Quickstart Manual

OSELAS.BSP( )

phyCARDs

	EUROPE	NORTH AMERICA
Address:	PHYTEC Technologie Holding AG Robert-Koch-Str. 39 55129 Mainz GERMANY	PHYTEC America LLC 203 Parfitt Way SW, Suite G100 Bainbridge Island, WA 98110 USA
Ordering Information:	+49 (800) 0749832 order@phytec.de	1 (800) 278-9913 sales@phytec.com
Technical Support:	+49 (6131) 9221-31 support@phytec.de	1 (800) 278-9913 support@phytec.com
Fax:	+49 (6131) 9221-33	1 (206) 780-9135
Web Site:	http://www.phytec.de	http://www.phytec.com

1st Edition: October 2010

Chapter 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 internet.

The central place for our BSPs is our ftp-server ftp://ftp.phytec.de/pub/Products and then navigate to the right product. Then in the directory *Linux* you'll find BSPs and images and in the directory *Software/Kit-CD* you'll find the corresponding whole CD with all: BSP, toolchain, PTXdist, documentations, examples and so on.

In order to build a BSP you need the appropriate toolchain and you need the build tool PTXdist from our partner Pengutronix. The central place for toolchains is http://www.oselas.com and for ptxdist it is http://www.ptxdist.de. This websites provide all required packages and documentation (at least for software components that are available to the public).

To build OSELAS.BSP-Phytec-phyCARD-PD10.2.0, the following archives have to be available on the development host:

- ptxdist-2010.08.0.tgz
- OSELAS.BSP-Phytec-phyCARD-PD10.2.0.tar.gz
- OSELAS.Toolchain-1.99.3.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 must be used to create all images for our embedded devices based on Linux. 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.

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

1.2.2 Extracting the Sources

To install PTXdist you need to extract the archive with the PTXdist software *ptxdist-2010.08.0.tgz*.

The PTXdist packet has 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 step is to extract the archive:

```
~/local# tar -zxf ptxdist-2010.08.0.tgz
```

If everything goes well, we now have a ptxdist-2010.08.0 directory, so we can change into it:

```
~/local# cd ptxdist-2010.08.0
```

1.2.3 Prerequisites

Before PTXdist can be installed it has to be checked if all necessary programs such as *quilt* and *wget* 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-2010.08.0# ./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 2010.08.0 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. Later you will call the current version of PTXdist with command *ptxdist* and all other versions with command *ptxdist-<version>* with *<version>* set to the version you want to use.

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 before re-running the configure script.

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

```
~/local/ptxdist-2010.08.0# 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-2010.08.0# 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-2010.08.0# cd
~# rm -rf 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.

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-cortexa8-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-cortexa8-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.

CAUTION

Our OSELAS.BoardSupport( ) Packages 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/*.

CAUTION

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-phyCARD-PD10.2.0

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 compilers to build the OSELAS.BSP-Phytec-phyCARD-PD10.2.0 board support package are:

<u>phyCARD-S</u>

```
arm-v5te-linux-gnueabi/gcc-4.3.2-glibc-2.8-binutils-2.18-kernel-
2.6.27-sanitized/bin
```

<u>phyCARD-M</u>

```
arm-1136jfs-linux-gnueabi/gcc-4.3.2-glibc-2.8-binutils-2.19-kernel-
2.6.27-sanitized/bin
```

<u>phyCARD-L</u>

```
arm-cortexa8-linux-gnueabi/gcc-4.3.2-glibc-2.8-binutils-2.18-kernel-
2.6.27-sanitized/bin
```

So the steps to build this toolchain for example for the phyCARD-S are:

```
~# tar xf OSELAS.Toolchain-1.99.3.tar.bz2
~# cd OSELAS.Toolchain-1.99.3
~/OSELAS.Toolchain-1.99.3# ptxdist select ptxconfigs/\      [Enter]
>      arm-v5te-linux-gnueabi_gcc-4.3.2_glibc-2.8_binutils-2.18_kernel-
2.6.27-sanitized.ptxconfig
```

```
~/OSELAS.Toolchain-1.99.3# 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.

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 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# ptxdist clean
~/OSELAS.Toolchain-1.99.3# rm selected_ptxconfig
~/OSELAS.Toolchain-1.99.3# ptxdist select \     [Enter]
> ptxconfigs/any_other_toolchain_def.ptxconfig
~/OSELAS.Toolchain-1.99.3# 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.
```

Chapter 2 Building phyCARD's BSP

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-phyCARD-PD10.2.0.tar.gz
~# cd OSELAS.BSP-Phytec-phyCARD-PD10.2.0
```

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

```
~/OSELAS.BSP-Phytec-phyCARD-PD10.2.0# ls -l
total 44
-rw-rw-r-- 1 jb users  598 2010-10-07 14:35 AUTHORS
-rw-r--r-- 1 jb users 4078 2010-10-07 14:35 ChangeLog
-rw-rw-r-- 1 jb users 1313 2010-10-07 14:35 Kconfig
drwxrwxr-x 6 jb users 4096 2010-10-07 14:35 configs
drwxrwxr-x 3 jb users 4096 2010-10-07 14:35 documentation
drwxrwxr-x 7 jb users 4096 2010-10-07 14:35 local_src
drwxrwxr-x 6 jb users 4096 2010-10-07 14:35 patches
drwxrwxr-x 8 jb users 4096 2010-10-07 14:35 projectroot
drwxrwxr-x 3 jb users 4096 2010-10-07 14:35 protocols
drwxrwxr-x 3 jb users 4096 2010-10-07 14:35 rules
drwxrwxr-x 3 jb users 4096 2010-10-07 14:35 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-phyCARD-PD10.2.0* comes with a predefined configuration we select in the following step:

