OSELAS.BSP( )
Phytec phyCORE-i.MX27

Quick Start Manual

http://www.oselas.com

983 2008-11-14 17:48:06 +0100 (Fri, 14 Nov 2008)

Contents

I OSELAS Quickstart for Phytec phyCORE-i.MX27 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 Abstract . . . 9
  - 1.3.2 Using Existing Toolchains . . . 10
  - 1.3.3 Building a Toolchain . . . 10
  - 1.3.4 Freezing the Toolchain . . . 12

2 Building phyCORE-i.MX27's root filesystem 13
- 2.1 Extracting . . . 13
- 2.2 Selecting a Userland Configuration . . . 14
- 2.3 Selecting a Target Configuration . . . 14
- 2.4 Selecting a Toolchain to Use . . . 14
- 2.5 Compiling the Root Filesystem . . . 14

3 Booting Linux 16
- 3.1 Target Side Preparation . . . 17
- 3.2 Remote-Booting Linux . . . 17
  - 3.2.1 Development Host Preparations . . . 18
  - 3.2.2 Preparations on the Embedded Board . . . 18
  - 3.2.3 Booting the Embedded Board . . . 18
- 3.3 Stand-Alone Booting Linux . . . 19
  - 3.3.1 Development Host Preparations . . . 19
  - 3.3.2 Preparations on the Embedded Board . . . 19
  - 3.3.3 Booting the Embedded Board . . . 19

4 Accessing Peripherals 21
- 4.1 NOR Flash . . . 21
- 4.2 NAND Flash . . . 22
  - 4.2.1 NAND Usage . . . 22
  - 4.2.2 NAND Preparation . . . 22

4.3 SRAM Memory . . . 23
4.4 Serial TTYs . . . 23
4.5 Network . . . 23
4.6 SPI . . . 23
4.7 Framebuffer . . . 24
4.8 Touch . . . 24
4.9 I²C Master . . . 24
4.9.1 I²C Realtime Clock (RTC8564) . . . 24
4.9.2 I²C device 24W32 . . . 25
4.9.3 I²C device LM75 . . . 25
4.10 CPU core frequency scaling . . . 25
4.11 AUDIO support . . . 25
4.11.1 Audio Sources and Sinks . . . 25
4.11.2 Playback . . . 28
4.11.3 Capture . . . 28
4.12 USB Host Controller . . . 29
4.13 OneWire Interface . . . 29
4.14 MMC/SD Card . . . 29

5 Special Notes 30
5.1 About Socket-CAN . . . 30
5.1.1 Starting and Configuring Interfaces from the Command Line . . . 30
5.1.2 Using the CAN Interfaces from the Command Line . . . 31
5.2 Analysing the CAN Bus Data Transfer . . . 32

6 Getting help 33
6.1 Mailing Lists . . . 33
6.2 News Groups . . . 33
6.3 Chat/IRC . . . 34
6.4 Miscellaneous . . . 34
6.5 phyCORE-i.MX27 Support Maillist . . . 34
6.6 Commercial Support . . . 34

Part I

OSELAS Quickstart for Phytec phyCORE-i.MX27

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-i.MX27-7.1, the following archives have to be available on the development host:

- ptxdist-1.99.8.tgz
- ptxdist-1.99.8-patches.tgz
- OSELAS.BSP-Phytec-phyCORE-i.MX27-7.1.tar.gz
- OSELAS.Toolchain-1.99.1.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 build a root filesystem for embedded Linux devices. In order to start development with PTXdist it is necessary to install the software on the development system.

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

1.2.1 Main parts of PTXdist

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

PTXdist consists of the following parts:

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

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

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

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

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

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

1.2.2 Extracting the Sources

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

ptxdist-1.99.8.tgz The PTXdist software itself.

ptxdist-1.99.8-patches.tgz All patches against upstream software packets (known as the 'patch repository').

ptxdist-1.99.8-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-1.99.8.tgz
~/local# tar -zxf ptxdist-1.99.8-patches.tgz
```

and if required the generic projects:

```
~/local# tar -zxf ptxdist-1.99.8-projects.tgz
```

If everything goes well, we now have a PTXdist-1.99.8 directory, so we can change into it:

```
~/local# cd ptxdist-1.99.8
~/local/ptxdist-1.99.8# ls -l

total 455
drwxr-xr-x   2 jb users   1024 29. Sep 17:32 autoconf/
-rwxr-xr-x   1 jb users     28 29. Sep 17:32 autogen.sh*
```

