

OSELAS.BSP()

Phytec phyCORE-i.MX35

PHYTEC



Quickstart Manual

<http://www.oselas.com>

© 2009 by Pengutronix, Hildesheim

1284 2009-05-10 12:39:09 +0200 (So, 10 Mai 2009)

Contents

I	OSELAS Quickstart for Phytec phyCORE-i.MX35	4
1	Getting a working Environment	5
1.1	Download Software Components	5
1.2	PTXdist Installation	5
1.2.1	Main parts of PTXdist	5
1.2.2	Extracting the Sources	6
1.2.3	Prerequisites	7
1.2.4	Configuring PTXdist	8
1.3	Toolchains	9
1.3.1	Using Existing Toolchains	10
1.3.2	Building a Toolchain	10
1.3.3	Building the OSELAS.Toolchain for OSELAS.BSP-Phytec-phyCORE-11	10
1.3.4	Freezing the Toolchain	11
2	Building phyCORE-i.MX35's root filesystem	13
2.1	Extracting	13
2.2	Selecting a Software Platform	14
2.3	Selecting a Hardware Platform	14
2.4	Selecting a Toolchain	14
2.5	Building the Root Filesystem	15
2.6	Building an Image	15
3	Bootng Linux	16
3.1	Target Side Preparation	17
3.2	Stand-Alone Bootng Linux	18
3.2.1	Development Host Preparations	18
3.2.2	Preparations on the Embedded Board	18
3.2.3	Bootng the Embedded Board	19
3.3	Remote-Bootng Linux	19
3.3.1	Development Host Preparations	20
3.3.2	Preparations on the Embedded Board	20
3.3.3	Bootng the Embedded Board	20
4	Accessng Peripherals	22
4.1	NOR Flash	22
4.2	Serial TTYS	23
4.3	Network	23
4.4	I ² C Master	23
4.4.1	I ² C Realtime Clock RTC8564	23
4.4.2	I ² C device 24W32	24

4.5	Framebuffer	24
4.6	USB Host Controller	24
4.7	OneWire Interface	25
4.8	USB Gadget	25
5	Getting help	26
5.1	Mailing Lists	26
5.1.1	About PTXdist in particular	26
5.1.2	About embedded Linux in general	26
5.2	News Groups	26
5.2.1	About Linux in embedded environments	26
5.2.2	About general Unix/Linux questions	26
5.3	Chat/IRC	27
5.4	phyCORE-i.MX35 Support Maillist	27
5.5	Commercial Support	27

Part I

OSELAS Quickstart for Phytec phyCORE-i.MX35

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-11, 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-11.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 snippet 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 before 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 its 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 advised 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 delimited 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-11

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-11 board support package is

```
arm-1136jfs-linux-gnueabi_gcc-4.3.2_glibc-2.8_binutils-2.19_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/\ 
> arm-1136jfs-linux-gnueabi_gcc-4.3.2_glibc-2.8_binutils-2.19_kernel-2.6.27-sanitized.ptxconf
~/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 Freezing the Toolchain

As we build and install this toolchain with regular user permissions we should modify the permissions as a last step to avoid any later manipulation. To do so we could set all toolchain files to read only or change recursively the owner of the whole installation to user root.

This is an important step for reliability. Do not omit it!

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 \ 
> 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/0SELAS.Toolchain-1.99.3/architecture_part.`

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

`/opt/0SELAS.Toolchain-1.99.3/architecture_part/version_part.`

2 Building phyCORE-i.MX35's root filesystem

2.1 Extracting

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

```
~# tar -zxf OSELAS.BSP-Phytec-phyCORE-11.tar.gz
~# cd OSELAS.BSP-Phytec-phyCORE-11
```

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