```
~/OSELAS.BSP-Phytec-phyCARD-PD10.2.0# 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 phyCARD-S:

```
~/OSELAS.BSP-Phytec-phyCARD-PD10.2.0# ptxdist platform \     [Enter]
```

```
> configs/phyCARD-S-2010.08.0/platformconfig
info: selected platformconfig:
'configs/phyCARD-S-2010.08.0/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/arm-v5te-linux-gnueabi/
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-phyCARD-PD10.2.0# ptxdist toolchain \     [Enter]
> /opt/OSELAS.Toolchain-2010.08.0/arm-v5te-linux-gnueabi/\     [Enter]
gcc-4.3.2-glibc-2.8-binutils-2.18-kernel-2.6.27-sanitized/bin
```

2.5 Building the Linux Kernel and its Root Filesystem

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

```
~/OSELAS.BSP-Phytec-phyCARD-PD10.2.0# 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 (the linux kernel and those that actually put 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 linux kernel can be found in *platform-phyCARD-S/images/* directory and the final root filesystem for the target board can be found in the *platform-phyCARD-S/root/* directory and a bunch of **.ipk* packets in the *platform-phyCARD-S/packages/* directory, containing the single applications the root filesystem consists of.

2.6 Building an Root Filesystem Image

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

```
~/OSELAS.BSP-Phytec-phyCARD-PD10.2.0# 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 several image types. What you need is:

- root.jffs2: root files inside a jffs2 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-phyCARD-PD10.2.0# ptxdist platformconfig
```

Then select the submenu *image creation options*. The generated image will be placed into *platform-phyCARD-S/images/*.

CAUTION

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-phyCARD-S/root/* will not be enclosed in the image. If custom files are needed for the target. Install it with ptxdist.

Chapter 3 phyCARD preparation

This step can be omitted if the boot loader is recent enough: For this BSP at least the barebox-2010.10.0 is required.

3.1 Updating Barebox

Build the whole BSP with *ptxdist go* or just build the barebox with *ptxdist targetinstall barebox*. When it’s built copy the generated file *platform-phyCARD-S/images/barebox-image* to your configured tftp exported directory.

On the target side first check for the correct network settings:

```
barebox:/ edit /env/config
```

With your development host set to IP *192.168.3.10* and netmask *255.255.255.0*, the target should contain the lines

```
eth0.ipaddr=192.168.3.11

eth0.netmask=255.255.255.0

eth0.serverip=192.168.3.10
```

and dhcp must be disabled, that means commented out:

```
#ip=dhcp
```

If you need to change something, save your settings by leaving the editor with Strg-D, then type *save* and reboot the board. Otherwise leave the editor with Strg-C.

Now enter:

```
barebox:/ tftp barebox-image
```

to load this binary image into the RAM disk of your target.

To replace the current Barebox in your target start with the following commands:

```
barebox:/ erase /dev/self0
barebox:/ cp barebox-image /dev/self0
```

The new Barebox revision is now stored in the NAND flash.

CAUTION Note that if something goes wrong at this, you don't have any bootloader anymore on your module. In this case you need to install a new one as being described in the Quickstart Manual.

Chapter 4 Booting Linux

Now that there is a linux kernel and a root filesystem in our workspace we'll have to make them visible to the phyCARD. There are two possibilities to do this:

1. Making the linux kernel image and the root filesystem image persistent in the onboard media.

2. Booting from the development host via network.

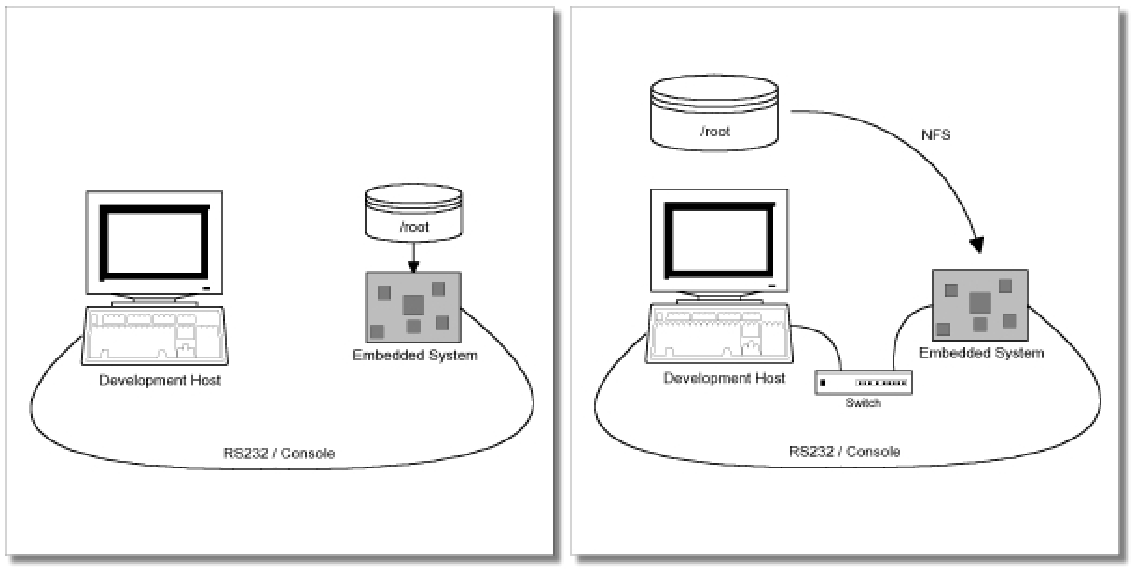


Figure 4.1: Booting the phyCARD: From its flash or from the host via network.

Figure 4.1 shows both methods. The main method used in the OSELAS.BSP-Phytec-phyCARD-PD10.2.0 BSP is to provide all needed components to run on the target itself. The linux kernel image and the root filesystem image are 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 needed components via network. In this case the development host is connected to the phyCARD 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-phyCARD-S/root/* directory in our PTXdist workspace.

The OSELAS.BSP-Phytec-phyCARD-PD10.2.0 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.

4.1 Target Side Preparation

The phyCARDs use Barebox as their bootloader. Barebox can be customized with environment variables and scripts to support any boot constellation. OSELAS.BSP-Phytec-phyCARD-PD10.2.0 comes with a predefined environment setup to easily bring up the phyCARDs.

Usually the environment doesn't have to be set manually on our target. PTXdist comes with an automated setup procedure to achieve a correct environment on the target.

Due to the fact that some of the values of these Barebox environment variables must meet our local network environment and development host settings we have to define them prior to running the automated setup procedure.

Note: At this point of time it makes sense to check if the serial connection is already working, because it is essential for any further step we will do.

We can try to connect to the target with our favorite terminal application (minicom or kermit for example). With a powered target we identify the correct physical serial port and ensure that the communication is working.

Make sure to leave this terminal application to unlock the serial port prior to the next steps.

To set up development host and target specific value settings, we run the command