```
drwxr-xr-x   2 jb users   1024 29. Sep 17:32 bin/
-rw-r--r--   1 jb users 115470 29. Sep 17:32 ChangeLog
drwxr-xr-x   5 jb users   1024 29. Sep 17:32 config/
-rwxr-xr-x   1 jb users 222451 29. Sep 17:34 configure*
-rw-r--r--   1 jb users  11445 29. Sep 17:34 configure.ac
-rw-r--r--   1 jb users  18361 29. Sep 17:32 COPYING
-rw-r--r--   1 jb users   3499 29. Sep 17:32 CREDITS
drwxr-xr-x   2 jb users   1024 29. Sep 17:30 debian/
drwxr-xr-x   2 jb users   1024 29. Sep 17:32 Documentation/
drwxr-xr-x   7 jb users   1024 29. Sep 17:32 generic/
-rw-r--r--   1 jb users     58 29. Sep 17:32 INSTALL
-rw-r--r--   1 jb users   2150 29. Sep 17:32 Makefile.in
drwxr-xr-x 159 jb users   4096 29. Sep 17:31 patches/
drwxr-xr-x   2 jb users   1024 29. Sep 17:32 platforms/
drwxr-xr-x   4 jb users   1024 29. Sep 17:30 plugins/
-rw-r--r--   1 jb users   4091 29. Sep 17:32 README
-rw-r--r--   1 jb users    691 29. Sep 17:32 REVISION_POLICY
drwxr-xr-x   6 jb users  28672 29. Sep 17:32 rules/
drwxr-xr-x   6 jb users   1024 29. Sep 17:30 scripts/
drwxr-xr-x   2 jb users   1024 29. Sep 17:32 tests/
-rw-r--r--   1 jb users  33418 29. Sep 17:32 TODO
```

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-1.99.8# ./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 1.99.8 configured.
Using '/usr/local' for installation prefix.


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

```
Report bugs to ptxdist@pengutronix.de
```

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

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

One of the most important tasks for the `configure` script is to find out if all the programs PTXdist depends on are already present on the development host. The script will stop with an error message in case something is missing. If this happens, the missing tools have to be installed from the distribution befor re-running the `configure` script.

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

```
~/local/ptxdist-1.99.8# make
```

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

```
~/local/ptxdist-1.99.8# 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-1.99.8# 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.

1.2.4.1 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>`

1.2.4.2 Source Archive Location

Whenever PTXdist downloads source archives it stores it project locally. 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.

1.2.4.3 Generic Project Location

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

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

`/usr/local/lib/ptxdist-1.99.8/projects:/office/my_projects/ptxdist`

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

1.3 Toolchains

1.3.1 Abstract

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.

1.3.2 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 menuconfig
```

and navigate to `PTXdist Config, Architecture` and `Check for specific toolchain vendor`. Clear this entry to disable the toolchain vendor check.

1.3.3 Building a Toolchain

PTXdist-1.99.8 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-1.99.1/`.

Usually the `/opt` directory is not world writable. So in order to build our OSELAS.Toolchain into that directory we need to use a root account to change the permissions so that the user can write (`mkdir /opt/OSELAS.Toolchain-1.99.1 ; chown <username> /opt/OSELAS.Toolchain-1.99.1; chmod a+rwx /opt/OSELAS.Toolchain-1.99.1`).

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* option to define the toolchain to be used on a per project base.

1.3.3.1 Building the OSELAS.Toolchain for OSELAS.BSP-Phytec-phyCORE-i.MX27-7.1

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-i.MX27-7.1 board support package is

```
arm-v5te-linux-gnueabi_gcc-4.1.2_glibc-2.5_binutils-2.17_kernel-2.6.18
```

So the steps to build this toolchain are:

```
~# tar xf OSELAS.Toolchain-1.99.1.tar.bz2
~# cd OSELAS.Toolchain-1.99.1
~/OSELAS.Toolchain-1.99.1# ptxdist select
  ptxconfigs/arm-v5te-linux-gnueabi_gcc-4.1.2_glibc-2.5_binutils-2.17_kernel-2.6.18.ptxconfig
~/OSELAS.Toolchain-1.99.1# 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
- 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!

1.3.4.1 Building additional Toolchains

The OSELAS.Toolchain-1.99.1 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 projekt, removing the current `selected_ptxconfig` link and creating a new one.

```
~/OSELAS.Toolchain-1.99.1# ptxdist clean
~/OSELAS.Toolchain-1.99.1# rm selected_ptxconfig
~/OSELAS.Toolchain-1.99.1# ptxdist select
       ptxconfigs/any_another_toolchain_def.ptxconfig
~/OSELAS.Toolchain-1.99.1# ptxdist go
```

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

`/opt/OSELAS.Toolchain-1.99.1/architecture_part.`

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

`/opt/OSELAS.Toolchain-1.99.1/architecture_part/version_part.`

2 Building phyCORE-i.MX27'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-i.MX27-7.1.tar.gz
~# cd OSELAS.BSP-Phytec-phyCORE-i.MX27-7.1
```

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

