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

Quickstart Manual
OSELAS.BSP( )
FriendlyARM mini6410

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

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

info@pengutronix.de

Cut Here and Stick on your Monitor

Don't Panic

Contents

Part I

OSELAS Quickstart for FriendlyARM mini6410

1 First steps with PTXdist

In the next sections you will work with PTXdist to get everything you need to get your mini6410 up and working. To give you a quick idea what PTXdist is, you should read this section.

PTXdist works as a console command tool. Everything we want PTXdist to do, we have to enter as a command. But it's always the same base command:

```
$ ptxdist <parameter>
```

To run different functions, this command must be extended by parameters to define the function we want to run.

If we are unsure what parameter must be given to obtain a special function, we run it with the parameter *help*.

```
$ ptxdist help
```

This will output all possible parameters and subcommands and their meaning.

As the list we see is very long, let's explain the major parameters usually needed for daily usage:

menu This starts a dialog based frontend for those who do not like typing commands. It will gain us access to the most common parameters to configure and build a PTXdist project. Note: it needs 'dialog' to be installed to make is work. It will fail if this tool is not installed on your host. `menuconfig` can be used instead in this case.

menuconfig Starts the Kconfig based project configurator for the current selected userland configuration. This menu will give us access to various userland components the root filesystem of our target should consist of.

platformconfig Starts the Kconfig based platform configurator. This menu lets us set up all target specific settings. Major parts are:

- Toolchain (architecture and revision)
- boot loader
- root filesystem image type
- Linux kernel (revision)

Note: A PTXdist project can consist of more than one platform configuration at the same time.

kernelconfig Runs the standard Linux kernel Kconfig to configure the kernel for the current selected platform. To run this feature, the kernel must be already set up for this platform.

menuconfig barebox Runs the standard Barebox's Kconfig to configure the bootloader. To run this feature, Barebox must be already set up for this platform.

toolchain Sets up the path to the toolchain used to compile the current selected platform. Without an additional parameter, PTXdist tries to guess the toolchain from platform settings. To be successful, PTXdist depends on the OSELAS.Toolchains installed to the `/opt` directory.
If PTXdist wasn't able to autodetect the toolchain, an additional parameter can be given to provide the path to the compiler, assembler, linker and so on.

select Used to select the current userland configuration, which is only required if there is no `selected_ptxconfig` in the project's main directory. This parameter needs the path to a valid `ptxconfig`. It will generate a soft link called `selected_ptxconfig` in the project's main directory.

platform Used to select the current platform configuration, which is only required if there is no `selected_platformconfig` in the project's main directory. This parameter needs the path to a valid `platformconfig`. It will generate a soft link called `selected_platformconfig` in the project's main directory.

go The mostly used command. This will start to build everything to get all the project defined software parts. Also used to rebuild a part after its configuration was changed.

images Used at the end of a build to create an image from all userland packages to deploy the target (its flash for example or its hard disk).

setup Mostly run once per PTXdist revision to set up global paths and the PTXdist behavior.

All these commands depending on various files a PTXdist based project provides. So, running the commands make only sense in directories that contain a PTXdist based project. Otherwise, PTXdist gets confused and then it tries to confuse the user with funny error messages.

2 Getting a working Environment

2.1 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-Pengutronix-Mini6410-2011.11.0, the following archives have to be available on the development host:

- ptxdist-2011.10.1.tar.bz2
- OSELAS.BSP-Pengutronix-Mini6410-2011.11.0.tar.gz
- OSELAS.Toolchain-2011.03.1.tar.bz2

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

2.2 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.

2.2.1 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.

2.2.2 Extracting the Sources

To install PTXdist, the archive Pengutronix provides has to be extracted:

ptxdist-2011.10.1.tar.bz2 The PTXdist software itself

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

```
$ cd
$ mkdir local
$ cd local
```

Next step is to extract the archive:

```
$ tar -xjf ptxdist-2011.10.1.tar.bz2
```