```
~/OSELAS.BSP-Phytec-phyCARD-PD10.2.0# ptxdist boardsetup
```

We navigate to *Network Configuration* and replace the default settings with our local network settings. In the next step we also should check if the *Host's Serial Configuration* entries meet our local development host settings. Especially the *serial port* must correspond to our real physical connection.

When everything is set up, we can *Exit* the dialog and and save our new settings.

Now the command

```
~/OSELAS.BSP-Phytec-phyCARD-PD10.2.0# ptxdist test setenv
```

will automatically set up a correct default environment on our phyCARD. We have to powercycle our target to make this step happen.

It should output lines like these when it was successful:

```
===============================
Please power on your board now!
===============================
Logging into U-Boot.....................OK
Setting new environment.................OK
Test finished successfully.
```

Note: If it fails, reading *test.log* in the corresponding *platform-phyCARD* directory will give further information about why it has failed. Also extending the command line shown above by a --*debug* can help to see whats going wrong.

CAUTION

Users reported this step could fail if the Linux system running PTXdist is a virtual machine as guest in an operating system from Redmont. In this case it seems at least one of the two OSes is eating up characters sent to the serial line. Pengutronix recommends running PTXdist on a real Linux system.

4.2 Stand-Alone Booting Linux

To use the the target standalone, the rootfs has to be made persistent in the onboard media of the phyCARD. The following sections describe the steps necessary to bring the rootfs into the onboard NAND type flash.

Only for preparation we need a network connection to the embedded board and a network aware bootloader which can fetch any data from a TFTP server.

After preparation is done, the phyCARD can work independently from the development host. We can ”cut” the network (and serial cable) and the phyCARD will continue to work.

4.2.1 Development Host Preparations

On the development host a TFTP server has to be installed and configured. The exact method to do so 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.

Usually TFTP servers are using the */tftpboot* directory to fetch files from, so if we want to push kernel images into 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. The boardsetup scripts coming with this BSP expect write permission in TFTP directory!

We can run a simple:

```
~# touch /tftpboot/my_file
```

to test if we have permissions to create files in this directory. If it fails we have to ask the administrator to grant these permissions.

Note: We must */tftpboot* part of the command above with our local settings.

4.2.2 Preparations on the Embedded Board

To boot a phyCARD stand-alone, anything needed to run a Linux system must be locally accessible. So at this point of time we must replace any current content in phyCARD’s flash memory.

But first we must create the new root filesystem image prepared for its usage on the phyCARD:

```
~/OSELAS.BSP-Phytec-phyCARD-PD10.2.0# ptxdist images
```

To simplify this step, OSELAS.BSP-Phytec-phyCARD-PD10.2.0 comes with an automated setup procedure for this step. To use this procedure we run the command:

```
~/OSELAS.BSP-Phytec-phyCARD-PD10.2.0# ptxdist test flash
```

Note: This command requires a serial and a network connection. The network connection can be cut after this step.

This command will automatically write the linux kernel and the root filesystem to the correct flash partition on the phyCARD. It only works if we previously have set up the environment variables successfully (described on previous pages of this manual). The command should output lines like this when it was successful:

```
===============================
Please power on your board now!
===============================
Logging into U-Boot…....................OK
Flashing kernel.........................OK
Flashing rootfs.........................OK
Flashing oftree.........................OK
Test finished successfully.
```

Note: If it fails, reading *test.log* in the corresponding *platform-phyCARD* directory will give further information about why it has failed.

4.2.3 Booting the Embedded Board

To check that everything went successfully up to here, we can run the boot test.

```
~/OSELAS.BSP-Phytec-phyCARD-PD10.2.0# ptxdist test boot
===============================
Please power on your board now!
===============================
Checking for U-Boot.....................OK
Checking for Kernel.....................OK
Checking for init.......................OK
Checking for login......................OK
Test finished successfully.
```

This will check if the environment settings and flash partitioning are working as expected, so the target comes up in stand-alone mode up to the login prompt.

Note: If it fails, reading *test.log* in the corresponding *platform-phyCARD* directory will give further information about why it has failed.

After the next reset or powercycle of the board, it should boot the kernel from the flash, start it and mount the root filesystem also from flash.

Note: The default login account is root with an empty password.

4.3 Remote-Booting Linux

The next method we want to try after building the linux kernel and the 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 *root/* directory of the corresponding *platform-phyCARD* directory simply ”appears” on the embedded device immediately.

All we need is a network interface on the embedded board and a network aware bootloader which can fetch the kernel from a TFTP server.

4.3.1 Development Host Preparations

If we already have booted the phyCARD locally (as described in the previous section), all of the development host preparations are done.

If not, then a TFTP server has to be installed and configured on the development host. The exact method of doing this 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.

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-phyCARD-PD10.2.0#          sudo          cp
<platform>/images/linuximage /tftpboot/uImage
```

Note: Replace `<platform>` with the corresponding *platform-phyCARD* directory.

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 192.168.3.10, so the IP addresses have to be adapted to the local needs:

```
/home/<user>/work 192.168.3.10/255.255.255.0(rw,no_root_squash,sync)
```

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

4.3.2 Preparations on the Embedded Board

We already provided the phyCARD with the default environment at previous pages. So there is no additional preparation required here.

4.3.3 Booting the Embedded Board

The default environment settings coming with the OSELAS.BSP-Phytec-phyCARD-PD10.2.0 has the possibility to boot from the internal flash or from the network. Configuration happens in the file */env/config*. As Barebox uses a full shell like console you can edit this file to configure the other scripts (the boot script for example).

