2246 Feb  9 14:29 Makefile.in
-rw-r--r--   1 jb users   4196 Jan 20 22:33 README
-rw-r--r--   1 jb users    691 Apr 26  2007 REVISION_POLICY
```

```
-rw-r--r--   1 jb users  54219 Mar 23 10:51 TODO
drwxr-xr-x   2 jb users   1024 Mar 23 11:27 autoconf/
-rwxr-xr-x   1 jb users     28 Jun 20  2006 autogen.sh*
drwxr-xr-x   2 jb users   1024 Mar 23 11:27 bin/
drwxr-xr-x   6 jb users   1024 Mar 23 11:27 config/
-rwxr-xr-x   1 jb users 226185 Mar 23 11:27 configure*
-rw-r--r--   1 jb users  12390 Mar 23 11:16 configure.ac
drwxr-xr-x   2 jb users   1024 Mar 23 11:27 debian/
drwxr-xr-x   8 jb users   1024 Mar 23 11:27 generic/
drwxr-xr-x 164 jb users   4096 Mar 23 11:27 patches/
drwxr-xr-x   2 jb users   1024 Mar 23 11:27 platforms/
drwxr-xr-x   4 jb users   1024 Mar 23 11:27 plugins/
drwxr-xr-x   6 jb users  30720 Mar 23 11:27 rules/
drwxr-xr-x   7 jb users   1024 Mar 23 11:27 scripts/
drwxr-xr-x   2 jb users   1024 Mar 23 11:27 tests/
```

1.2.3 Prerequisites

Before PTXdist can be installed it has to be checked if all necessary programs are installed on the development host. The configure script will stop if it discovers that something is missing.

The PTXdist installation is based on GNU autotools, so the first thing to be done now is to configure the packet:

```
~/local/ptxdist-1.99.12# ./configure
```

This will check your system for required components PTXdist relies on. If all required components are found the output ends with:

```
[...]
checking whether /usr/bin/patch will work... yes

configure: creating ./config.status
config.status: creating Makefile
config.status: creating scripts/ptxdist_version.sh
config.status: creating rules/ptxdist-version.in

ptxdist version 1.99.12 configured.
Using '/usr/local' for installation prefix.

Report bugs to ptxdist@pengutronix.de
```

Without further arguments PTXdist is configured to be installed into `/usr/local`, which is the standard location for user installed programs. To change the installation path to anything non-standard, we use the `--prefix` argument to the `configure` script. The `--help` option offers more information about what else can be changed for the installation process.

The installation paths are configured in a way that several PTXdist versions can be installed in parallel. So if an old version of PTXdist is already installed there is no need to remove it.

One of the most important tasks for the `configure` script is to find out if all the programs PTXdist depends on are already present on the development host. The script will stop with an error message in case something is missing.

If this happens, the missing tools have to be installed from the distribution befor re-running the `configure` script.

When the `configure` script is finished successfully, we can now run

```
~/local/ptxdist-1.99.12# make
```

All program parts are being compiled, and if there are no errors we can now install PTXdist into it's final location. In order to write to `/usr/local`, this step has to be performed as user *root*:

```
~/local/ptxdist-1.99.12# sudo make install
[enter root password]
[...]
```

If we don't have root access to the machine it is also possible to install into some other directory with the `--prefix` option. We need to take care that the `bin/` directory below the new installation dir is added to our `$PATH` environment variable (for example by exporting it in `~/.bashrc`).

The installation is now done, so the temporary folder may now be removed:

```
~/local/ptxdist-1.99.12# cd
~# rm -fr local
```

1.2.4 Configuring PTXdist

When using PTXdist for the first time, some setup properties have to be configured. Two settings are the most important ones: Where to store the source packages and if a proxy must be used to gain access to the world wide web.

Run PTXdist's setup:

```
~# ptxdist setup
```

Due to PTXdist is working with sources only, it needs various source archives from the world wide web. If these archives are not present on our host, PTXdist starts the `wget` command to download them on demand.

Proxy Setup

To do so, an internet access is required. If this access is managed by a proxy `wget` command must be adviced to use it. PTXdist can be configured to advice the `wget` command automatically: Navigate to entry *Proxies* and enter the required addresses and ports to access the proxy in the form:

`<protocol>://<address>:<port>`

Source Archive Location

Whenever PTXdist downloads source archives it stores these archives in a project local manner. If we are working with more than one project, every project would download its own required archives. To share all source archives between all projects PTXdist can be configured to use only one archive directory for all projects it handles: Navigate to menu entry *Source Directory* and enter the path to the directory where PTXdist should store archives to share between projects.

Generic Project Location

If we already installed the generic projects we should also configure PTXdist to know this location. If we already did so, we can use the command `ptxdist projects` to get a list of available projects and `ptxdist clone` to get a local working copy of a shared generic project.

Navigate to menu entry *Project Searchpath* and enter the path to projects that can be used in such a way. Here we can configure more than one path, each part can be delemited by a colon. For example for PTXdist's generic projects and our own previous projects like this:

`/usr/local/lib/ptxdist-1.99.12/projects:/office/my_projects/ptxdist`

Leave the menu and store the configuration. PTXdist is now ready for use.

1.3 Toolchains

Before we can start building our first userland we need a cross toolchain. On Linux, toolchains are no monolithic beasts. Most parts of what we need to cross compile code for the embedded target comes from the *GNU Compiler Collection*, `gcc`. The gcc packet includes the compiler frontend, `gcc`, plus several backend tools (cc1, g++, ld etc.) which actually perform the different stages of the compile process. `gcc` does not contain the assembler, so we also need the *GNU Binutils package* which provides lowlevel stuff.

Cross compilers and tools are usually named like the corresponding host tool, but with a prefix – the *GNU target*. For example, the cross compilers for ARM and powerpc may look like

- `arm-softfloat-linux-gnu-gcc`
- `powerpc-unknown-linux-gnu-gcc`

With these compiler frontends we can convert e.g. a C program into binary code for specific machines. So for example if a C program is to be compiled natively, it works like this:

```
~# gcc test.c -o test
```

To build the same binary for the ARM architecture we have to use the cross compiler instead of the native one:

```
~# arm-softfloat-linux-gnu-gcc test.c -o test
```

Also part of what we consider to be the "toolchain" is the runtime library (libc, dynamic linker). All programs running on the embedded system are linked against the libc, which also offers the interface from user space functions to the kernel.

The compiler and libc are very tightly coupled components: the second stage compiler, which is used to build normal user space code, is being built against the libc itself. For example, if the target does not contain a hardware floating point unit, but the toolchain generates floating point code, it will fail. This is also the case when the toolchain builds code for i686 CPUs, whereas the target is i586.

So in order to make things working consistently it is necessary that the runtime libc is identical with the libc the compiler was built against.

PTXdist doesn't contain a pre-built binary toolchain. Remember that it's not a distribution but a development tool. But it can be used to build a toolchain for our target. Building the toolchain usually has only to be done once. It may be a good idea to do that over night, because it may take several hours, depending on the target architecture and development host power.

1.3.1 Using Existing Toolchains

If a toolchain is already installed which is known to be working, the toolchain building step with PTXdist may be omitted.

The OSELAS.BoardSupport() Packages shipped for PTXdist have been tested with the OSELAS.Toolchains() built with the same PTXdist version. So if an external toolchain is being used which isn't known to be stable, a target may fail. Note that not all compiler versions and combinations work properly in a cross environment.

Every OSELAS.BoardSupport() Package checks for its OSELAS.Toolchain it's tested against, so using a different toolchain vendor requires an additional step:

Open the OSELAS.BoardSupport() Package menu with:

```
~# ptxdist platformconfig
```

and navigate to `architecture --> toolchain` and `check for specific toolchain vendor`. Clear this entry to disable the toolchain vendor check.

1.3.2 Building a Toolchain

PTXdist handles toolchain building as a simple project, like all other projects, too. So we can download the OSELAS.Toolchain bundle and build the required toolchain for the OSELAS.BoardSupport() Package.

A PTXdist project generally allows to build into some project defined directory; all OSELAS.Toolchain projects that come with PTXdist are configured to use the standard installation paths mentioned below.

All OSELAS.Toolchain projects install their result into `/opt/OSELAS.Toolchain-1.99.3/`.

Usually the `/opt` directory is not world writeable. So in order to build our OSELAS.Toolchain into that directory we need to use a root account to change the permissions. PTXdist detects this case and asks if we want to run `sudo` to do the job for us. Alternatively we can enter:

```
mkdir /opt/OSELAS.Toolchain-1.99.3
chown <username> /opt/OSELAS.Toolchain-1.99.3
chmod a+rwx /opt/OSELAS.Toolchain-1.99.3.
```

We recommend to keep this installation path as PTXdist expects the toolchains at `/opt`. Whenever we go to select a platform in a project, PTXdist tries to find the right toolchain from data read from the platform configuration settings and a toolchain at `/opt` that matches to these settings. But that's for our convenience only. If we decide to install the toolchains at a different location, we still can use the *toolchain* parameter to define the toolchain to be used on a per project base.

1.3.3 Building the OSELAS.Toolchain for OSELAS.BSP-Phytec-phyCORE-12

To compile and install an OSELAS.Toolchain we have to extract the OSELAS.Toolchain archive, change into the new folder, configure the compiler in question and start the build.

The required compiler to build the OSELAS.BSP-Phytec-phyCORE-12 board support package is

```
i586-unknown-linux-gnu_gcc-4.3.2_glibc-2.8_binutils-2.18_kernel-2.6.27-sanitized
```

So the steps to build this toolchain are:

```
~# tar xf OSELAS.Toolchain-1.99.3.2.tar.bz2
~# cd OSELAS.Toolchain-1.99.3.2
~/OSELAS.Toolchain-1.99.3.2# ptxdist select ptxconfigs/\ Enter
> i586-unknown-linux-gnu_gcc-4.3.2_glibc-2.8_binutils-2.18_kernel-2.6.27-sanitized.ptxconfig
~/OSELAS.Toolchain-1.99.3.2# ptxdist go
```

At this stage we have to go to our boss and tell him that it's probably time to go home for the day. Even on reasonably fast machines the time to build an OSELAS.Toolchain is something like around 30 minutes up to a few hours.

Measured times on different machines:

- Single Pentium 2.5 GHz, 2 GiB RAM: about 2 hours
- Turion ML-34, 2 GiB RAM: about 1 hour 30 minutes
- Dual Athlon 2.1 GHz, 2 GiB RAM: about 1 hour 20 minutes
- Dual Quad-Core-Pentium 1.8 GHz, 8 GiB RAM: about 25 minutes

Another possibility is to read the next chapters of this manual, to find out how to start a new project.

When the OSELAS.Toolchain project build is finished, PTXdist is ready for prime time and we can continue with our first project.

1.3.4 Protecting the Toolchain

All toolchain components are built with regular user permissions. In order to avoid accidential changes in the toolchain, the files should be set to read-only permissions after the installation has finished successfully. It is also possible to set the file ownership to root. This is an important step for reliability, so it is highly recommended.

Building Additional Toolchains

The OSELAS.Toolchain-1.99.3.2 bundle comes with various predefined toolchains. Refer the `ptxconfigs/` folder for other definitions. To build additional toolchains we only have to clean our current toolchain project, removing the current `selected_ptxconfig` link and creating a new one.

```
~/OSELAS.Toolchain-1.99.3.2# ptxdist clean
~/OSELAS.Toolchain-1.99.3.2# rm selected_ptxconfig
~/OSELAS.Toolchain-1.99.3.2# ptxdist select \ Enter
> ptxconfigs/any_other_toolchain_def.ptxconfig
~/OSELAS.Toolchain-1.99.3.2# ptxdist go
```

All toolchains will be installed side by side architecture dependent into directory

`/opt/OSELAS.Toolchain-1.99.3/architecture_part`.

Different toolchains for the same architecture will be installed side by side version dependent into directory

`/opt/OSELAS.Toolchain-1.99.3/architecture_part/version_part`.

2 Building phyCORE-Z500PT's root filesystem

2.1 Extracting the Board Support Package

In order to work with a PTXdist based project we have to extract the archive first.

```
~# tar -zxf OSELAS.BSP-Phytec-phyCORE-12.tar.gz
~# cd OSELAS.BSP-Phytec-phyCORE-12
```

PTXdist is project centric, so now after changing into the new directory we have access to all valid components.

```
~/OSELAS.BSP-Phytec-phyCORE-12# ls -l

total 44
-rw-r--r--  1 jb users 4078 Dec  3 18:10 ChangeLog
-rw-r--r--  1 jb users 1313 Nov  1 13:31 Kconfig
-rw-r--r--  1 jb users 1101 Nov  4 21:05 TODO
drwxr-xr-x 10 jb users 4096 Jan 14 17:33 configs/
drwxr-xr-x  3 jb users 4096 Jan 14 15:08 documentation/
drwxr-xr-x  5 jb users 4096 Nov 13 12:30 local_src/
drwxr-xr-x  5 jb users 4096 Dec 15 10:19 patches/
drwxr-xr-x  6 jb users 4096 Jun  8  2008 projectroot/
drwxr-xr-x  3 jb users 4096 Nov  1 14:18 protocols/
drwxr-xr-x  4 jb users 4096 Jan  8 16:28 rules/
drwxr-xr-x  3 jb users 4096 Jan  7 08:55 tests/
```

Notes about some of the files and directories listed above:

ChangeLog Here you can read what has changed in this release. Note: This file does not always exist.

documentation If this BSP is one of our OSELAS BSPs, this directory contains the Quickstart you are currenly reading in.

configs A multiplatform BSP contains configurations for more than one target. This directory contains the platform configuration files.

projectroot Contains files and configuration for the target's runtime. A running GNU/Linux system uses many text files for runtime configuration. Most of the time the generic files from the PTXdist installation will fit the needs. But if not, customized files are located in this directory.

rules If something special is required to build the BSP for the target it is intended for, then this directory contains these additional rules.

patches If some special patches are required to build the BSP for this target, then this directory contains these patches on a per package basis.

tests Contains test scripts for automated target setup.

2.2 Selecting a Software Platform

First of all we have to select a software platform for the userland configuration. This step defines what kind of applications will be built for the hardware platform. The OSELAS.BSP-Phytec-phyCORE-12 comes with a predefined configuration we select in the following step:

```
~/OSELAS.BSP-Phytec-phyCORE-12# ptxdist select \ Enter
> configs/ptxconfig
info: selected ptxconfig:
     'configs/ptxconfig'
```

2.3 Selecting a Hardware Platform

Before we can build this BSP, we need to select one of the possible targets to build for. In this case we want to build for the phyCORE-Z500PT:

```
~/OSELAS.BSP-Phytec-phyCORE-12# ptxdist platform \ Enter
> configs/phyCORE-Z500PT-1.99.12-2/platformconfig
info: selected platformconfig:
      'configs/phyCORE-Z500PT-1.99.12-2/platformconfig'
```

Note: If you have installed the OSELAS.Toolchain() at its default location, PTXdist should already have detected the proper toolchain while selecting the platform. In this case it will output:

```
found and using toolchain:
'/opt/OSELAS.Toolchain-1.99.3/i586-unknown-linux-gnu/
      gcc-4.3.2-glibc-2.8-binutils-2.18-kernel-2.6.27-sanitized/bin'
```

If it fails you can continue to select the toolchain manually as mentioned in the next section. If this autodetection was successful, we can omit the steps of the section and continue to build the BSP.

2.4 Selecting a Toolchain

If not automatically detected, the last step in selecting various configurations is to select the toolchain to be used to build everything for the target.