```
~/OSELAS.BSP-Phytec-phyCORE-i.MX27-7.1# ls -l
```

```
total 44
-rw-r--r--  1 jb users 3586 Jun  8 12:13 ChangeLog
-rw-r--r--  1 jb users 1313 Oct 31 16:34 Kconfig
drwxr-xr-x 10 jb users 4096 Oct 25 13:48 configs/
drwxr-xr-x  3 jb users 4096 Oct 20 16:43 documentation/
-rw-r--r--  1 jb users 6171 Oct 25 15:29 logfile
drwxr-xr-x  4 jb users 4096 Oct 20 16:50 patches/
drwxr-xr-x  6 jb users 4096 Jun  8 12:13 projectroot/
drwxr-xr-x  3 jb users 4096 Jun  8 12:12 protocols/
drwxr-xr-x  3 jb users 4096 Oct 31 16:34 rules/
drwxr-xr-x  3 jb users 4096 Oct 31 16:34 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 Userland Configuration

Userland is more generic as the kernel and – for example – the boot loader. So mostly there is only one configuration setup for userland. But that is not true for all BSPs. There may be special cases where also userland must be built in a different way per target. So PTXdist can be configured to support also these special cases. In this example we are using a generic configuration file:

```
~/OSELAS.BSP-Phytec-phyCORE-i.MX27-7.1# ptxdist select
     configs/ptxconfig
info: selected ptxconfig:
     'configs/ptxconfig'
```

2.3 Selecting a Target Configuration

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.MX27:

```
~/OSELAS.BSP-Phytec-phyCORE-i.MX27-7.1# ptxdist platform
     configs/phyCORE-i.MX27/platformconfig
info: selected platformconfig:
     'configs/phyCORE-i.MX27-trunk/platformconfig'
```

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

```
found and using toolchain:
'/opt/OSELAS.Toolchain-1.99.1/arm-v5te-linux-gnueabi/
     gcc-4.1.2-glibc-2.5-binutils-2.17-kernel-2.6.18/bin'
```

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

2.4 Selecting a Toolchain to Use

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-i.MX27-7.1# ptxdist toolchain
     /opt/OSELAS.Toolchain-1.99.1/arm-v5te-linux-gnueabi/
     gcc-4.1.2-glibc-2.5-binutils-2.17-kernel-2.6.18/bin
```

2.5 Compiling the Root Filesystem

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

```
~/OSELAS.BSP-Phytec-phyCORE-i.MX27-7.1# 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 `root/` directory and a bunch of .ipkg packets in the `images/` directory, containing the single applications the root filesystem consists of.

3 Booting Linux

Now that there is a root filesystem in our workspace we'll have to make it visible to the phyCORE-i.MX27. There are two possibilities to do this:

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

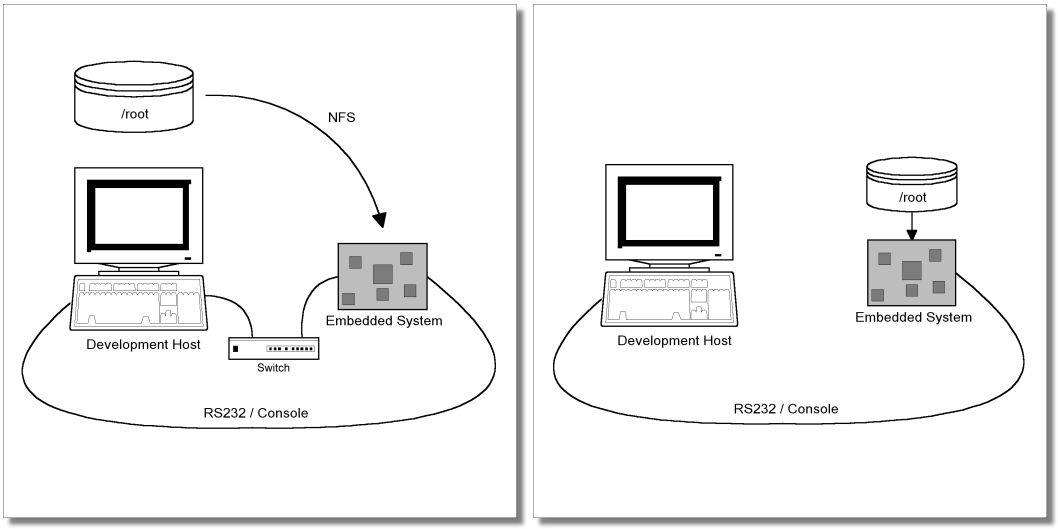

Figure 3.1: Booting the root filesystem, built with PTXdist, from the host via network and from flash.