To edit this configuration file we run the *edit* command on it:

```
uboot:/ edit /env/config
```

We move to the lines that define the *kernel_loc* and *rootfs_loc* variables. They can be defined to *nand* or *net* (option *nor* is not available on phyCARDs, because phyCARDs don’t have any NOR flash memory). *nand* lets the boot script load everything from the NAND flash memory.

In this example we change it to *net* to load all parts from the network. When we do that, we also have to configure the network setup a few lines above in this file. We setup these values to the network we want to run the phyCARD.

Leaving this editor with saving the changes happens with CTRL-D. Leaving it without saving the changes happens with CTRL-C.

Note: Saving here means the changes will be saved to the RAM disks Barebox uses for the environment. To store it to the persistent memory, an additional *save* command is required.

Now its time to boot the phyCARD. To do so, simply run:

```
barebox:/ boot
```

This command should boot the phyCARD into the login prompt.

Note: The default login account is *root* with an empty password.

Chapter 5 Accessing Peripherals

The following sections provide an overview of the supported hardware components and their corresponding operating system drivers. Further changes can be ported on demand of the customer.

Phytec's phyCARD starter kit consists of the following individual boards:

1. The phyCARD module itself, containing the controller (PCA-A-S1: i.MX27, PCA-A-M1: i.MX35, PCA-A-L1: Cortex-A8), RAM, flash and several other peripherals.

2. The starter kit baseboard (PBA-A-01).

To achieve maximum software re-use, the Linux kernel offers a sophisticated infrastructure, layering software components into board specific parts. The OSELAS.BSP( ) tries to modularize the kit features as far as possible; that means that when a customized baseboard or even customer specific module is developed, most of the software support can be re-used without error prone copy-and-paste. So the kernel code corresponding to the boards above can be found in

PCA-A-S1: *arch/arm/mach-mx2/pca100.c*

PCA-M-S1: *arch/arm/mach-mx3/pca101.c*

PCA-L-S1: *arch/arm/mach-omap2/board-pca102.c*

In fact, software re-use is one of the most important features of the Linux kernel and especially of the ARM port, which always had to fight with an insane number of possibilities of the System-on-Chip CPUs.

CAUTION Note that the huge variety of possibilities offered by the phyCARD modules makes it difficult to have a completely generic implementation on the operating system side. Nevertheless, the OSELAS.BSP( ) can easily be adapted to customer specific variants. In case of interest, contact our sales department (sales@phytec.de) and ask for a dedicated offer.

The following sections provide an overview of the supported hardware components and their operating system drivers.

5.1 NAND Flash

The phyCARD modules come with NAND memory to be used as media for storing linux and its root filesystem, including applications and their data files. This type of media will be managed by the JFFS2 filesystem. This filesystem uses compression and decompression on the fly, so there is a chance to bring more data into this device.

NOTE: JFFS2 has a disadvantage: We cannot use the mmap()-API to manipulate data on a mapped file.

From linux userspace the NAND flash partitions can be seen as

• */dev/mtdblock0* (Barebox partition)

• */dev/mtdblock1* (Barebox environment partition)

• */dev/mtdblock2* (Kernel partition)

• */dev/mtdblock3* (Linux rootfs partition)

5.2 Serial TTYs

The phyCARDs support one UART unit, that is routed to the connector and can be used in user’s application: *ttyS0*.

5.3 Network

The phyCARD modules feature ethernet, 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.

5.4 Camera Interface

The PBA-A1-01 comes with a port for phyCAM-S cameras. The current BSP supports this only for the phyCARD-S. The drivers are compatible to Video4Linux2 (v4l2) and thus they work for example with gstreamer. The PHYTEC camera series VM-006, VM-007 and VM-009 are supported.

Detailed information about the phyCAM - interface and the PHYTEC camera boards you will find in the "phyCAM-P-S" manual L-748.

To use the camera feature the OSELAS.BSP-Phytec-phyCARD-PD10.2.0 includes scripts that activate them. Their usage depends on the connected camera. Within file */env/config* of barebox’s environment you can declare settings for the camera drivers you need. As default you will find the entry *mt9v022.sensor_type=colour* that configures the driver for a PHYTEC VM-007 camera to use colour. Without this entry the driver would be set to b/w. Furthermore the b/w camera driver *mt9m001* for a PHYTEC VM-006 camera will be loaded as default without having entries to be declared here.

5.4.1 Using the Camera Support

Before we start with details, we should know that along with our OSELAS.BSP-Phytec-phyCARD-PD10.2.0 BSP comes a set of preconfigured scripts which are capable to do some basic guessing what kind of hard/software we have and start the camera capturing with proper parameters.

Neverless one should go through the following sections to understand how the stuff exactly works.

We can find the convenience scripts on our target under */home/gstreamer-examples*.

```
~# ls gstreamer-examples
bwcam-fbdev_240x320          colcam-mpeg4
bwcam-fbdev_240x320-roi      colcam-save_jpg_full_res
bwcam-fbdev_640x480          colcam-save_raw_full_res
bwcam-mpeg4                  colcam-xv
bwcam-save_jpg_full_res      func.sh
bwcam-save_raw_full_res      more_mt9m131_scripts
bwcam-xv                     register-settings-mt9m001.txt
colcam-fbdev_240x320         register-settings-mt9m131.txt
colcam-fbdev_240x320-roi     register-settings-mt9v022.txt
colcam-fbdev_640x480
```

The names of the scripts stand for what they can be used. If we have a colour camera PHYTEC VM-007 for example and the standard 640x480 display on our board, we can start capturing by simply doing:

```
~# cd gstreamer-examples
~# ./colcam-fbdev_640x480
```