```
~/OSELAS.BSP-Phytec-phyCORE-12# ptxdist toolchain \ Enter
> /opt/OSELAS.Toolchain-1.99.3/i586-unknown-linux-gnu/\ Enter
> gcc-4.3.2-glibc-2.8-binutils-2.18-kernel-2.6.27-sanitized/bin
```

2.5 Building the Root Filesystem

Now everything is prepared for PTXdist to compile the BSP. Starting the engines is simply done with:

```
~/OSELAS.BSP-Phytec-phyCORE-12# ptxdist go
```

PTXdist does now automatically find out from the `selected_ptxconfig` and `selected_platformconfig` files which packages belong to the project and starts compiling their *targetinstall* stages (that one that actually puts the compiled binaries into the root filesystem). While doing this, PTXdist finds out about all the dependencies between the packets and brings them into the correct order.

While the command `ptxdist go` is running we can watch it building all the different stages of a packet. In the end the final root filesystem for the target board can be found in the `platform-phyCORE-Z500PT/root/` directory and a bunch of **.ipk* packets in the `platform-phyCORE-Z500PT/packages/` directory, containing the single applications the root filesystem consists of.

2.6 Building an Image

After we have built a root filesystem, we can make an image, which can be flashed to the target device. To do this call

```
~/OSELAS.BSP-Phytec-phyCORE-12# ptxdist images
```

PTXdist will then extract the content of priorly created **.ipk* packages to a temporary directory and generate an image out of it. PTXdist supports following image types:

- **hd.img:** contains grub bootloader, kernel and root files in a ext2 partition. Mostly used for X86 target systems.
- **root.jffs2:** root files inside a jffs2 filesystem.
- **uRamdisk:** a u-boot loadable Ramdisk
- **initrd.gz:** a traditional initrd RAM disk to be used as initrdramfs by the kernel
- **root.ext2:** root files inside a ext2 filesystem.
- **root.squashfs:** root files inside a squashfs filesystem.
- **root.tgz:** root files inside a plain gzip compressed tar ball.

The to be generated Image types and addtional options can be defined with

```
~/OSELAS.BSP-Phytec-phyCORE-12# ptxdist platformconfig
```

Then select the submenu `"image creation options"`. The generated image will be placed into `platform-phyCORE-Z500PT/images/`.

Only the content of the **.ipk* packages will be used to generate the image. This means that files which are put manually into the `platform-phyCORE-Z500PT/root/` will not be enclosed in the image. If custom files are needed for the target. Install it with ptxdist.

2.6.1 Generating a hd.img

The hd.img contains a grub bootloader and a partition table. The kernel and the root filesystem are packed into an EXT2 filesystem. It is to be used mostly for bootalbe storage devices like harddisk, sd/mmc cards, CF cards or USB sticks on X86 Systems. To generate a hd.img settins must be done for a root.ext2 Image and a partition layout.

2.6.1.1 Setting up a Partition Layout

The configuration options for the partition layout can be found in the submenu "`image creation options ---> Generate images/hd.img`" after calling

```
~/OSELAS.BSP-Phytec-phyCORE-12# ptxdist platformconfig
```

As for default one partition is defined. To create more partitions activate the entry "`Create partition 2`", which will bring up selection for one more partition. PTXdist supports up to four primary partions. Logical partitions are not supported.
Addtionally the size and type of the partitions have to be defined.

- Partition size can be defined by giving the first and the last sector of each partition. Sectors are units of 512 bytes. Sectors can be given either decimal or in hex prefixed with 0x. The first sector (sector 0) is reserved for the MBR and the partition table. Also keep in mind that we want to spare some place for the 2nd stage Grub bootloader, which is stored right after the MBR. Practically a value of e.g. 300 is a good start for the start sector of the first partition. The first sectors of the following partitions shall follow the prior ending sector. So if the first partition ends at sector 300000. The second partition should start at 3000001. The size of a partition will be `(ENDSECTOR - STARTSECTOR + 1) * 512/ 1024(KiB)` Beware that a flash device does not have exactly `SIZE (MiB) * 1024 (KiB/MiB)* 2 sectors` . but slightly less due to defekt sectors. So please define the last ending sector to be slightly smaller than e.g. `2 (GiB) * 1024 (MiB/GiB) * 1024 (KiB/MiB) * 2 (Sectors/KiB)` for a device with 2GiB capacity. Mostly something about 200 Sectors will be fine.
- Partition type can be defined as a decimal or hex (prefixed with 0x) value. See at the output of

  `~/OSELAS.BSP-Phytec-phyCORE-12# sfdisk -T`

  for a list of valid types. 0x83 is e.g. Linux filesystem, while 0xc is vfat with LBA addressing.

- Since the traditional C/H/S addressing scheme is hardware dependent, the PTXdist generated partition table does not have CHS entries but only LBA entries, so the (x86-) Bios on the target has to be switched to LBA Modus, which is the standard konfiguration of the most BIOS. Also note that fdisk will complain about inconsistent CHS/LBA entries. The warnings can be ignored safely.
- The first partition will be the system partition, which contains the kernel image and the rootfs. If more parttions are created, only an entry in the partition table will be made and no filesystems will be created. Use your favourite mkfs tool to create a file system on the corresponding partition.

2.6.2 Generating a root.ext2 image

The hd.img forces the generation of a EXT2 filesystem image. The options for the EXT2 image can be set in the submenu "`image creation options ---> Generate images/root.ext2`" after calling