If everything goes well, we now have a PTXdist-2011.10.1 directory, so we can change into it:

```
$ cd ptxdist-2011.10.1
$ ls -lF
total 508
-rw-r--r--   1 jb user  18361 Sep  2 12:40 COPYING
-rw-r--r--   1 jb user   3914 Sep  2 12:40 CREDITS
-rw-r--r--   1 jb user 115540 Sep  2 12:40 ChangeLog
-rw-r--r--   1 jb user     57 Sep  2 12:40 INSTALL
-rw-r--r--   1 jb user   2531 Sep  2 12:40 Makefile.in
-rw-r--r--   1 jb user   4252 Sep  2 12:40 README
-rw-r--r--   1 jb user  63516 Sep  2 12:40 TODO
-rwxr-xr-x   1 jb user     28 Sep  2 12:40 autogen.sh
drwxr-xr-x   2 jb user     72 Sep  2 12:40 bin
drwxr-xr-x  10 jb user    296 Sep  2 12:40 config
-rwxr-xr-x   1 jb user 212213 Sep  2 17:10 configure
-rw-r--r--   1 jb user  12515 Sep  2 12:40 configure.ac
drwxr-xr-x  10 jb user    248 Sep  2 12:40 generic
drwxr-xr-x 221 jb user   7440 Sep  2 12:40 patches
drwxr-xr-x   2 jb user   1240 Sep  2 12:40 platforms
drwxr-xr-x   4 jb user    112 Sep  2 12:40 plugins
drwxr-xr-x   6 jb user  52248 Sep  2 12:40 rules
drwxr-xr-x   8 jb user    912 Sep  2 12:40 scripts
drwxr-xr-x   2 jb user    512 Sep  2 12:40 tests
```

2.2.3 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:

```
$ ./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 2011.10.1 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

```
$ 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*:

```
$ sudo make install
[enter password]
[...]
```

If we don't have root access to the machine it is also possible to install PTXdist 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:

```
$ cd ../../
$ rm -fr local
```

2.2.4 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 archives and if a proxy must be used to gain access to the world wide web.

Run PTXdist's setup:

```
$ ptxdist setup
```

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

Proxy Setup

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

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

Source Archive Location

Whenever PTXdist downloads source archives it stores these archives in a project local manner. This is the default behaviour. If we are working with more than one PTXdist based project, every project would download its own required archives in this case. To share all source archives between all projects, PTXdist can be configured to share 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 its projects.

Generic Project Location

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

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

```
/usr/local/lib/ptxdist-2011.10.1/projects:/office/my_projects/ptxdist
```

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

2.3 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.

2.3.1 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.

Preconditions an external toolchain must meet:

- it shall be built with the configure option `--with-sysroot` pointing to its own C libraries.
- it should not support the *multilib* feature as this may confuse PTXdist which libraries are to select for the root filesystem

If we want to check if our toolchain was built with the `--with-sysroot` option, we just run this simple command:

```
$ mytoolchain-gcc -v 2>&1  grep with-sysroot
```

If this command **does not** output anything, this toolchain was not built with the `--with-sysroot` option and cannot be used with PTXdist.

2.3.2 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.

Building any toolchain of the OSELAS.Toolchain-2011.03.1 is tested with PTXdist-2011.03.0. Pengutronix recommends to use this specific PTXdist to build the toolchain. So, it might be essential to install more than one PTXdist revision to build the toolchain and later on the Board Support Package if the latter one is made for a different PTXdist revision.

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-2011.03.1/`.

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-2011.03.1`
`chown <username> /opt/OSELAS.Toolchain-2011.03.1`
`chmod a+rwx /opt/OSELAS.Toolchain-2011.03.1`.

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

2.3.3 Building the OSELAS.Toolchain for OSELAS.BSP-Pengutronix-Mini6410-2011.11.0

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

The required compiler to build the OSELAS.BSP-Pengutronix-Mini6410-2011.11.0 board support package is

arm-1136jfs-linux-gnueabi_gcc-4.5.2_glibc-2.13_binutils-2.21_kernel-2.6.36-sanitized

So the steps to build this toolchain are:

In order to build any of the OSELAS.Toolchains, the host must provide the tool *fakeroot*. Otherwise the message `bash: fakeroot: command not found` will occur and the build stops.

```
$ tar xf OSELAS.Toolchain-2011.03.1.tar.bz2
$ cd OSELAS.Toolchain-2011.03.1
$ ptxdist select ptxconfigs/↵
       arm-1136jfs-linux-gnueabi_gcc-4.5.2_glibc-2.13_binutils-2.21_kernel-2.6.36-sanitized.ptxconfig
$ 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
- 24 Xeon cores 2.54 GHz, 96 GiB RAM: about 22 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.

2.3.4 Protecting the Toolchain

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

2.3.5 Building Additional Toolchains

The OSELAS.Toolchain-2011.03.1 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 project, removing the current `selected_ptxconfig` link and creating a new one.