```
~/OSELAS.BSP-Phytec-phyCORE-11# ls -l
```

```
total 44
-rw-r--r--  1 jbb users 4078 Dec  3 18:10 ChangeLog
-rw-r--r--  1 jbb users 1313 Nov  1 13:31 Kconfig
-rw-r--r--  1 jbb users 1101 Nov  4 21:05 TODO
drwxr-xr-x 10 jbb users 4096 Jan 14 17:33 configs/
drwxr-xr-x  3 jbb users 4096 Jan 14 15:08 documentation/
drwxr-xr-x  5 jbb users 4096 Nov 13 12:30 local_src/
drwxr-xr-x  5 jbb users 4096 Dec 15 10:19 patches/
drwxr-xr-x  6 jbb users 4096 Jun  8 2008 projectroot/
drwxr-xr-x  3 jbb users 4096 Nov  1 14:18 protocols/
drwxr-xr-x  4 jbb users 4096 Jan  8 16:28 rules/
drwxr-xr-x  3 jbb 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 currently 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-11 comes with a predefined configuration we select in the following step:

```
~/OSELAS.BSP-Phytec-phyCORE-11# ptxdist select \   
> 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-i.MX35:

```
~/OSELAS.BSP-Phytec-phyCORE-11# ptxdist platform \   
> configs/phyCORE-i.MX35-1.99.12-2/platformconfig  
info: selected platformconfig:  
      'configs/phyCORE-i.MX35-1.99.12-2/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/arm-1136jfs-linux-gnueabi/  
  gcc-4.3.2-glibc-2.8-binutils-2.19-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-11# ptxdist toolchain \   
> /opt/OSELAS.Toolchain-1.99.3/arm-1136jfs-linux-gnueabi/\   
> gcc-4.3.2-glibc-2.8-binutils-2.19-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-11# 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-i.MX35/root/` directory and a bunch of **.ipk* packets in the `platform-phyCORE-i.MX35/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-11# 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 additional options can be defined with

```
~/OSELAS.BSP-Phytec-phyCORE-11# ptxdist platformconfig
```

Then select the submenu "image creation options". The generated image will be placed into `platform-phyCORE-i.MX35/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-i.MX35/root/` will not be enclosed in the image. If custom files are needed for the target. Install it with `ptxdist`.

3 Booting Linux

Now that there is a root filesystem in our workspace we'll have to make it visible to the phyCORE-i.MX35. 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.

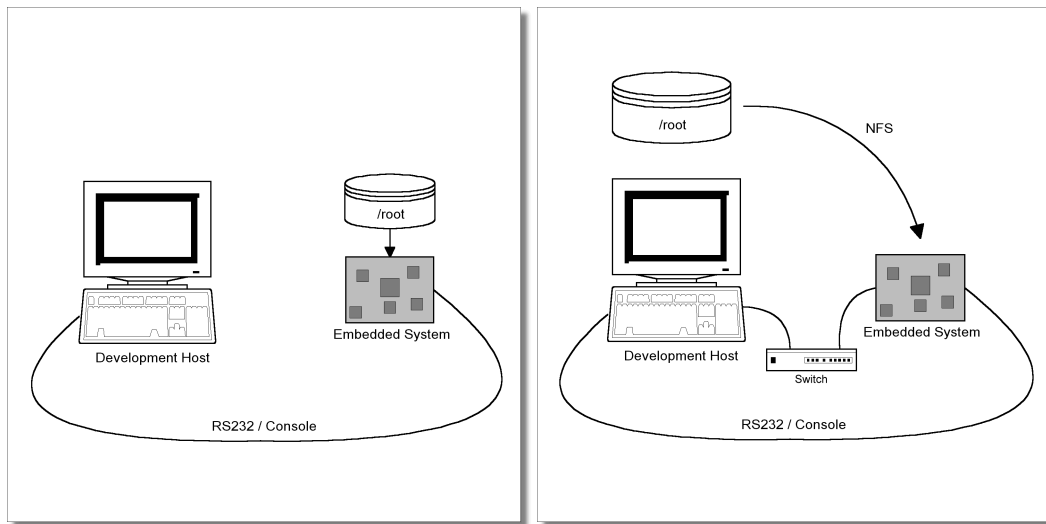


Figure 3.1: Booting the root filesystem, built with PTXdist, from the host via network and from flash.

Figure 3.1 shows both methods. The main method used in the OSELAS.BSP-Phytec-phyCORE-11 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-i.MX35 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-i.MX35/root/` directory in our PTXdist workspace.

The OSELAS.BSP-Phytec-phyCORE-11 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 challenge.