Figure 3.1 shows both methods. On the left side the development host is connected to the phyCORE-i.MX27 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 it's root filesystem via NFS from the original location of the root/ directory in our PTXdist workspace.

The other way is to provide all needed components to run on the traget itself. The Linux kernel and the root filesystem is persistent in target's flash. This means the only connection needed is the nullmodem cable to see what is happen on our target.

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

3.1 Target Side Preparation

The phyCORE-i.MX27 uses U-Boot as its bootloader. U-Boot can be customised with environment variables to support any boot constellation. OSELAS.BSP-Phytec-phyCORE-i.MX27-7.1 comes with a predefined environment setup to easily bring up the phyCORE-i.MX27.

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 some of the values of these U-Boot's 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-i.MX27-7.1# 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.

"Exit" the dialouge and and save your new settings.

The command

```
~/OSELAS.BSP-Phytec-phyCORE-i.MX27-7.1# ptxdist test setenv
```

now will automatically set up a correct default environment on the phyCORE-i.MX27.
It should output a line like this when it was successful:

```
uboot: set default environment PASS
```

Note: If it fails, reading `test.log` will give further information about why it has failed.

We now must restart the phyCORE-i.MX27 to activate the new environment settings. Then we should run the `ping` command on the target's ip address to check if the network settings are working correctly on the target.

3.2 Remote-Booting Linux

The first method we probably want to try after building a root filesystem is the network-remote boot variant. 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.

The network boot method has the advantage that we don't have to do any flashing at all to "see" a file on the target board: All we have to do is to copy it to some location in the `root/` directory and it simply "appears" on the embedded device. This is especially helpful during the development phase of a project, where things are changing frequently.

3.2.1 Development Host Preparations

On the development host a TFTP server has to be installed and configured. The exact method to do 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 kernel images 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 gain access; a safe method is to use the `sudo(8)` command to push our kernel:

```
~# sudo cp platform-phyCORE-i.MX27/images/linuximage /tftpboot/uImage-pcm038
```

The NFS server is not restricted to a certain filesystem location, so all we have to do on most distributions is to configure `/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.

3.2.2 Preparations on the Embedded Board

We already provided the phyCORE-i.MX27 with the default environment at page 17. So there is no additional preparation required here.

3.2.3 Booting the Embedded Board

The default environment coming with the OSELAS.BSP-Phytec-phyCORE-i.MX27-7.1 has a predefined script for booting from NFS. To use it, we can simple enter

```
uboot> run bootcmd_net
```

This command should boot phyCORE-i.MX27 into the login prompt.

As U-Boot automatically runs the `bootcmd` environment variable as a script after power-on, we set this variable to start from NFS automatically:

```
uboot> setenv bootcmd 'run bootcmd_net'
```

After the next reset or powercycle of the board it should boot the kernel from the TFTP server, start it and mount the root filesystem via NFS.

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

3.3 Stand-Alone Booting Linux

Usually, after working with the NFS-Root system for some time, the rootfs has to be made persistent in the onboard flash of the phyCORE-i.MX27, without requiring the network infrastructure any more. The following sections describe the steps necessary to bring the rootfs into the onboard 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.MX27 can work independently from the development host. We can "cut" the network (and serial cable) and the phyCORE-i.MX27 will continue to work.

3.3.1 Development Host Preparations

If we already booted the phyCORE-i.MX27 remotely (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 gain access.

3.3.2 Preparations on the Embedded Board

To boot phyCORE-i.MX27 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.MX27's flash memory. To simplify this, OSELAS.BSP-Phytec-phyCORE-i.MX27-7.1 comes with an automated setup procedure for this step.

To use this procedure run the command

```
~# 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.MX27. It only works if we previously have set up the environment variables successfully (described at page 17).
The command should output a line like this when it was successful:

```
u-boot: flashing root image PASS
```

Note: If it fails, reading `test.log` will give further information about why it has failed.

3.3.3 Booting the Embedded Board

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

```
~# ptxdist test boot
```

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

To do it manually, the default environment coming with the OSELAS.BSP-Phytec-phyCORE-i.MX27-7.1 has a predefined script for booting stand-alone. To use it, we can simply enter

```
uboot> run bcmd_flash
```

This command should boot phyCORE-i.MX27 into the login prompt.

As U-Boot automatically runs the `bootcmd` environment variable as a script after power-on, we set this variable to start from NFS automatically:

```
uboot> setenv bootcmd 'run bcmd_flash'
```

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 Accessing Peripherals

The following sections provide an overview of the supported hardware components and their corresponding operating system drivers. As there is currently no specification about what the BSP shall support, this is a more or less arbitrary selection. Further changes can be ported on demand of the customer.

Phytec's phyCORE-i.MX27 starter kit consists of the following individual boards:

1. The phyCORE-i.MX27 module itself (PCM-038), containing the i.MX27, 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-pxa/pcm038.c` for the CPU module
2. `arch/arm/mach-pxa/pcm970-baseboard.c` for the baseboard

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.

Older versions of the Linux kernel had separate mapping drivers for each board, specifying the flash layout in a driver. Modern kernels offer a method to define flash partitions on the kernel command line, using the "mtdparts" command line argument:

```
mtdparts=phys_mapped_flash:128k(uboot)ro,128k(ubootenv),1536k(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. In this section we assume that the standard configuration delivered with the OSELAS.BSP-Phytec-phyCORE-i.MX27-7.1 is being used.