We will get the output:

```
mx27-camera mx27-camera.0: MX27 Camera driver unloaded
mx27-camera mx27-camera.0: initialising
mx27-camera mx27-camera.0: Camera clock frequency: 53200000
mx27-camera mx27-camera.0: Camera clock frequency: 53200000
mx27-camera mx27-camera.0: Using EMMA
camera 0-0: Probing 0-0
camera 0-0: Camera driver attached to camera 0
mx27-camera mx27-camera.0: mclk_get_divisor not implemented. Running at max speed
mt9m001 0-005d: No MT9M001 chip detected, register read fffffffb
mx27-camera mx27-camera.0: Camera driver detached from camera 0
camera: probe of 0-0 failed with error -12
camera 0-1: Probing 0-1
camera 0-1: Camera driver attached to camera 1
mx27-camera mx27-camera.0: mclk_get_divisor not implemented. Running at max speed
mt9v022 0-0048: Detected a MT9V022 chip ID 1313, colour sensor
mx27-camera mx27-camera.0: Camera driver detached from camera 1
Camera mt9v022 attached.
starting gstreamer ...
Setting pipeline to PAUSED ...
camera 0-1: Camera driver attached to camera 1
mx27-camera mx27-camera.0: mclk_get_divisor not implemented. Running at max speed
i2c open /dev/video0
Pipeline is live and does not need PREROLL ...
Setting pipeline to PLAYING ...
New clock: GstSystemClock
```

The occuring error messages can be ignored: The script tries to probe for a suitable camera module. We will only see them if we access the board through a serial terminal. If everything goes well, there won't be any further messages on the console and after a while the display should start showing camera's picture. We can stop capturing and displaying by pressing the well known Strg-C.

CAUTION

When trying to capture for the first time, it might happen that the display will only show some coloured snow. In that case just stop the script and start it again.

Some of the scripts the OSELAS.BSP-Phytec-phyCARD-PD10.2.0 BSP provides and their intentional usage:

- *bwcam-fbdev_640x480* for b/w camera + 640x480 display
- *colcam-fbdev_240x320* for colour camera + 240x320 display
- *colcam-fbdev_640x480* for colour camera + 640x480 display
- *bwcam-xv* for b/w camera + X enabled display

• *colcam-xv* for colour camera + X enabled display

Furthermore there are some subdirectories within the example directory. They provide more GStreamer scripts for dedicated sensors/cameras.

5.4.2 Turn on Camera Support in System

PHYTEC ships different kind of cameras with their development kits. Depending on the kit we might need different drivers for these cameras. Selecting the right one must be done in */env/config* of berebox's environment. The following camera drivers are available:

mt9v022 This driver supports Micron MT9V022 picture sensors. If we have a camera of the VM-007 series, we will most probably need this driver. This driver is not capable to detect if the camera comes with a colour or monochrome sensor, so you must set the boot parameter *mt9v022.sensor_type=colour* within */env/config* of barebox's environment for a colour camera and you must remove it for a b/w camera.

mt9m001 This driver supports Micron MT9M001 b/w picture sensors, as built in cameras of the VM-006 series.

mt9m111 This driver supports Micron MT9M131 colour picture sensors, as built in cameras of the VM-009 series. It uses the same I²C address as the mt9v022 driver, so these both cannot be loaded at the same time. If you need this driver, you have to declare *use_mt9m111=1* as boot parameter within */env/config* of barebox's environment, what will disable the mt9v022 driver at the same time.

When loaded the drivers try to detect a camera, which they support. If the check is successful, the driver will register the camera device to the video subsystem and create a device node name */dev/video<x>* in the file system, where the *<x>* stands for the numbering of the device. If we have one camera only, which is mostly the case, it should be */dev/video0*. If our system misses this file, we have most probably loaded the wrong driver or our camera is not attached properly.

5.5 Basic Usage of GStreamer

GStreamer is a streaming media framework, based on graphs of filters which operate on media data. GStreamer consists of several command line tools and sets of plugins. Generally there are three kind of plugins:

Source Plugins These plugins access directly media sources on the system and pass the media data on to further plugins. For example, we can use the v4l2src plugin to access a camera.

Pipe Plugins These plugins are like double ended tubes. We can put data in one side and take the processed data from the other side.

Sink Plugins These plugins take the media data and put them to ”sinks”, which are output devices like framebuffer device, x.org client or even Network sockets.

Using GStreamer on command line is like building a pipeline. To launch the pipeline, we need the command-line tool *gst-launch*.

5.5.1 Simple Usage Example

To get a feeling how this works we can use the built-in fake plugins to build a pipeline, which simply process some empty buffers:

```
~# gst-launch -v fakesrc num-buffers=1 ! fakesink
```

This will result to output that looks simlar to this:

```
Setting pipeline to PAUSED ...
Pipeline is PREROLLING ...
/GstPipeline:pipeline0/GstFakeSrc:fakesrc0: last-message = "get       ******* > (     0
bytes, timestamp: none, duration: none, offset: 0, offset_end: -1, flags: 0) 0x6e280"
/GstPipeline:pipeline0/GstFakeSink:fakesink0: last-message = "preroll   ******* "
/GstPipeline:pipeline0/GstFakeSink:fakesink0: last-message = "event    ******* E (type:
102, GstEventNewsegment, update=(boolean)false, rate=(double)1, applied-rate=(double)1,
format=(GstFormat)GST_FORMAT_BYTES,          start=(gint64)0,          stop=(gint64)-1,
position=(gint64)0;) 0x6a950"
Pipeline is PREROLLED ...
Setting pipeline to PLAYING ...
/GstPipeline:pipeline0/GstFakeSrc:fakesrc0: last-message = "event    ******* E (type:
289, GstEventLatency, latency=(guint64)0;) 0x6ab80"
/GstPipeline:pipeline0/GstFakeSink:fakesink0: last-message = "chain    ******* < (     0
bytes, timestamp: 0:00:00.000000000, duration: none, offset: 0, offset_end: -1, flags:
32) 0x6e280"
New clock: GstSystemClock
/GstPipeline:pipeline0/GstFakeSink:fakesink0: last-message = "event    ******* E (type:
86, ) 0x6a950"
Got EOS from element "pipeline0".
Execution ended after 4900932 ns.
Setting pipeline to PAUSED ...
Setting pipeline to READY ...
Setting pipeline to NULL ...
Freeing pipeline ...
```

All plugins are connected with an exclamation mark (!)