```
~/OSELAS.BSP-Phytec-phyCORE-12# ptxdist platformconfig
```

The size of the EXT2 Image can be defined there. Addtionally some extra arguments can be passed to setup customizable values like inodes, permissions etc.

The EXT2 Image will be used as the first partition for the hd.img. So it must be equal or smaller than the size given in the partition layout. For example, if we have defined a size of 157286000 KiB, which is 150 MiB. We will need in the configuration of our hd.img a first partition with at least `157286000 (KiB) * 2 (Sectors/KiB) = 314572000` sectors.

3 Booting Linux

Now that there is a root filesystem in our workspace we'll have to make it visible to the phyCORE-Z500PT. There are two possibilities to do this:

1. Making the root filesystem persistent in the onboard media.
2. Booting from the development host, via network.

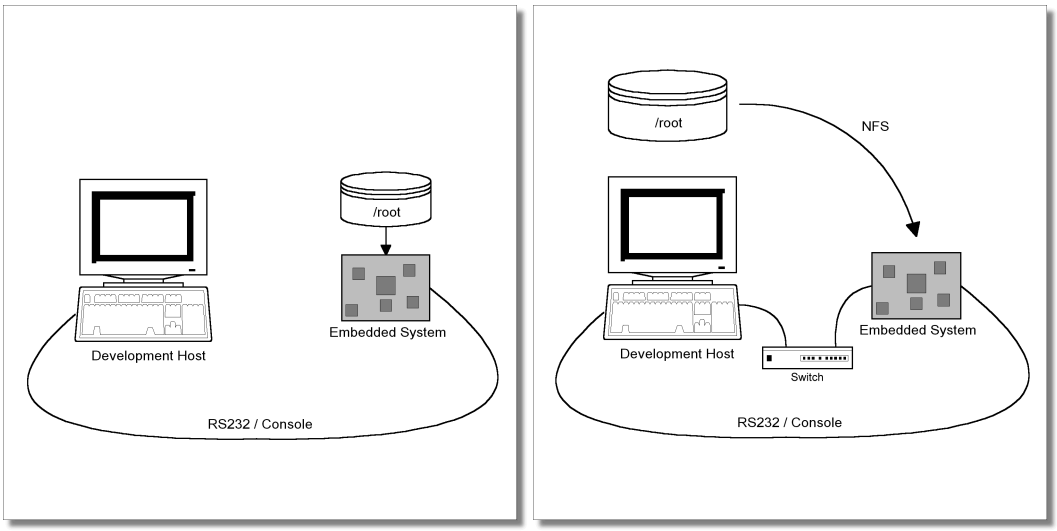

Figure 3.1: *Booting the root filesystem, built with PTXdist, from the host via network and from flash.*

Figure 3.1 shows both methods. The main method used in the OSELAS.BSP-Phytec-phyCORE-12 BSP is to provide all needed components to run on the target itself. The Linux kernel and the root filesystem is persistent in the media the target features. This means the only connection needed is the nullmodem cable to see what is happening on our target. We call this method *standalone*.

The other method is to provide all needed components via network. In this case the development host is connected to the phyCORE-Z500PT with a serial nullmodem cable **and** via ethernet; the embedded board boots into the bootloader, then issues a TFTP request on the network and boots the kernel from the TFTP server on the host. Then, after decompressing the kernel into the RAM and starting it, the kernel mounts its root filesystem via NFS (Network File System) from the original location of the `platform-phyCORE-Z500PT/root/` directory in our PTXdist workspace.

The OSELAS.BSP-Phytec-phyCORE-12 provides both methods. The latter one is especially for development purposes, as it provides a very quick turnaround while testing the kernel and the root filesystem.

This chapter describes how to set up our target with features supported by PTXdist to simplify this challange.

3.1 Configuring BIOS

Before we can start the system. We should choose our bootmedia in BIOS. To do so power up the target press "del" as soon as the first BIOS messages appears. The setup screen will than appear. Choose the item `"System Setup utility"` and goto the tab `"Boot"`. We can set our favourite boot media to highest prior boot device here. phyCORE-Z500PT supports several kind of boot media:

- USB flash stick
- ATA disc
- interal SSD
- Intel(R) PRO/1000 PCI-Express Gigabit Ethernet

Please see section 3.2 for more information about preparing a local boot media like USB flash disk or ATA disc, and section 3.3 for more information about doing a boot via network.

3.2 Stand-Alone Booting Linux

3.2.1 Configuring Grub

We use the the GR(and) U(nified) B(ootloader) to startup the operation system for our phyCORE-Z500PT. Grub is installed in two stages. The stage1 code is written directly to the MBR of the boot media. The stage2 is loaded by stage1. It can mounts a root file system, read configurations and load the kernel according to its configuration file. The configuration file for grub ist placed into /boot/grub/menu.lst. A template for this file can be found in /OSELAS.BSP-Phytec-phyCORE-12/projectroot/menu.lst.phyCORE-Z500PT-1.99.12-2. PTXdist can be used to generate a target specific menu.lst file. To do so call