```
$ ptxdist clean
$ rm selected_ptxconfig
$ ptxdist select ptxconfigs/any_other_toolchain_def.ptxconfig
$ ptxdist go
```

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

/opt/OSELAS.Toolchain-2011.03.1/architecture_part.

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

/opt/OSELAS.Toolchain-2011.03.1/architecture_part/version_part.

3 Building a root filesystem for the mini6410

3.1 Extracting the Board Support Package

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

```
$ tar -zxf OSELAS.BSP-Pengutronix-Mini6410-2011.11.0.tar.gz
$ cd OSELAS.BSP-Pengutronix-Mini6410-2011.11.0
```

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

```
total 36
-rw-r--r-- 1 jbe ptx 1393 Nov  1 14:25 Changelog
-rw-r--r-- 1 jbe ptx  797 Nov  1 14:25 FAQ
-rw-r--r-- 1 jbe ptx  177 Nov  1 14:25 README
drwxr-xr-x 3 jbe ptx 4096 Nov  1 14:25 configs
drwxr-xr-x 3 jbe ptx 4096 Nov  1 14:25 documentation
drwxr-xr-x 3 jbe ptx 4096 Nov  1 14:25 local_src
drwxr-xr-x 3 jbe ptx 4096 Nov  1 14:25 projectroot
drwxr-xr-x 2 jbe ptx 4096 Nov  1 14:25 protocol
drwxr-xr-x 2 jbe ptx 4096 Nov  1 14:25 rules
```

Notes about some of the files and directories listed above:

ChangeLog Here you can read what has changed in this release.

FAQ Some questions and some answers

documentation This directory contains the Quickstart you are currenly reading in.

configs This directory contains the platform specific configuration files for the Mini6410.

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.

local_src Application sources especially related to the Mini6410.

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.

protocol Contains the test protocol made for the release. Also known open issues if any.

3.2 Selecting a Userland Configuration

First of all we have to select a userland configuration. This step defines what kind of applications will be built for the hardware platform. The OSELAS.BSP-Pengutronix-Mini6410-2011.11.0 comes with a predefined configuration we select in the following step:

```
$ ptxdist select configs/ptxconfig
info: selected ptxconfig:
      'configs/ptxconfig'
```

3.3 Selecting a Hardware Platform

Before we can build this BSP, we need to select one of the possible platforms to build for. In this case we want to build for the mini6410:

```
$ ptxdist platform configs/platform-friendlyarm-mini6410/platformconfig
info: selected platformconfig:
       'configs/platform-friendlyarm-mini6410/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-2011.03/arm-1136jfs-linux-gnueabi/↵
    gcc-4.5.2-glibc-2.13-binutils-2.21-kernel-2.6.36-sanitized/bin'
```

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

3.4 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.

```
$ ptxdist toolchain /opt/OSELAS.Toolchain-2011.03/arm-1136jfs-linux-gnueabi/↵
      gcc-4.5.2-glibc-2.13-binutils-2.21-kernel-2.6.36-sanitized/bin
```

3.5 Building the Root Filesystem

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

```
$ 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 packages and builds them in the correct order.

While the command `ptxdist go` is running we can watch it building all the different stages of a package. In the end the final root filesystem for the target board can be found in the `platform-mini6410/root/` directory and a bunch of *.*ipk* packets in the `platform-mini6410/packages/` directory, containing the single applications the root filesystem consists of.

3.6 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

```
$ 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 barebox/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.
- **root.ubi:** root files inside a ubi volume.

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