If this works properly, we can go on to do some real work in the next section.

5.5.2 Simple Monochrome Usage Example

The following line will grab the video stream from a monochrome camera like the VM-006 and put it on the framebuffer device:

```
~# gst-launch v4l2src ! video/x-raw-gray,width=1280,height=1024 ! ffmpegcolorspace ! fbdevsink
```

Three plugins are used here:

1. *v4l2src* plugin grabs the raw frames from the camera using the Video4Linux2 API

2. *video/x-raw-gray,width=1280,height=1024* in the pipeline is a capability filter. It sets a mime type to specify a desired video format (in this case grayscale) and the frame size (in this case 1280x1024).

3. *ffmpegcolorspace* takes the stream and converts it to suitable colourspace

4. *fbdevsink* takes the converted stream and displays it on a framebuffer device

5.5.3 Simple Colour Usage Example

If we have a camera with a colour sensor like the VM-009 instead, we can use the following line to process the videostream:

```
~# gst-launch v4l2src ! video/x-raw-rgb,width=1280,height=1024 ! ffmpegcolorspace ! fbdevsink
```

If you use a camera that delivers bayer pattern instead of RGB values like the PHYTEC VM-007-COL, there is a plugin named *bayer2rgb* to convert the raw content provided by the camera sensor into an RGB signal:

```
~# gst-launch v4l2src ! video/x-raw-bayer,width=752,height=480 ! bayer2rgb bg_first=false ! ffmpegcolorspace ! fbdevsink
```

CAUTION

The bayer2rgb plugin uses a parameter bg_first to define if the first line of the bayer pattern provided by the sensor is a blue/green line or a green/red line. We can try to change this to true if we experience any troubles with our colourspace.

5.6 Advanced Usage of GStreamer

5.6.1 Manually Setting the Frame Rate and Framesize

Along with the mimetype definition we can also optionally set information like frame size and rate. e.g.

```
~# gst-launch v4l2src \      [Enter]
> ! video/x-raw-gray,width=640,height=480,framerate=30/1 \      [Enter]
> ! ffmpegcolorspace ! fbdevsink
```

Doing this will set a fix value for the input frame format. Such options may come handy if we e.g. have trouble with the size of the input stream. Note:

• If we adjust the width and height ourself, the shown region starts at the upperlefter corner of the frame captured by the camera.

• The frame size we can choose depends on the one our camera can provide. If we choose any size which extends the picture range of the camera, GStreamer will fail to start with an output like this:

```
ERROR: from element /pipeline0/v4l2src0: Could not negotiate format
```

CAUTION

Some cameras do not provide their framesize properly to the system or just simply provide a frame bigger than the system can process. In this case the command will fail and we have to define the framesize manually.

5.6.2 Manipulate Input Frame Size with Plugins

Normally the onboard framebuffer device has smaller size than the picture captured by the camera. Because of that, only a part of the captured video might be visible, starting at the upleft corner, on the display. To get the picture we actually want on our display, we can use various plugins:

• we can crop the videosignal using the *videocrop* plugin

```
~# gst-launch v4l2src ! video/x-raw-gray \     [Enter]
> ! videocrop left=250 right=250 top=80 bottom=80 \     [Enter]
> ! ffmpegcolorspace ! fbdevsink
```

Will "chop out" 432 pixel in the horizontal and 240 pixel in the vertical direction of the input frame, so that we will get a 320x240 sized frame which is positioned in the central of the input frame. This command line will work with a PHYTEC VM-007 (MT9V022) camera, which provides a 752(H)x480(V) sized frame at default. We should consult the datasheet of our display and our camera to find out the correct crop parameter for the camera kit.

• If we'd rather prefer not to crop out regions of the input frame, we can use the *videoscale* plugin to resize the input frame

```
~# gst-launch v4l2src ! video/x-raw-gray ! ffmpegcolorspace !
videoscale ! video/x-raw-yuv,width=320,height=240 ! ffmpegcolorspace !
fbdevsink
```

With this command we can resize the video frame to 320x240. Notice:

– Unlike the *videocrop* plugin, *videoscale* cannot process raw data provided by *v4l2src* directly. So we put a *ffmpegcolorspace* between the *v4l2src* and *videoscale*.

– Aspect ration in the resized frame should be proportional to the original size. Otherwise we might get a disorted image.

5.6.3 Manipulate Picture's Orientation

Further we can flip and rotate our video with the *videoflip* plugin. We take the command above as example:

```
~# gst-launch v4l2src ! video/x-raw-gray \     [Enter]
> ! ffmpegcolorspace \     [Enter]
> ! videoscale ! video/x-raw-yuv,width=320,height=240 \     [Enter]
> ! videoflip method=clockwise \     [Enter]
> ! ffmpegcolorspace ! fbdevsink
```

The addtional videoflip plugin in the pipeline flips the video 90 degrees clockwise.

5.6.4 Using other Sinks Than the Framebuffer

Beside the local framebuffer there are additional locations where to show the video stream.

ximagesink and *xvimagesink*

If the support for X protocols is turned on in our OSELAS.BSP-Phytec-phyCARD-PD10.2.0, we can use it to display our camera stream on a remote host, which runs a common X server like X.Org. To do this we run the following steps:

On our host:

```
user@host ~ xhost +
access control disabled, clients can connect from any host
```

CAUTION

Note: With this command we will grant access from any X-Clients in our network to our local X-Server. This could be a security issue. Hence we might probably want to consult our system administrator first before doing this.

On target:

```
~# export DISPLAY=[IP of our host in here]:0
```

Now any X-related application started on our target will be shown on our host and we can start our Gstreamer with the *ximagesink*

```
~# gst-launch v4l2src ! video/x-raw-gray ! ffmpegcolorspace ! ximagesink
```

Notice that displaying our videostream like this we might get very low framerate due to high network load.

udpsink

With GStreamer we can also stream our video with various streaming protocols through the network. However we have to encode the stream with a video codec first. Currently only hardware encoding is supported. We only can use this feature if our phyCARD provides a video processing unit.