3.1 Target Side Preparation

The phyCORE-i.MX35 uses U-Boot as its bootloader. U-Boot can be customized with environment variables and scripts to support any boot constellation. OSELAS.BSP-Phytec-phyCORE-11 comes with a predefined environment setup to easily bring up the phyCORE-i.MX35.

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 U-Boot 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-phyCORE-11# 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 save our new settings.

Now the command

```
~/OSELAS.BSP-Phytec-phyCORE-11# ptxdist test setenv
```

will automatically set up a correct default environment on our phyCORE-i.MX35. 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 `platform-phyCORE-i.MX35/test.log` will give further information about why it has failed. Also extending the command line shown above by a `--debug` can help to see what's going wrong.



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.

3.2 Stand-Alone Booting Linux

To use the target standalone, the rootfs has to be made persistent in one of the onboard supported media of the phyCORE-i.MX35. The following sections describe the steps necessary to bring the rootfs into the onboard NOR 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 phyCORE-i.MX35 can work independently from the development host. We can "cut" the network (and serial cable) and the phyCORE-i.MX35 will continue to work.

3.2.1 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.

3.2.2 Preparations on the Embedded Board

To boot phyCORE-i.MX35 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 phyCORE-i.MX35's flash memory.

But first we must create the new root filesystem image prepared for its usage on the phyCORE-i.MX35:

```
~/OSELAS.BSP-Phytec-phyCORE-11# ptxdist images
```

To simplify this step, OSELAS.BSP-Phytec-phyCORE-11 comes with an automated setup procedure for this step. To use this procedure we run the command:

```
~/OSELAS.BSP-Phytec-phyCORE-11# 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 a root filesystem to the correct flash partition on the phyCORE-i.MX35. It only works if we previously have set up the environment variables successfully (described at page 17).

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 `platform-phyCORE-i.MX35/test.log` will give further information about why it has failed.

3.2.3 Booting the Embedded Board

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

```
~/OSELAS.BSP-Phytec-phyCORE-11# 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 `platform-phyCORE-i.MX35/test.log` 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.

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-i.MX35/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 which can fetch the kernel from a TFTP server.

3.3.1 Development Host Preparations

If we already have booted the phyCORE-i.MX35 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-phyCORE-11# sudo cp platform-phyCORE-i.MX35/images/linuximage  
/tftpboot/uImage-pcm043
```

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.

3.3.2 Preparations on the Embedded Board

We already provided the phyCORE-i.MX35 with the default environment at page 17. So there is no additional preparation required here.

3.3.3 Booting the Embedded Board

The default environment settings coming with the OSELAS.BSP-Phytec-phyCORE-11 has the possibility to boot from the internal flash or from the network. Configuration happens in the file `/env/config`. As U-Boot-v2 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 `nor`, `nand` or `net`. `nor` let the boot script load everything from the internal NOR flash memory, `nand` 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 phyCORE-i.MX35.

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 U-Boot-v2 uses for the environment. To store it to the persistent memory, an additional `saveenv` command is required.

Now its time to boot the phyCORE-i.MX35. To do so, simply run:

```
uboot: / boot
```

This command should boot phyCORE-i.MX35 into the login prompt.

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

4 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 phyCORE-i.MX35 starter kit consists of the following individual boards:

1. The phyCORE-i.MX35 module itself (PCM-037), containing the i.MX31, RAM, flash and several other peripherals.
2. The starter kit baseboard (PCM970).

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 baseboards 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

1. `arch/arm/mach-mx3/pcm043.c` for the CPU module

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.



Note that the huge variety of possibilities offered by the phyCORE 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 the Pengutronix support (support@pengutronix.de) and ask for a dedicated offer.

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

4.1 NOR Flash

Linux offers the Memory Technology Devices Interface (MTD) to access low level flash chips, directly connected to a SoC CPU.

Modern kernels offer a method to define flash partitions on the kernel command line, using the `mtddparts` command line argument:

```
physmap-flash.0:256k(uboot)ro,128k(ubootenv),2M(kernel),-(root)
```

This line, for example, specifies several partitions with their size and name which can be used as `/dev/mtd0`, `/dev/mtd1` etc. from Linux. Additionally, this argument is also understood by reasonably new U-Boot bootloaders, so if there is any need to change the partitioning layout, the U-Boot environment is the only place where the layout has to be changed.

From userspace the NOR flash partitions can be accessed as

- `/dev/mtdblock0` (e.g. U-Boot partition)
- `/dev/mtdblock1` (e.g. U-Boot environment partition)
- `/dev/mtdblock2` (e.g. Kernel partition)
- `/dev/mtdblock3` (e.g. Linux rootfs partition)

Note: This is an example only. The partitioning on our phyCORE-i.MX35 target can differ from this layout.

Only the `/dev/mtdblock3` on the phyCORE-i.MX35 has a filesystem, so the other partitions cannot be mounted into the rootfs. The only way to access them is by pushing a prepared flash image into the corresponding `/dev/mtd` device node.

4.2 Serial TTYs

The i.MX35 SoC supports up to 4 so called UART units. On the phyCORE-i.MX35 two UARTs are routed to the connectors and can be used in user's application.

- `ttymxc0` at connector P1 (bottom connector) used as the main kernel and control console.
- `ttymxc2` at connector P1 (top connector). Unused in this BSP

4.3 Network

The phyCORE-i.MX35 module features ethernet OnChip, 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.

4.4 I²C Master

The i.MX35 processor based phyCORE-i.MX35 supports a dedicated I²C controller onchip. The kernel supports this 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.

4.4.1 I²C Realtime Clock RTC8564

Due to the Real Time Clock framework of the kernel the RTC8564 clock chip 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-phyCORE-11 tries to set up the date at system startup. If there was a powerfail `hwclock` will state:

```
pcf8564 0-0051: low voltage detected, date/time is not reliable.  
pcf8564 0-0051: 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.

4.4.2 I²C device 24W32

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

This type of memory is accessible through the sysfs filesystem. To read the EEPROM content simply `open()` the entry `/sys/bus/i2c/devices/0-0052/eeprom` and use `fseek()` and `read()` to get the values.

4.5 Framebuffer

This driver gains access to the display via device node `/dev/fb0`. For this BSP the Hitachi TX09D70VM1CCA display with a resolution of 240x320 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.



Earlier system revisions are using the SHARP LQ035Q7DH06 display instead. To continue using this display U-Boot's environment or the BSP must be changed.

In order to continue using the predecessor display the kernel must be setup with the `video=mx3fb:Sharp-LQ035Q7` parameter.

If the target system is already configured, this parameter can be changed from inside the running U-Boot: Modify the file `/env/config` and replace the current `video=mx3fb:TX090` with the string shown above.

To modify the BSP to make this change persistent we can simply modify the `configs/phyCORE-i.MX35-1.99.12-2/u-boot` and also replace the `video=mx3fb:TX090` by the `video=mx3fb:Sharp-LQ035Q7` string. With this modification target's auto setup will do the right thing.

4.6 USB Host Controller

The i.MX35 CPU embeds 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-11 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.

4.7 OneWire Interface

"There is something like 1W existing in this universe."

As the real support for this kind of devices is currently very broken, only a direct access is provided. 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 following output:

```
root@phyCORE:~ 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
```

4.8 USB Gadget

Linux can run USB protocol in the host, or in the peripheral mode. With OSELAS.BSP-Phytec-phyCORE-11 BSP the USB peripheral mode is supported as well. This means that we can connect our phyCORE-i.MX35 to a USB host. Depending on which driver we load the phyCORE-i.MX35 can act as various USB gadgets. At default the file backed storage gadget driver and the network gadget driver are shipped. In the following is a short description about how to use the file backed storage gadget. For more information about USB gadgets please consult <http://www.linux-usb.org/gadget/>:

5 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.

5.1 Mailing Lists

5.1.1 About PTXdist in particular

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

http://www.pengutronix.de/maillinglists/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.

5.1.2 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/maillinglists/index_de.html

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

5.2 News Groups

5.2.1 About Linux in embedded environments

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

`comp.os.linux.embedded`

5.2.2 About general Unix/Linux questions

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

`de.comp.os.unix.programming`

5.3 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.

5.4 phyCORE-i.MX35 Support Maillist

OSELAS.Phytec@pengutronix.de

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

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

5.5 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