```
~/OSELAS.BSP-Phytec-phyCORE-12# ptxdist platformconfig
```

and then enter the submenu `"bootloaders -> grub"`. The most options in the menu than appeard can be safely left "as it is". However changes are probably required in the Option `"Device of Rootfs"`. With this option we define where the rootfs is located. This depends on which kind of media is used to boot and also how many storage devices are connected to the target. On our phyCORE-Z500PT the rootfs will be e.g. found at `/dev/sdb1` if the target boots from a USB Flash disk. Default setting is /dev/sdb1.

3.2.2 flashing a image

To use the the target standalone, the rootfs has to be made persistent in one of the onboard suported media of the phyCORE-Z500PT. See section 3.1 for a list of supported boot media. After running

```
~# ptxdist images
```

there will be a image file named `hd.img` in platform-phyCORE-Z500PT//images. The image file can be used to flash a bootable media device. Simply use the tool `dd` to write the image. Suppose `/dev/sdb` is our media device, we can run

```
~/OSELAS.BSP-Phytec-phyCORE-12# sudo dd if=platform-phyCORE-Z500PT/images/hd.img \
of=/dev/sdb
```

The write process can cost quite some time, depending on the usb access speed of the host system.

Make sure that the right device is written, because otherwise important the host system cnould get damaged. `/proc/partitions` can be e.g. used to check available storage devices and their corresponding device file. It's also recommended to use id alias device files, which is unfortunately not available on all Linux systems. To check if you have device id support, have a look into `/dev/disk/by-id`.

```
~/OSELAS.BSP-Phytec-phyCORE-12# ls -C /dev/disk/by-id/
scsi-SATA_SAMSUNG_HD161HJS0V3J9CPC08676
scsi-SATA_SAMSUNG_HD161HJS0V3J9CPC08676-part1
scsi-SATA_SAMSUNG_HD161HJS0V3J9CPC08676-part2
scsi-SATA_SAMSUNG_HD161HJS0V3J9CPC08676-part5
usb-CBM_Flash_Disk_2720220016533500-0:0
```

The files with actually softlinks to the "real" device files. `scsi-SATA_SAMSUNG_HD161HJS0V3J9CPC08676` e.g. links to `../../sda` and `usb-CBM_Flash_Disk_2720220016533500-0:0` links to `../../sdb`. Hence we can simply do

```
~/OSELAS.BSP-Phytec-phyCORE-12# sudo dd if=platform-phyCORE-Z500PT/images/hd.img \
of=/dev/disk/by-id/usb-CBM_Flash_Disk_2720220016533500-0:0
```

3.3 Remote-Booting Linux

The next method we want to try after building a root filesystem is the network-remote boot variant. This method is especially intended for development as everything related to the root filesystem happens on the host only. It's the fastest way in a phase of a project, where things are changing frequently. Any change made in the local `platform-phyCORE-Z500PT/root/` directory simply "appears" on the embedded device immediately.

All we need is a network interface on the embedded board and a network aware bootloader. On the phyCORE-Z500PT the network interface is aware of the PXE protocol. PXE cannot fetch a kernel image directly becaus it has unfortunately a file transfer limit of 640KiB. Hence we need to a two-stage network boot here. First we fetch a pxelinux binary. The pxelinux binary will be run localy and read a configuration file we have to prepare priorly, before it fetchs a kernel from a tftp server and kicks it start.

3.3.1 Development Host Preparations

3.3.1.1 Required Services and Software

We need at least working TFTP, DHCP NFS services on the hostmachine to do a net boot

- **TFTP:** We need a TFTP server on our host which supports the "tsize" TFTP option. A possible candidate, which was tested successfully is the "tftp-hpa" server, which can be retrieved as a common packet in the most distributions. Usually TFTP servers are using the `/tftpboot` directory to fetch files from, so if we want to push data files to this directory, we have to make sure we are able to write there. As the access permissions are normally configured in a way to let only user **root** write to `/tftpboot` we have to change it. If we don't want to change the permission or if its disallowed to

change anything, the sudo command may help. `~/OSELAS.BSP-Phytec-phyCORE-12# sudo cp platform-phyCORE-Z500PT/images/linuximage /tftpboot/bzImage-phyCORE-Atom`
Please Note:

  - Not all TFTP Server software supports the "tsize" option. If not sure, please stick to our recommendation of the "tftp-hpa" server.
  - The exact method of starting the tftp server is distribution specific; as the TFTP server is usually started by one of the `inetd` servers, the manual sections describing `inetd` or `xinetd` should be consulted.

- DHCP: The PXE bootloader will try get an IP and bootfile using DHCP as soon as it is started. For this we need a DHCP server. Some standard DHCP extensions are also required. We use the dhcp3-server package provided by Debian Lenny, which supports all options the PXE protocol needs. Also for the DHCP server we have to do some configuration about IP range, route/DNS server information etc.. Please refer to the manual of your distribution for more information about configuring DHCP server
- PXE: Alternatively to DHCP + TFTP a "real" PXE server can also be used. There're an commercial implementation of PXE server by intel and also a free variant available in most linux distributions. For more information about PXE server please refer to the manual by the software vendor. In this document we only handle the combination DHCP + TFTP.
- **NFS:** The NFS server is not restricted to a certain filesystem location, so all we have to do on most distributions is to modify the file `/etc/exports` and export our root filesystem to the embedded network. In this example file the whole work directory is exported, and the "lab network" between the development host is 192.168.23.0, so the IP addresses have to be adapted to the local needs:

  `/home/<user>/work 192.168.23.0/255.255.255.0(rw,no_root_squash,sync)`

  Note: Replace `<user>` with your home directory name.

3.3.1.2 Configuring PXE

To support the PXE protocol, we have to do some additional configurations to DHCP service and prepare some binary and configuration files.

- **DHCP:** We have to provide some information needed by PXE in our dpcp configuration. As an example we add an group definition, which might look like this.