If our target system supports such a feature, we can start streaming on the target with following command:

```
~# gst-launch v4l2src ! video/x-raw-gray ! ffmpegcolorspace ! mfw_vpuencoder \codec-type=std_avc bitrate=32767 gopsize=10 ! rtph264pay ! udpsink host=[HOST IP ADDRESS] port=5555
```

On the host we can use the following command to decode this video stream:

```
user@host ~ gst-launch udpsrc port=5555 caps = "application/x-rtp, media=(string)video" ! rtph264depay ! ffdec_h264 ! xvimagesink
```

5.7 SPI Master

phyCARDs support an SPI bus. Connected device can be found in the sysfs at the path */sys/bus/spi/devices*. It depends on the corresponding SPI slave device driver if it provides access to the SPI slave device through this way (sysfs), or any different kind of API.

We plan to offer SPI-driven expansion boards to the phyBASE, for example to provide a CAN port.

5.8 Framebuffer

This driver gains access to the display via device node */dev/fb0*. For this BSP the Primeview PD050VL1 display with a resolution of 640x480 is default.

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 colour 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.

The BSP also supports the Primeview displays PD035VL1 (640x480), PD104SLF (800x600) and PM070WL4 (800x480). Selection will be done in the barebox in script */env/config*. There you will find the line

```
bootargs="console=ttymxc0,115200"
```

Here you can add more boot parameters, for example:

```
bootargs="console=ttymxc0,115200 video=imxfb:Primeview-PD104SLF"
```

5.9 Touch

A simple test of this feature can be run with

```
~# ts_calibrate
```

to calibrate the touch and with

```
~# ts_test
```

to do a simple application using this feature.

5.10 I²C Master

phyCARDs support one I²C on the module and one on the baseboard. The kernel supports the controller as a master controller.

Additional I²C device drivers can use the standard I²C device API to gain access to their devices through this master controller. For further information about the I²C framework see *Documentation/i2c* in the kernel source tree.

5.10.1 I²C Realtime Clock RTC8564

Due to the Real Time Clock framework of the kernel the RTC8564 clock chip on the phyBASE can be accessed using the same tools 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-phyCARD-PD10.2.0 tries to set up the date at system startup. If there was a powerfail *hwclock* will state (example for phyCARD-S, where the I²C address of the RTC is 1-0051):

```
pcf8564 1-0051: low voltage detected, date/time is not reliable.
pcf8564 1-0051: retrieved date/time is not valid.
```

In this case set the date manually (see *man date*) and run *hwclock -w -u* to store the new date into the RTC8564.

5.10.2 I²C Device 24WC32W

This device is a 4 kiB non-volatile memory for general purpose usage on phyCARDs.

This type of memory is accessible through the sysfs filesystem. To read the EEPROM content simply *open()* the entry */sys/bus/i2c/devices/<I²C address>/eeprom* (set I²C address to *1-0052* in case of phyCARD-S) and use *fseek()* and *read()* to get the values.

5.11 USB Host Controller

phyCARDs embed a USB 2.0 EHCI controller that is also able to handle low and full speed devices (USB 1.1).

The OSELAS.BSP-Phytec-phyCARD-PD10.2.0 includes support for mass storage devices and 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.

5.12 OneWire Interface

Currently only available for phyCARD-S and phyCARD-M.

Any detected 1W device will be mapped to the *sysfs* filesystem. For example a connected temperature sensor could be accessed via this entry:

```
/sys/bus/w1/devices/10-000801018ed7/w1_slave
```

A simple *cat* command can give you the follwing output:

```
root@phyCARD:~ cat /sys/bus/w1/devices/10-000801018ed7/w1_slave
2e 00 4b 46 ff ff 0e 10 91 : crc=91 YES
2e 00 4b 46 ff ff 0e 10 91 t=22875
```

The baseboard and the display board each contain an eeprom DS28EC20 20kbit. Such an eeprom can be accessed via an entry like that:

```
/sys/bus/w1/devices/43-0000000f2510/eeprom
```

Testing can be done with lines like that:

```
echo "hello world" > /sys/bus/w1/devices/43-0000000f2510/eeprom
cat /sys/bus/w1/devices/43-0000000f2510/eeprom
```

5.13 MMC/SD Card

phyCARDs support Secure Digital Cards and Multi Media Cards to be used as general purpose blockdevices. These devices can be used in the same way as any other blockdevice.

CAUTION These kind of devices are hot pluggable, so you must pay attention not to unplugg the device while its still mounted. This may result in data loss.

After inserting an MMC/SD card, the kernel will generate new device nodes in *dev/*. The full device can be reached via its */dev/mmcblk0* device node, MMC/SD card partitions will occure in the following way:

```
/dev/mmcblk0p<Y>
```

<Y> counts as the partition number starting from 1 to the max count of partitions on this device.

Note: These partition device nodes will only occure if the card contains a valid partition table (”harddisk” like handling). If it does not contain one, the whole device can be used for a filesystem (”floppy” like handling). In this case */dev/mmcblk0* must be used for formatting and mounting.

The partitions can be formatted with any kind of filesystem and also handled in a standard manner, e.g. the *mount* and *umount* command work as expected.