```
$ ptxdist platformconfig
```

Then select the submenu “`image creation options`”. The generated image will be placed into `platform-mini6410/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-mini6410/root/` will not be enclosed in the image. If custom files are needed for the target, install them with PTXdist.

4 How to Boot the Mini6410

Various methods are exist to bring up the Mini6410. The main difference is if the Mini6410 can boot in a standalone manner or if it depends on some services from another host via network.

To start the Mini6410 in a standalone manner it provides some local memory types to store the relevant software parts.

- `SD/MMC card memory` is a nice and easy way to deploy the run-time relevant software parts at the development host and simply booting the Mini6410 with it.

4.1 SD/MMC Card Memory

The Mini6410 can boot from an SD/MMC card only. To make it work, all components to boot a target must be present on this card:

- the bootloader. In our case the U-Boot-1.1.6
- the U-Boot environment
- the Linux kernel
- the root filesystem for the kernel

To make all components on the card co-exist, we need some tricks to do so. Main problem here is, the Samsung S3C6410 CPU has no clue about filesystems. Also the used U-Boot does not support filesystems in a regular fashion. This means the bootloader, its environment and the kernel form a binary blob on the disk at specific sectors.

To prevent any filesystem operation later on from clobbering the binary blob on the disk, the binary blob area on disk should not part of any partition on the disk.

It sounds complicated, but it isn't. All required tools are part of this board support package. And all information we need to do it in the right way, are coming from the tools.

After building the images with PTXdist we have several files:

- platform-mini6410/images/u-boot.bin
- platform-mini6410/images/linuximage
- platform-mini6410/images/root.tgz

The first step is to write the binary blob to the disk.

If you follow these steps, all data on your SD card is lost! Ensure to use an empty card or with worthless data.
We also will format this card with a Linux specific filesystem. Do not expect to be able to read the filesystem data from a non Linux system!

All changes on the SD card must be done as user *root*. Best way is to use the sudo command to only do the special commands with root permissions. PTXdist itself rejects to work as root!

My host has no SD card socket, so I'm using a USB based card reader. That's why the devices nodes of the card in the following examples are using SCSI names. Later on on the target the device node name will differ.

In my system the device node for the SD card in the card reader is /dev/sdd, while my system disk is /dev/sda. If you are using autocompletion for file names, take care to not, not!, not!!!11F1!! clobber your system disk! As the shown commands are running as user *root* no warnings will occur!

Assumed here is, the SD card is already inserted into the USB card reader and occurs as /dev/sdd. Replace the /dev/sdd with the name in your setup.

To write the binary blob run:

```
$ sudo platform-mini6410/sysroot-host/bin/mini6410-flasher /dev/sdd -b platform-mini6410/images/u-boot. ↵
    bin -k platform-mini6410/images/linuximage
Writing bootloader part BL1 and BL2... Done
Writing kernelimage... Done
Summary:
------------------------------------------------------------------------------
block dev:                      /dev/sdd
full sector count:              990976
sector size:                    512 bytes
card type:                      SD
 BL1 'platform-mini6410/images/u-boot.bin' 16 sectors beginning at sector 990958
 BL2 'platform-mini6410/images/u-boot.bin' 512 sectors beginning at sector 990190
 Kernel 'platform-mini6410/images/linuximage' 11520 sectors beginning at sector 978670

Take care: Last partition must end in sector 978669!
Ready
```

Note: Samsung calls their ROM based bootloader in the S3C6410 CPU *BL0*. This *BL0* code loads a *BL1*. As the *BL1* can only be up to 8 kiB in size, the *BL1* codes finally loads the *BL2*. And the *BL2* is the full blown bootloader without any restriction in size.

The mini6410-flasher write the binary blob in the specific layout to the disk. And it presents one important number: The sector number the last partition must end on the SD card. In the example shown here this sector number is *978669*. It's the result of using a 512 MiB SD card. We must ensure to note this number for our SD card, as it differs from card to card. We need it in the next step.

The next step is to partion the SD card and to ensure the partitions/filesystems do not conflict with the binary blob area.

In this example only one partition is created. It's sufficient to boot the Mini6410 from the SD card. If you want to create more partitions keep always in mind, the last partition must end at the sector the mini6410-flasher tool gives you!

We run now the tool to partion the SD card and it will 'greet' us:

```
$ sudo /sbin/fdisk /dev/sdd

Command (m for help):
```

First we change to the unit 'sectors' instead cylinders (the default).

```
Command (m for help): u
Changing display/entry units to sectors
```

Using sectors as the unit simplifies partition generation. Now we create the first and only partition on this SD card.

```
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First sector (61-990975, default 61):
Using default value 61
```

We can use the default here. But the next number is important:

```
Last sector, +sectors or +size{K,M,G} (61-990975, default 990975):
```

Here we must enter the number the tool `mini6410-flasher` has given us (adapt it to the number of your SD card):

```
Last sector, +sectors or +size{K,M,G} (61-990975, default 990975): 978669

Command (m for help): p

Disk /dev/sdd: 507 MB, 507379712 bytes
16 heads, 61 sectors/track, 1015 cylinders, total 990976 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
Disk identifier: 0xf3dfa864

   Device Boot      Start         End      Blocks   Id  System
/dev/sdd1              61      978669      489304+  83  Linux
Partition 1 does not end on cylinder boundary.

