OSELAS.BSP( )
Phytec phyCORE-i.MX27

Quick Start Manual

http://www.oselas.com

$Rev : 851\ Date : 2008-03-1417:29:26+0100(Fri, 14Mar2008)$

Contents

I OSELAS Quickstart for Phytec phyCORE-i.MX27 3

1 Getting a working Environment 4
- 1.1 Download Parts . . . 4
- 1.2 PTXdist Installation . . . 4
- 1.3 Toolchains . . . 7

2 Accessing Peripherals 10
- 2.1 NOR Flash . . . 10
- 2.2 Serial TTYs . . . 11
- 2.3 Network . . . 11
- 2.4 SPI . . . 11
- 2.5 Framebuffer . . . 11
- 2.6 Touch . . . 11
- 2.7 I²C Master . . . 12
- 2.8 CPU core frequency scaling . . . 12

3 Getting help 13
- 3.1 Mailing Lists . . . 13
- 3.2 News Groups . . . 13
- 3.3 Chat/IRC . . . 13
- 3.4 Miscellaneous . . . 14
- 3.5 phyCORE-i.MX27 specific Maillist . . . 14
- 3.6 Commercial Support . . . 14

Part I

OSELAS Quickstart for Phytec phyCORE-i.MX27

1 Getting a working Environment

1.1 Download Parts

In order to follow this manual, some software archives are needed. There are several possibilities how to get these: either as part of an evaulation board package, or by download from a world wide 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.MX27-4, the following archives should be available on the development host:

- ptxdist-1.0.1.tgz
- ptxdist-1.0.1-patches.tgz
- OSELAS.BSP-Phytec-phyCORE-i.MX27-4.tar.gz
- OSELAS.Toolchain-1.1.1.tar.bz2

1.2 PTXdist Installation

PTXdist is shipped divided into several archives. This chapter provides information about how to install and configure PTXdist on the development host to get a working environment to build root filesystems for target systems.

1.2.1 Building Blocks

The main tool of the OSELAS.BoardSupport() Package is PTXdist. 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_de.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-1.0.1.tgz The PTXdist software itself.

ptxdist-1.0.1-patches.tgz All patches against upstream software packets (known as the 'patch repository').

ptxdist-1.0.1-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-1.0.1.tgz
~/local$ tar -zxf ptxdist-1.0.1-patches.tgz
```

and if required the generic projects:

```
~/local$ tar -zxf ptxdist-1.0.1-projects.tgz
```

If everything goes well, we now have a PTXdist-1.0.1 directory, so we can change into it:

```
~/local$ cd ptxdist-1.0.1
~/local/ptxdist-1.0.1$ ls -l

total 429
drwxr-xr-x   2 jbe users   1024 2007-10-19 23:24 autoconf/
-rwxr-xr-x   1 jbe users     28 2007-10-19 23:20 autogen.sh
drwxr-xr-x   2 jbe users   1024 2007-10-19 23:24 bin/
-rw-r--r--   1 jbe users 121724 2007-10-19 23:20 ChangeLog
drwxr-xr-x   8 jbe users   1024 2007-10-19 23:24 config/
-rwxr-xr-x   1 jbe users 204310 2007-10-19 23:24 configure
-rw-r--r--   1 jbe users   9755 2007-10-19 23:20 configure.ac
-rw-r--r--   1 jbe users  18361 2007-10-19 23:20 COPYING
-rw-r--r--   1 jbe users   2792 2007-10-19 23:20 CREDITS
drwxr-xr-x   2 jbe users   1024 2007-10-19 23:24 debian/
drwxr-xr-x   2 jbe users   1024 2007-10-19 23:24 Documentation/
drwxr-xr-x   7 jbe users   1024 2007-10-19 23:24 generic/
-rw-r--r--   1 jbe users     58 2007-10-19 23:20 INSTALL
-rw-r--r--   1 jbe users   2686 2007-10-19 23:20 Makefile.in
drwxr-xr-x 132 jbe users   4096 2007-10-19 23:24 patches/
drwxr-xr-x   5 jbe users   1024 2007-10-19 23:22 projects/
-rw-r--r--   1 jbe users   3866 2007-10-19 23:20 README
-rw-r--r--   1 jbe users    691 2007-10-19 23:20 REVISION_POLICY
drwxr-xr-x   6 jbe users  24576 2007-10-19 23:24 rules/
drwxr-xr-x   6 jbe users   1024 2007-10-19 23:24 scripts/
drwxr-xr-x   2 jbe users   1024 2007-10-19 23:24 tests/
-rw-r--r--   1 jbe users  28468 2007-10-19 23:20 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-1.0.1$ ./configure
```

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

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

ptxdist version 1.0.1 configured.
Using '/usr/local' for installation prefix.

Report bugs to ptxdist@pengutronix.de

~/local/ptxdist-1.0.1$
```