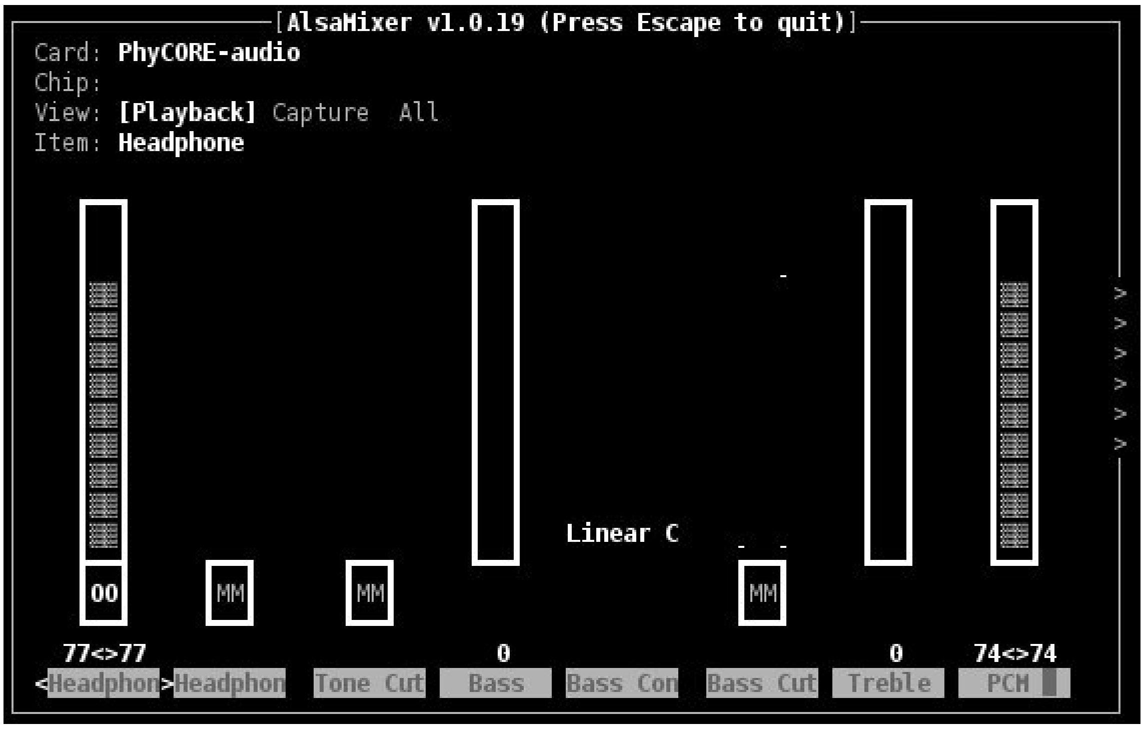
5.14 AUDIO support

Audio support on the module is done via the AC97 interface.

5.14.1 Audio Sources and Sinks

The phyBASE has three jacks for Line Out, Line In and the Handset microphone can be connected through three jacks.

Enabling and disabling input and output channels can be done with the *alsamixer* program. Select the screen *Playback* for this view (using F3/F4 or the tabulator key).



```
[AlsaMixer v1.0.19 (Press Escape to quit)]
Card: PhyCORE-audio
Chip:
View: [Playback] Capture  All
Item: Headphone
```

Figure 5.1: Playback controls

With the keys cursor left and cursor right you can step through the different channels. There are much more channels than fit onto one screen, so the screen will scroll if your cursor reaches the right or left edge of it.

Please navigate as far to the right until you reach channel *Left Cap*. There are always max. 8 letters being displayed in the bottom line, but for the currently selected channel its full name will be displayed in the line *Item*: *Left Capture Select*. With the keys cursor up and cursor down you can choose another source than the default *Mic*. Please set it to *Line*. After that, please do the same with channel *Right Ca* (*Right Capture Select*).

Now please select the second of the *Left HP* channels, its full name is *Left HP Mixer Line Bypass*. With key m you can switch this *on* and *off*. Please switch it to *on* in order to get sound directly directed from Line In to Line Out. Please do the same with the second of the *Right HP* channels, that means with *Right HP Mixer Line Bypass*. Now sound you put into Line In should be available at Line Out.

alsamixer can be left by pressing the *ESC* key. If you want these settings to be persistent you can run a *alsactl store* now. This will store the current audio mixer settings into the file */etc/asound.state*. At the next system start these settings will be restored by the */etc/init.d/alsa-utils* script.

5.14.2 Playback

This BSP comes with two command line tools to playback various audio stream files.

To playback MP3 based streams, we can use the *madplay* tool.

```
~# madplay <your-favorite-song-file>
```

To playback simple audio streams, we can use *aplay* instead.

```
~# aplay /usr/share/sounds/KDE_Startup.wav
```

5.14.3 Capture

arecord is a command line tool for capturing audio streams. Default input source is *Mic*. We can use the *alsamixer* in order to select a different audio source to capture.

The following example will capture the current stereo input source with 16kHz sample rate and will create an audio file in *WAV* format (signed 16 bit per channel, 32 bit per sample):

```
~# arecord -t wav -c 2 -r 16000 -f S16_LE /demo.wav
```

The following example will capture the current stereo input source with 8kHz sample rate and will create an audio file in *WAV* format (signed 16 bit per channel, 32 bit per sample):

```
~# arecord -t wav -c 2 -r 8000 -f S16_LE /demo.wav
```

Note: Sample rate is restricted by the external PMIC device. Only 8kHz and 16kHz are possible.

Capturing can be stopped again using Strg-C.

Chapter 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 phyCARD Support

support@phytec.de

Ask your questions in english or german to Phytec's Support or visit our FAQs in the web. Call

http://www.phytec.eu (english) or http://www.phytec.de (german)

and then navigate to *Support / FAQ / Modules / phyCARD-…*

6.5 Commercial Support

You can order immediate support or direct contact to the developers, by telephone or mail. Ask our sales representative for a price quotation for your special requirements.

Contact us at:

PHYTEC Messtechnik GmbH
Robert-Koch-Straße 39
D-55129 Mainz
Germany
Phone: +49 6131 9221 - 32
Fax: +49 6131 9221 - 33

or by electronic mail:

sales@phytec.de

Document:	Quickstart Manual OSELAS.BSP( ) phyCARDs
Document Number:	L-754e_0 October 2010

How would you improve this manual?

Did you find any mistakes in this manual? page

Submitted by:

Customer number:	
Name:	
Company:	
Address:	

Return to:

PHYTEC Messtechnik GmbH
Robert-Koch-Str. 39
D-55129 Mainz

Fax: +49 (6131) 9221-26

Published by

© 2010 PHYTEC Messtechnik GmbH

Ordering No.L-754e_0

Printed in Germany