From userspace the flash partitions can be accessed as

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

Only `/dev/mtdblock3` has a filesystem, so the other partition 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 NAND Flash

The phyCORE-i.MX27 module comes with a 64MiB NAND memory to be used as an additional media to store 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 change to bring more than 64MiB of data into this device.

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

4.2.1 NAND Usage

The OSELAS.BSP-Phytec-phyCORE-i.MX27-7.1 will mount the whole NAND memory to `/media/nand` while system startup. Its up to us user to store usefull data on it.

4.2.2 NAND Preparation

On a fresh phyCORE-i.MX27 the NAND memory can be unformatted. In this case the mount will fail. To prepare it for later usage, we must prepare it by erasing the whole memory.

```
~# flash_eraseall /dev/mtd4
```

After erasing, mounting is possible via:

```
~# mount /media/nand
```

Note: Mounting this memory the first time after erasing it can take a few seconds. In this case the JFFS2 filesystem does its own preparation of the memory in the background.

Now this filesystem is available through `/media/nand` and can be used like any other filesystem.

4.3 SRAM Memory

A phyCORE-i.MX27 is equipped with an 512kiB SRAM. The OSELAS.BSP-Phytec-phyCORE-i.MX27-7.1 uses this memory as an additional filesystem for regular usage.

On a fresh phyCORE-i.MX27 the SRAM memory can be unformatted. In this case the mount will fail. To prepare it for later usage, we must prepare it by formatting the whole memory.

```
~# mkfs.minix -n 30 -v /dev/mtdblock5
```

After formatting, mounting is possible via:

```
~# mount /media/sram
```

Now this filesystem is available through `/media/sram` and can be used like any other filesystem.

4.4 Serial TTYs

The i.MX27 SoC supports up to 6 so called UART units. On the phyCORE-i.MX27 three 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.
- `ttymxc1` at connector P1 (top connector). Unused in this BSP
- `ttymxc2` at expansion connector. Unused in this BSP

4.5 Network

The i.MX27 CPU embedds an Fast Ethernet Controller (FEC) onchip, which is being used to provide the etho network interface. The interface offers a standard Linux network port which can be programmed using the BSD socket interface.

4.6 SPI

This BSP currently supports one dedicated SPI bus. Its used to control the external so called PMIC, the main peripheral controller. Until now, the only used PMIC device is the ADC to scan the touch.

4.7 Framebuffer

This driver gains access to the display via device node `/dev/fb0`. For this BSP only the SHARP LQ035Q7DH06 display with a resolution of 230x320 is supported.

A simple test of this feature can be run with:

```
~# fbtest
```

This will show various pictures on the display.

4.8 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.

4.9 I²C Master

The i.MX27 processor based phyCORE-i.MX27 supports two dedicated I²C controllers on chip. The kernel supports these controller as master controllers.

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.

4.9.1 I²C Realtime Clock (RTC8564)

This device is connected to I²C bus 1 and uses the address 0x51 as default.

Due to the kernel's Real Time Clock framework the RTC8564 clock chip can be accessed using the same tools as any other clocks.

Date and time can be manipulated with the `hwclock` tool, using the *–systohc* and *–hctosys* options. For more information about this tool refer to hwclock's manpages.

4.9.2 I²C device 24W32

This device is a 4kByte non-volatile memory. Its connected to I²C bus 1 and uses the address 0x52 as default.

This type of memory is accessible through the sysfs filesystem. To read the EEPROM contents simply `open()` the entry `/sys/bus/i2c/devices/1-0052/eeprom` and use `fseek()` and `read()` to get the values. When using `write()` instead of `read()` changing EEPROM's content is also possible.

4.9.3 I²C device LM75

This device is connected to I²C bus 1 and uses the address 0x4a as default.

Not yet supported. Note: This device is an option and most of the time not soldered.

4.10 CPU core frequency scaling

This kernel supports the switching of the CPU core frequency. As there are various restrictions what frequencies can be used (and what power supply voltage must be valid to run them) only the full speed (399MHz), a middle speed (266MHz) and one low speed (133MHz) are currently available.