Command (m for help):
```

We can ignore the warning, since everyone is using sector numbers to read or write from or to a disk like media, no-one is interested in cylinders anymore (beside the fact, an SD card has no cylinders, nor heads).

Last step is to write this new partition table to the SD card:

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

Calling ioctl() to re-read partition table.
Syncing disks.
```

On my system I have now `/dev/sdd` and `/dev/sdd1`. The layout on this SD card is now from the beginning up to sector 978669 used by a filesystem and from sector 978670 to the end used by the binary blob. Both can now co-exist. Using the `mini6410-flasher` again on this SD card (to update the kernel for example) will not destroy the filesystem. And any operation on the filesystem will not destroy the kernel nor the bootloader.

Lets bring in some life into the filesystem area. But first the filesystem itself:

```
$ sudo /sbin/mkfs.ext2 /dev/sdd1
mke2fs 1.41.10 (10-Feb-2009)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
Stride=0 blocks, Stripe width=0 blocks
122400 inodes, 489304 blocks
24465 blocks (5.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=67633152
60 block groups
8192 blocks per group, 8192 fragments per group
2040 inodes per group
Superblock backups stored on blocks:
        8193, 24577, 40961, 57345, 73729, 204801, 221185, 401409

Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 37 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
```

And now its content.

```
$ sudo mount /dev/sdd1 /mnt
$ sudo tar --directory=/mnt -xzf platform-mini6410/images/root.tgz
$ sudo umount /mnt
```

This SD card is now ready to boot the Mini6410.

First we must switch off the Mini6410, insert the prepared SD card and moving the **S2** into the "SDBOOT" position. Then its time to switch on the Mini6410 again and see it booting from the SD card.

BTW: LED3 is configured to signal accesses to the SD card.

5 Special Notes

5.1 Available Kernel Releases

The predifined Mini6410 platform configuration always uses the latest Linux kernel release. If users want to stay with an older Linux kernel release, they are also available. Here is a list of currently available Linux kernel releases in the OSELAS.BSP-Pengutronix-Mini6410-2011.11.0:

- 3.1, (experimental)
- 3.0, stable patch level 8 (default)
- 2.6.39, stable patch level 4

If we want to build the BSP with a non-default kernel release, we just run `ptxdist platformconfig` and change the kernel setting prior to building. As PTXdist checks the MD5 sums of the archives, we also have to change the MD5 sum of the corresponding kernel archive.

Note: The MD5 sums for the kernels are (used by PTXdist):

- 3.1: `8d43453f8159b2332ad410b19d86a931`
- 3.0: `398e95866794def22b12dfbc15ce89c0`
- 2.6.39: `1aab7a741abe08d42e8eccf20de61e05`

5.2 Available Userland Configuration

The Mini6410 BSP comes with two different predefined userland configurations:

- **configs/ptxconfig**: it is the standard one to get a small running embedded system. It can be used as a base for your own development running the Mini6410 headless (without an LCD unit).
- **configs/ptxconfig.qt**: this configuration is intended for graphical usage of the Mini6410. It has the Qt library enabled and brings in a small Qt based application. This application will be started automatically at system's startup, to show how to get a graphical system up and running.

It's up to you and your needs which configuration you may choose in section 3.2.

5.2.1 Some details about the `configs/ptxconfig.qt`

The mentioned small Qt based application we can find in `local_src/qt4-demo-2011.07.0/`. It can act as a template for our own Qt development.

The "secrets" how to build and install this application we can find in `rules/qt4-demo.make` and the corresponding menu file in `rules/qt4-demo.in`.

The "magic" behind the autostart of this small Qt based application at system startup can be found in `projectroot/etc/init.d/startup`.

Note: The small Qt demo is prepared to run on a landsscape 480 x 272 screen. If your screen differs from this setup, don't expect a correct image.

5.3 Framebuffer

This driver gains access to the display via device node `/dev/fb0`. For this BSP the LCDN43-1020 (N43) display with a resolution of 480x240 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 colour depth. Depending on the framebuffer device different kernel command line may be needed to do this. Please refer to your display driver manual for details.

5.4 GPIO

Like most modern System-on-Chip CPUs, the S3C6410 has numerous GPIO pins. Some of them are inaccessible for the userspace, as Linux drivers use them internally. Others are also used by drivers but are exposed to userspace via sysfs. Finally, the remaining GPIOs can be requested for custom use by userspace, also via sysfs.