```
group
    # PXE-specific configuration directives...
        next-server 192.168.23.3;
        filename "/tftpboot/pxelinux.0";
```

  This way the DHCP server will tell the target client to search for the TFTP service at the host machine with the IP 192.168.23.3 to fetch the pxe binary we copied to /tftpboot/pxelinux.0.
- **PXE binary** The pxelinux.0 binary is provided by the syslinux package, which is available in most linux distributions, as for our Debina Lenny, the file can be retrieved from `/usr/lib/syslinux/pxelinux.0` after installing the package `syslinux-common`.
- **PXE configuration:** After copying the pxelinux binary. The directory `/tftpboot/pxelinux.cfg` must be created on the tftp server to hold the PXE configuration files. The configuration files can be created predevice. e.g. `/tftpboot/pxelinux.cfg/01-00-50-c2-6e-2e-2f` will be read if a PXE client with the MAC address 01:00:50:C2:6E:2E:2F ist detected. The content of this file might look like this:

```
DEFAULT linux
LABEL linux
```

```
SAY Now booting bzImage-phyCORE-Atom from PXELINUX...
KERNEL bzImage-phyCORE-Atom
APPEND root=/dev/nfs console=ttyS0,115200
ip=192.168.42.61:192.168.23.2:192.168.23.1:255.255.0.0::eth0:
nfsroot=/home/user/OSELAS.BSP-Phytec-phyCORE-12/platform-phyCORE-Z500PT/root
vga=0x317
```

The name of the boot image is defined with `KERNEL`, which means, that we should have copied our /OSELAS.BSP-Phytec-phyCORE-12/platform-phyCORE-Z500PT//images/linuximage to /tftpboot/bzImage-phyCORE-Atom priorly. The kernel command line is passed over with `APPEND`, change this line if any boot settings like e.g. nfsroot shall be changed.

3.3.2 Booting the Embedded Board

After we've got all the stuffs above on the host working, we can now try a netboot by reboot the target, call the BIOS setup screen, set the Network card as first boot devices, save and enter.

There're more options and possibillity setting up the PXE, all of which we certainly are not able to handle here. To learn more please consult sources in the net. http://syslinux.zytor.com/wiki/index.php/PXELINUX is e.g a good start.

4 Accessing Peripherals

4.1 Network

The phyCORE-Z500PT module has aIntel(R) PRO/1000 PCI-Express Gigabit Ethernet ethernet chip onboard, which is being used to provide the `eth0` network interface. The interface offers a standard Linux network port which can be programmed using the BSD socket interface.

4.2 USB Host Controller

The Intel(R) US15W Controller Hub embedds a USB 2.0 EHCI controller that is also able to handle low and full speed devices (USB 1.1).

The OSELAS.BSP-Phytec-phyCORE-12 includes support for mass storage devices and input devices like mouse or keyboards. Other USB related device drivers must be enabled in the kernel configuration on demand.

Due to `udev`, connecting various mass storage devices get unique IDs and can be found in `/dev/disks/by-id`. These IDs can be used in `/etc/fstab` to mount different USB memory devices in a different way.

4.3 Realtime Clock CMOS

The Intel(R) US15W Controller Hub on the phyCORE-Z500PT includes a interal Realtime clock, which is supported by the CMOS driver in kernel. It can be accessed using common time keeping tools in userspace as for any other real time clock.

Date and time can be manipulated with the `hwclock` tool, using the `-w` (systohc) and `-s` (hctosys) options. For more information about this tool refer to the manpage of hwclock.

OSELAS.BSP-Phytec-phyCORE-12 tries to set up the date at system startup. If the system time is changed manually, run `hwclock -w -u` to store the new date into the CMOS.

4.4 Framebuffer

This driver gains access to the display via device node `/dev/fb0`. For this BSP the vesafb display with a resolution of 1024x768 is supported.

A simple test of this feature can be run with:

```
~# fbtest
```

This will show various pictures on the display.

You can check your framebuffer resolution with the command

```
~# fbset
```

NOTE: `fbset` cannot be used to change display resolution or color depth. Depending on the framebuffer device different kernel command line are mostly needed to do this. Please refer to the manual of your display driver for more details.

To change the display resolution, all you have to do is to edit the kernel command line, which is generated and passed over by grub. To do this please open the file `projectroot/boot/grub/menu.lst.phyCORE-Z500PT-1.99.12-2` in your OSELAS.BSP-Phytec-phyCORE-12 with an editor, find the line beginning with the word `kernel`, which might look like this