So called governors are selecting one of this frequencies in accordance to their goals. Available governors are:

performance Always selects the highest possible CPU core frequency.

powersave Always selects the lowest possible CPU core frequency.

conservative Switches between possible CPU core frequencies in reference to the current system load. It follows the system load very slowly (integrate).

ondemand Switches between possible CPU core frequencies in reference to the current system load. When the system load increases above a specific limit it increases the CPU core frequency immediately. This is the default governor when the system starts up.

For calibrating governor's settings refer kernel's documentation in: `Documentation/cpu-freq/governors.txt`.

We will find the reported sysfs entries on our target below: `/sys/devices/system/cpu/cpu0/cpufreq/`.

4.11 AUDIO support

Audio support comes with the processor internal synchronous serial units together with the separate PMIC device.

4.11.1 Audio Sources and Sinks

The PMIC device contains various audio output and input devices.

List of output devices:

- Mono loudspeaker

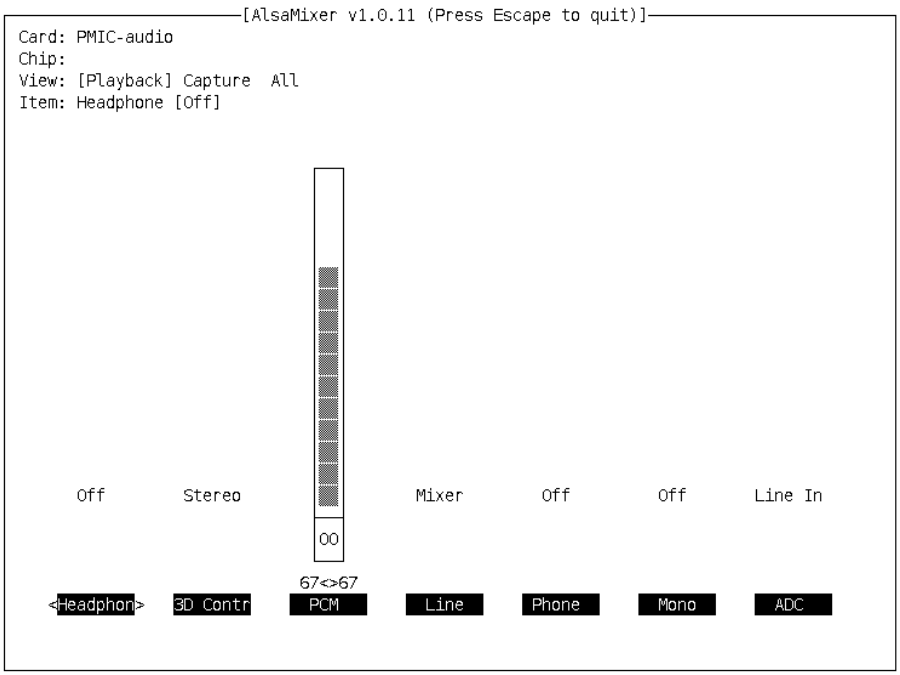

Figure 4.1: Playback controls

- Headphone loudspeaker (stereo)
- Earpiece loudspeaker (mono)
- Line Out

List of input devices:

- Line In (stereo)
- Phone microphone (mono)
- Handset microphone (stereo)

Note: Not all in- and output may be available. It depends on the baseboard what audio sinks and sources are possible.

For the pcm970 evaluation board *Line Out*, *Line In* and the *Handset microphone* can be connected through three jacks:

X13 Line In

X12 Line Out

X11 Microphon In

Multiple output sinks can be enabled through the *alsamixer* command. Refer figure 4.1 for the screen output.

Select the screen *Playback* for this view (using the tabulator key).

Controls in this view:

Headphone Controls the headphone amplifier and the source to be output

- **Off** Headphone amplifiers are powered down.
- **Codec** Codec's output is routed through the amplifier (no supported yet)
- **Mixer** Mixer's output is routed through the amplifier

Line Controles the Line Out amplifier and the source to be output

- **Off** Headphone amplifiers are powered down.
- **Codec** Codec's output is routed through the amplifier (no supported yet)
- **Mixer** Mixer's output is routed through the amplifier

Phone Controles the earpiece amplifier and the source to be output

- **Off** Earpiece amplifier is powered down.
- **Codec** Codec's output is routed through the amplifier (no supported yet)
- **Mixer** Mixer's output is routed through the amplifier

Mono Controles the loadspeaker amplifier and the source to be output

- **Off** Loadspeaker amplifier is powered down.
- **Codec** Codec's output is routed through the amplifier (no supported yet)
- **Mixer** Mixer's output is routed through the amplifier

3D Contr Controles mixer effects

- **Stereo** Mixer forwards two independent audio channels.
- **Phase Mi** Mixer changes one channel's phase (aka wide stereo)
- **Mono** Mixer adds both input channels and outputs the result at both output channels
- **Mono Mix** Mixer adds both input channels and inverts one phase (aka pseudo stereo)

PCM Controles DAC's output volume into the mixer

ADC Controls the input source into the ADC unit (codec). Note: This is a *Capture* control, but *alsamixer* decides to display it in the *Playback* view.