Refer to the in-kernel documentation `Documentation/gpio.txt` for complete details how to use the sysfs-interface for manually exporting GPIOs.

5.4.1 GPIO Usage Example

When generic architecture GPIO support is enabled in the kernel, some new entries appear in sysfs. Everything is controlled via read and writable files to generate events on the digital lines.

We find all the control files in `/sys/class/gpio`. In that path, there are a number of `gpiochipXXX` entries, with `XXX` being a decimal number. Each of these folders provide information about a single GPIO controller registered on the Mini6410 board, for example with `gpiochip32`:

```
# ls /sys/class/gpio/gpiochip32
base       label      ngpio      subsystem  uevent
```

The entry `base` contains information about the base GPIO number and `ngpio` contains all GPIOs provided by this GPIO controller.
We use `GPIO34` as an example to show the usage of a single GPIO pin.

```
# echo 34 > /sys/class/gpio/export
```

This way we export gpio34 for userspace usage. If the export was successful, we will find a new directory named /sys/class/gpio/gpio34 afterwards. Within this directory we will be able to find the entries to access the functions of this GPIO. If we wish to set the direction and initial level of the GPIO, we can use the command:

```
# echo high > /sys/class/gpio34/direction
```

This way we export GPIO34 for userspace usage and define our GPIO's direction attribute to an output with initially high level. We can change the value or direction of this GPIO by using the entries direction or value.

Note: This method is not very fast, so for quickly changing GPIOs it is still necessary to write a kernel driver. The method shown works well, for example to influence an LED directly from userspace.

To unexport an already exported GPIO, write the corresponding gpio-number into /sys/class/gpio/export.

```
# echo 34 > /sys/class/gpio/unexport
```

Now the directory /sys/class/gpio/gpio34 will disappear.

Note: The GPIO34 is available at connector 6, pin 4 for measurement.

The current usage of most of the pins can be read from the entry in /sys/kernel/debug/gpio.

5.5 I²C Master

The S3C6410 processor based Mini6410 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.

5.5.1 I²C Device AT24c04

This device is a 512 bytes 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/0-0050/eeprom and use fseek() and read() to get the values.

5.6 LEDs

The LEDs on the Mini6410 can be controlled via the LED-subsystem of the Linux kernel. It provides methods for switching them on and off as well as using so-called triggers. For example, you could trigger the LED using a timer. That enables us to make it blink with any frequency we want.

All LEDs can be found in the directory /sys/class/leds. Each one has its own subdirectory. We will use led4 for the following examples.

For each directory, you have a file named brightness which can be read and written with a decimal value between 0 and 255. The first one means LED off, the latter maximum brightness. Inbetween values scale the brightness if the LED supports that. If not, non-zero means just LED on.

```
/sys/class/leds/led4# echo 255 > brightness; # LED on
/sys/class/leds/led4# echo 128 > brightness; # LED at 50% (if supported)
/sys/class/leds/led4# echo 0 > brightness; # LED off
```

LEDs can be connected to triggers. A list of available triggers we can get from the `trigger` entry

```
/sys/class/leds/led4# cat trigger
[none] nand-disk mmc0 mmc1 timer heartbeat backlight gpio
```

The embraced entry is the currectly connected trigger to this LED.

To change the trigger source to the *timer*, just run a:

```
/sys/class/leds/led4# echo timer > trigger
```

If the timer-trigger is activated you should see two additional files in the current directory, namely `delay_on` and `delay_off`. You can read and write decimal values there, which will set the corresponding delay in milliseconds. As an example:

```
/sys/class/leds/led4# echo 250 > delay_on
/sys/class/leds/led4# echo 750 > delay_off
```

will blink the LED being on for 250ms and off for 750 ms.

Replace `timer` with `none` to disable the trigger again. Or select a different one from the list read from the `trigger` entry.

Refer to `Documentation/leds/leds-class.txt` in-kernel documentation for further details about this subsystem.

5.7 MMC/SD Card

The Mini6410 supports *Secure Digital Cards* and *Multi Media Cards* to be used as general purpose blockdevices. These devices can be used in the same way as any other blockdevice.

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

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

```
/dev/mmcblk0pY
```

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

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

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

5.8 SDIO Card

This interface is available at connector CON9. Its feature is untested yet.