```
kernel /boot/bzImage root=/dev/sda1 console=ttyS0,115200  ip=@IPADDR@:@SERVERIP@:@GATEWAY@:@NET
```

The value behind `vga=` declares the resolution and color depth of vesafb, as used by phyCORE-Z500PT. Change this value to change the resolution and color depth. Table 4.1 shows some possible values.

Colour depth	640x480	800x600	1024x768	1280x1024
8 (256)	769 (0x301)	771 (0x303)	773 (0x305)	775 (0x307)
15 (32K)	784 (0x310)	787 (0x313)	790 (0x316)	793 (0x319)
16 (65K)	785 (0x311)	788 (0x314)	791 (0x317)	794 (0x31a)
24 (16M)	786 (0x312)	789 (0x315)	792 (0x318)	795 (0x31b)

Table 4.1: *List of some vesa vga mode values*

We can used decimal as well as hexidecimal notation. In our example we use 0x317, which is 1024x768x65K. To find out more usable vga modes please consult the kernel documentation `Documentation/fb/vesafb.txt` in the kernel source code directory. After chaning the the vga value we run

```
~# ptxdist clean grub
~# ptxdist go
~# ptxdist images
```

to generate an image with an update grub config file. Flash this image to your boot medium and restart your board to startup with a new display resolution. Alternatively you can edit the file `/boot/grub/menu.lst` on your target directly. We don't recommend doing this though since the change will get lost the next time you re-generate or flash a image to your target.
If the board is started with remote boot (see section 3.3). Then the vga settings should be changed in the corresponding PXE configuration file of the board. For informations about PXE configuration file please see section 3.3.1.2 on page 20.

5 Special Notes

5.1 Qt Library

The OSELAS.BSP-Phytec-phyCORE-12 ships Qt Library edition qt-embedded of the version 4.5.0. Qt Build configurations can be found in the submenu `"Graphics & Multimedia ---> qt"`. A sample Qt application can be found in /OSELAS.BSP-Phytec-phyCORE-12/local_src/qt4-daemon-trunk. Enable the selection qt4-daemon in the qt configuration menu to build this sample. For a start the sample application code can be used as a base to develop your first Qt application. Pengutronix also provides enterprise support and workshops for GUI development under IDE for embedded devices. Please contact our commercial support if you want to know more. (contact information see section 6.5)

5.2 GTK library

The OSELAS.BSP-Phytec-phyCORE-12 ships GTK Toolkit with GTK+ of the version 2.14.7 and GLib of the version 2.19.10. GTK Build configurations can be found in the submenu `"Graphics & Multimedia ---> gtk+ & friends"`. Demo applications can be found in the furhter submenu `"gtk+"`. PTXdist also provides a set of graphical IDE tools for GTK development with C and C++ as host tools.
Pengutronix also provides enterprise support and workshops for GUI development under IDE for embedded devices. Please contact our commercial support if you want to know more. (contact information see section 6.5)

6 Getting help

Below is a list of locations where you can get help in case of trouble. For questions how to do something special with PTXdist or general questions about Linux in the embedded world, try these.

6.1 Mailing Lists

6.1.1 About PTXdist in Particular

This is an English language public mailing list for questions about PTXdist. See

http://www.pengutronix.de/mailinglists/index_en.html

how to subscribe to this list. If you want to search through the mailing list archive, visit

http://www.mail-archive.com/

and search for the list *ptxdist*. Please note again that this mailing list is just related to the PTXdist as a software. For questions regarding your specific BSP, see the following items.

6.1.2 About Embedded Linux in General

This is a German language public mailing list for general questions about Linux in embedded environments. See

http://www.pengutronix.de/mailinglists/index_de.html

how to subscribe to this list. Note: You can also send mails in English.

6.2 News Groups

6.2.1 About Linux in Embedded Environments

This is an English newsgroup for general questions about Linux in embedded environments.

comp.os.linux.embedded

6.2.2 About General Unix/Linux Questions

This is a German newsgroup for general questions about Unix/Linux programming.

de.comp.os.unix.programming

6.3 Chat/IRC

About PTXdist in particular

irc.freenode.net:6667

Create a connection to the **irc.freenode.net:6667** server and enter the chatroom **#ptxdist**. This is an English room to answer questions about PTXdist. Best time to meet somebody there is at European daytime.

6.4 phyCORE-Z500PT Support Mailing List

OSELAS.Phytec@pengutronix.de

This is an english language public maillist for all BSP related questions specific to Phytec's hardware. See web site

http://www.pengutronix.de/mailinglists/index_en.html

6.5 Commercial Support

You can order immediate support through customer specific mailing lists, by telephone or also on site. Ask our sales representative for a price quotation for your special requirements.

Contact us at:

Pengutronix
Peiner Str. 6-8
31137 Hildesheim
Germany
Phone: +49 - 51 21 / 20 69 17 - 0
Fax: +49 - 51 21 / 20 69 17 - 55 55

or by electronic mail:

sales@pengutronix.de