- **Off** Codec is powerd down.
- **Line In** Stereo Line in is routed to the codec.
- **Handmic** Stereo handset microphone is routed to the codec.
- **Headmic** Mono headset microphone is routed to the codec.

To control the capture controls, change to the *Capture* view. Refer figure 4.2 for the screen output.

Controls in this view:

PCM Controls the volume of the input amplifier into the ADC.

Line Byp Controls the volume of the *Line In* into the mixer. Pressing the space bar enables or disables this audio bypass.

alsamixer can be left by pressing the *ESC* key. If we want these settings to be persistent we 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/ini.d/sound* script.

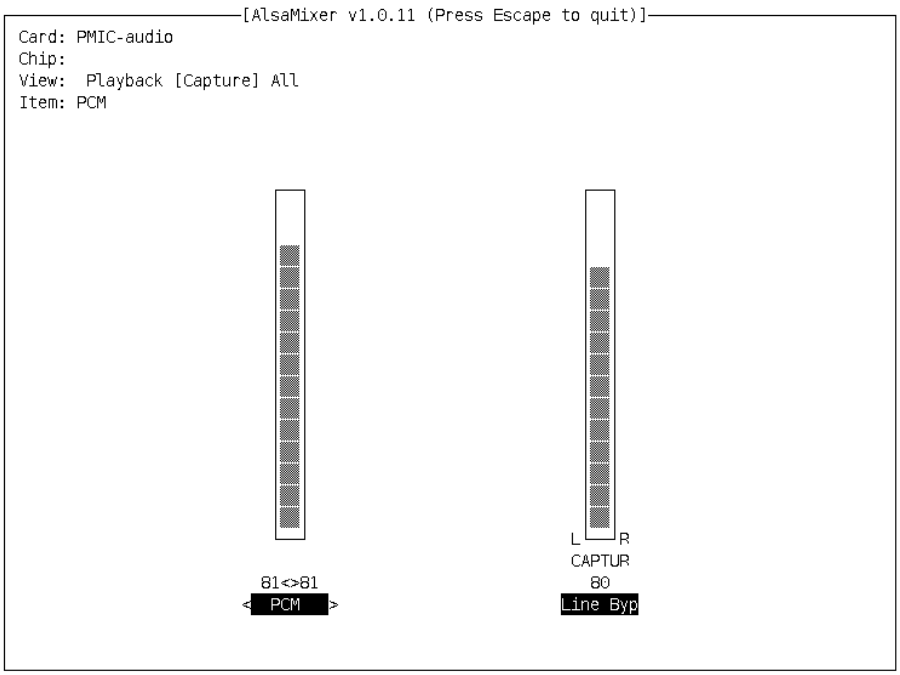

Figure 4.2: Capture controls

4.11.2 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 /greet.wav
```

4.11.3 Capture

arecord is a command line tool for capturing audio streams. First we must select the audio source to capture. Default is *Line In*. We can use the *alsamixer* to switch to a different source.

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.

4.12 USB Host Controller

The i.MX27 CPU embedds a USB 2.0 EHCI controller that is also able to handle low and full speed devices (USB 1.1).

The OSELAS.BSP-Phytec-phyCORE-i.MX27-7.1 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.

4.13 OneWire Interface

FIXME

4.14 MMC/SD Card

FIXME

5 Special Notes

5.1 About Socket-CAN

The CAN (Controller Area Network[1]) bus offers a low-bandwidth, prioritised message fieldbus for communication between microcontrollers. Unfortunately, CAN was not designed with the ISO/OSI layer model in mind, so most CAN APIs available throughout the industry don't support a clean separation between the different logical protocol layers, like for example known from ethernet.

The *Socket-CAN* framework for Linux extends the BSD socket API concept towards CAN bus. It consists of

- a core part (candev.ko)
- chip drivers (e. g. mscan, sja1000 etc.)

So in order to start working with CAN interfaces we'll have to make sure all necessary drivers are loaded.

5.1.1 Starting and Configuring Interfaces from the Command Line

If all drivers are present in the kernel, "ifconfig -a" shows which network interfaces are available; as Socket-CAN chip interfaces are normal Linux network devices (with some additional features special to CAN), not only the ethernet devices can be observed but also CAN ports.

For this example, we are only interested in the first CAN port, so the information for `can0` looks like