5.9 Network

The Mini6410 module has a DM9000 ethernet chip onboard, which is being used to provide the `eth0` network interface. The interface offers a standard Linux network port which can be programmed using the BSD socket interface.

5.10 Touchscreen

A simple test of this feature can be run with:

```
# ts_calibrate
```

to calibrate the touch and with:

```
# ts_test
```

to run a simple application using this feature.

To see the exact events the touch generates, we can also use the `evtest` tool.

```
# evtest /dev/input/event1
Input driver version is 1.0.1
Input device ID: bus 0x19 vendor 0xdead product 0xbeef version 0x102
Input device name: "S3C24XX TouchScreen"
Supported events:
  Event type 0 (Sync)
  Event type 1 (Key)
    Event code 330 (Touch)
  Event type 3 (Absolute)
    Event code 0 (X)
     Value      0
     Min        0
     Max     1023
  Event code 1 (Y)
     Value      0
     Min        0
     Max     1023
Testing ... (interrupt to exit)
```

Whenever we touch the screen this tool lists the values the driver reports.

5.10.1 If the Touchscreen does not work

A functional touchscreen depends on some external configurations and parameters. Firstly, the touchscreen driver for the S3C6410CPU must be enabled in the kernel. If it is supported, it can be checked at run-time with the following command:

```
# ls /sys/bus/platform/drivers
```

A samsung-ts must be listed in this directory. If not, the kernel must be re-configured to support this device.

Secondly, a functional touchscreen depends on is the registered touchscreen device. If it is registered, this can be checked at run-time with this command:

```
# ls /sys/bus/platform/devices
```

A samsung-ts must be listed in this directory. If not, something is preventing the kernel from registering this device.

5.10.2 If the Touchscreen does not work as expected

It's not easy to create a touchscreen driver that works with all kinds of touchscreens. They differ in their hardware parameters, so most of the time some adaptions must be done to get better results.

Two locations exists where parameters can be changed:

- in the kernel driver
- in the *tslib* (**t**ouch**s**creen **lib**rary)

The *tslib* is a userland component and can be changed at any time. The kernel driver is a compiled in component, so the kernel must be re-built and re-started to make any change visible.

Lets start with the kernel driver: It uses three parameters to support the physical behaviour of the touchscreen.

- **.delay** a delay counted in clocks of 3.68 MHz between the measurement of the X and Y coordinate. If the touchscreen lines are filtered with a low-pass it could be useful to increase this value. Max value is oxffff.
- **.presc** clock prescaler for the AD converter. The larger the value is, the lower the AD converter works (FIXME: Seems not be used)
- **.oversampling_shift** defines the samples to be measured and to be averaged before reporting a coordinate. '0' means one sample per report, '1' means two samples per report, '2' means 4 samples and so on.

To modify the setting, open the file platform-mini6410//build-target/linux-3.0/arch/arm/mach-s3c64xx/mach-mini6410.c and search for the s3c_ts_platform structure. It looks like this:

```
static struct s3c2410_ts_mach_info s3c_ts_platform __initdata = {
    .delay = 10000,
    .presc = 49,
    .oversampling_shift = 2,
};
```

After modifying, the kernel must be re-built:

```
$ ptxdist drop kernel compile
$ ptxdist go
```

These steps ensure the modified sources are re-compiled. Use this new kernel and do the tests with the touchscreen again.

To change the userland *tslib* this can be done at run-time of the Mini6410. Just modify the /etc/ts.conf.

- **module_raw input** means the tslib uses the raw data from the Linux's input system
- **module pthres pmin=1** means the minimal pressure must be '1' to count as a touchscreen event. Other values do not make sense yet, as the driver does not support pressure measurement
- **module variance delta=30** FIXME
- **module dejitter delta=10** FIXME
- **module linear** FIXME

After changing one of these entries a run of `ts_test` can show if the new settings are better than the previous ones.

Some display units with touchscreen support for the Mini6410 are using a OneWire interface to control the touchscreen. These kind of display units are not supported by this board support package.

5.11 LCD Backlight

The backlight of the LCD can be controlled via the sysfs entry in:

```
/sys/class/leds/backlight/
```

To switch it *off*, just write a '0' into its `brightness` entry:

```
# echo 0 > /sys/class/leds/backlight/brightness
```

and a '1' to switch it *on* again:

```
# echo 1 > /sys/class/leds/backlight/brightness
```

5.12 ADC

Getting the digital equivalent of one of the analogue input channels can be done by reading the corresponding entries in the sysfs.

For example the analogue input channel 0 on the Mini6410 is connected to the potentiometer W1. By reading the entry `/sys/devices/platform/s3c64xx-adc/s3c-hwmon/in0_input` we can watch the different digital values while turning the potentiometer W1.

Note: The analogue input channels 4 ... 7 are occupied by the touchscreen feature and can only be used as simple analogue inputs if the touchscreen feature is disabled.

5.13 Keypad

Using the up to 6 available key buttons on the Mini6410 in a regular manner requires a working console in the kernel. Here the list of the current key codes they generate when pressed:

- K1, code 'F1'
- K2, code 'F2'
- K3, code 'F3'
- K4, code 'F4'
- K5, code 'F5'
- K6, code 'F6'
- K7, code 'F7'
- K8, code 'F8'

If one wants to change the generated codes, she/he can change it in the platform code found in `arch/arm/mach-s3c64xx/mach-mini6410.c`, specially in the array `mini6410_buttons`.

If the key buttons are working as expected, can also be checked without a working console with the following command:

```
# evtest /dev/input/event0
Input driver version is 1.0.1
Input device ID: bus 0x19 vendor 0x1 product 0x1 version 0x100
Input device name: "gpio-keys"
Supported events:
  Event type 0 (Sync)
  Event type 1 (Key)
   Event code 59 (F1)
    Event code 60 (F2)
    Event code 61 (F3)
    Event code 62 (F4)
    Event code 63 (F5)
    Event code 64 (F6)
    Event code 65 (F7)
    Event code 66 (F8)
Testing ... (interrupt to exit)
```

5.14 Get the latest BSP Release for the Mini6410

Information and the latest release of the Mini6410 BSP can be found on our website at:

http://www.oselas.org/oselas/bsp/index_en.html

5.15 Be Part of the Mini6410 BSP Development

If you want to use the latest and greatest board support package for the Mini6410 you can use the git repository as your working source, instead of a release archive.

The git repository can be found here:

```
http://git-public.pengutronix.de/git-public/OSELAS.BSP-Pengutronix-Mini6410.git
```

If you want to contribute to this project by sending patches, these patches should always be based on the **master** branch of this repository.

6 Document Revisions

2011/07/12	Initial Revision
2011/10/30	Add the how to boot the Mini6410 from SD card
2011/10/30	Add the special notes how to use some features of the Mini6410
	×

7 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.

7.1 Mailing Lists

7.1.1 About PTXdist in Particular

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

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

on 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.

7.1.2 About Embedded Linux in General

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

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

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

7.2 About Working on the Linux Kernel

The book *Linux Kernel in a Nutshell* from *Greg Kroah-Hartman*. Its online version can be read here:

http://www.kroah.com/lkn/

7.3 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.

7.4 FriendlyARM mini6410 specific Mailing List

oselas@community.pengutronix.de

This is a community mailing list open for everyone for all mini6410's board support package related questions. Refer our page at

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

to subscribe to this mailing list.

Note: Please be aware that we cannot answer hardware only related questions on this list.

7.5 Commercial Support

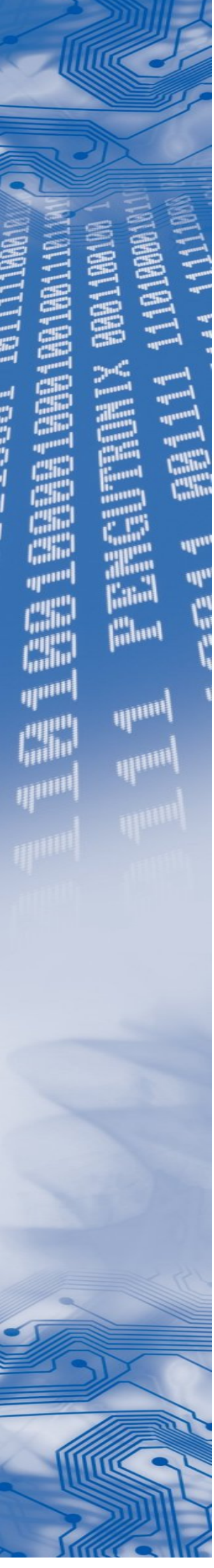
You can order immediate support through customer specific mailing lists, by telephone or also on site. Ask our sales representative for a price quotation for your special requirements.

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



This is a Pengutronix Quickstart Manual

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