```
~# ifconfig can0
can0 Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
     inet addr:127.42.23.180  Mask:255.255.255.0
     UP RUNNING NOARP  MTU:16  Metric:1
     RX packets:35948 errors:0 dropped:0 overruns:0 frame:0
     TX packets:1 errors:0 dropped:0 overruns:0 carrier:0
     collisions:0 txqueuelen:10000
     RX bytes:243744 (238.0 KiB)  TX bytes:2 (2.0 B)
     Interrupt:145 Base address:0x900
```

The output contains the usual parameters also shown for ethernet interfaces, so not all of these are necessarily relevant for CAN (for example the MAC address). These parameters contain useful information:

Inferfaces shown by the "ifconfig -a" command can be configured with `canconfig`. This command adds CAN specific configuration possibilities for network interfaces, similar to for example `iwconfig` for wireless ethernet cards.

The baudrate for cano can now be changed:

[1] ISO 11898/11519

Field	Description
can0	Interface Name
NOARP	CAN cannot use ARP protocol
MTU	Maximum Transfer Unit, always 8
RX packets	Number of Received Packets
TX packets	Number of Transmitted Packets
RX bytes	Number of Received Bytes
TX bytes	Number of Transmitted Bytes
errors...	Bus Error Statistics

Table 5.1: CAN interface information

```
~# canconfig can0 bitrate 250000
```

and the interface is started with

```
~# ifconfig can0 up
```

5.1.2 Using the CAN Interfaces from the Command Line

After successfully configuring the local CAN interface and attaching some kind of CAN devices to this physical bus, we can test this connection with command line tools.

The tools `cansend` and `candump` are dedicated to this purpose.

To send a simple CAN message with ID 0x20 and one data byte of value 0xAA just enter:

```
~# cansend can0 --identifier=0x20 0xAA
```

To receive CAN messages run the `candump` command:

```
~# candump can0
interface = can0, family = 29, type = 3, proto = 0
<0x020> [1] aa
```

The output of `candump` shown in this example was the result of running the `cansend` example above on a different machine.

See cansend's and candump's manual pages for further information about using and options.

5.2 Analysing the CAN Bus Data Transfer

The OSELAS.BSP-Phytec-phyCORE-i.MX27-7.1 BSP comes with the standard *pcap* library and `tcpdump` tool. Both are capable of analyzing CAN data transfer wich includes time stamping.

We set up the CAN interface(s) as usual and use it in our application. With `tcpdump` we can sniff at any point of time the data transferred on the CAN line.

To do so, we simply start tcpdump:

```
~# tcpdump -i can0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on can0, link-type LINUX_CAN (Linux CAN), capture size 68 bytes
```

Whenever there is any traffic on the line, `tcpdump` will log it to stdout. We will generate some traffic by using the `cansend` command:

```
~# cansend can0 -i 0x12 0x0f 0xf0 0x10 0x01
```

For this data, `tcpdump` will output:

```
00:15:52.482066 CAN Out ID:00000012 PL_LEN:4 PAYLOAD: 0x0f 0xf0 0x10 0x01
```

The log `tcpdump` generates consist of six fields:

1. `00:15:52.482066` is the timestamp this data was on the line. Its format is HH:MM:SS:TTTTTT, with TTTTTT as second's fraction
2. `CAN` interface type
3. `Out` message'S data direction on this interface
4. `ID:00000012` CAN message ID
5. `PL_LEN:4` byte count of message data
6. `PAYLOAD: 0x0f 0xf0 0x10 0x01` the payload data

Some notes:

- The data direction field could be `Out` or `In`
- The CAN message ID encodes some additional info into higher bit values:
  - Bit 31 encodes an extended frame. If this bit is set, an extended message frame was on the line
  - Bit 30 encodes an RTR frame. If this bit is set, a remote transmission message frame was on the line

  The message ID resides in the lower bits of this field
- The PAYLOAD field could be empty, when there were no data elements in the message

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

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*.

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 also can send mails in English.

6.2 News Groups

About Linux in embedded environments

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

comp.os.linux.embedded

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 Miscellaneous

Online Linux Kernel Cross Reference

A powerful online cross reference.

http://lxr.linux.no/

U-Boot manual (partially)

Manual how to survive in an embedded environment and how to use the U-Boot on target's side

http://www.denx.de/wiki/DULG

6.5 phyCORE-i.MX27 Support Maillist

OSELAS.Phytec@pengutronix.de

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

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

6.6 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
Hannoversche Strasse 2
D-31134 Hildesheim
Germany
Phone: +49 - 51 21 / 20 69 17 - 0
Fax: +49 - 51 21 / 20 69 17 - 9

or by electronic mail:

sales@pengutronix.de