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.

In this early PTXdist version not all tests are implemented in the `configure` script yet. So if something goes wrong or you don't understand some error messages send a mail to `support@pengutronix.de` and help us improve the tool.

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

```
~/local/ptxdist-1.0.1$ 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 root:

```
~/local/ptxdist-1.0.1$ su
[enter root password]
/home/username/local/ptxdist-1.0.1$ make install
[...]
```

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-1.0.1$ cd
~$ rm -fr local/ptxdist-1.0.1
```

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-1.0.1/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-1.0.1 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-1.1.1/`.

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-1.1.1 ; chown <username> /opt/OSELAS.Toolchain-1.1.1; chmod a+rwx /opt/OSELAS.Toolchain-1.1.1`).

1.3.3.1 Building the OSELAS.Toolchain for OSELAS.BSP-Phytec-phyCORE-i.MX27-4

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.MX27-4 board support package is

`arm-v4t-linux-gnueabi_gcc-4.1.2_glibc-2.5_linux-2.6.18.ptxconfig`.

So the steps to build this toolchain are:

```
~$ tar xf OSELAS.Toolchain-1.1.1.tar.bz2
~$ cd OSELAS.Toolchain-1.1.1
~/OSELAS.Toolchain-1.1.1$ ptxdist select
  ptxconfigs/arm-v4t-linux-gnueabi_gcc-4.1.2_glibc-2.5_linux-2.6.18.ptxconfig.ptxconfig
~/OSELAS.Toolchain-1.1.1$ 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 rights 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 changing recursivley 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-1.1.1 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 `ptxconfig` link and creating a new one.

```
~/OSELAS.Toolchain-1.1.1$ ptxdist clean
~/OSELAS.Toolchain-1.1.1$ rm ptxconfig
~/OSELAS.Toolchain-1.1.1$ ptxdist select
       ptxconfigs/any_another_toolchain_def.ptxconfig
~/OSELAS.Toolchain-1.1.1$ ptxdist go
```

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

`/opt/OSELAS.Toolchain-1.1.1/architecture_part`.

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

`/opt/OSELAS.Toolchain-1.1.1/architecture_part/version_part`.

2 Accessing Peripherals

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 (PCM-970).

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.

2.1 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.MX27-4 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.

2.2 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

2.3 Network

The i.MX27 CPU embedds an Fast Ethernet Controller (FEC) onchip, which is being used to provide the eth0 network interface. The interface offers a standard Linux network port which can be programmed using the BSD socket interface.

2.4 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.

2.5 Framebuffer

This feature is for test purposes only. Not yet for production usage.

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

A simple test of this feature can be run with:

```
~# fbtest
```

This will show various pictures on the display.

2.6 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.

2.7 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.

2.7.1 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.

2.7.2 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.

2.7.3 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.

2.8 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) 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/`.

3 Getting help

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

3.1 Mailing Lists

About PTXdist in special

This is an english language public mailing list for questions about PTXdist. See web site

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 web site

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

how to subscribe to this list. Note: You also can send english language mails.

3.2 News Groups

About Linux in embedded environments

This is an english language news group for general questions about Linux in embedded environments.

comp.os.linux.embedded

About general Unix/Linux questions

This is a german language news group for general questions about Unix/Linux programming.

de.comp.os.unix.programming

3.3 Chat/IRC

About PTXdist in special

irc.freenode.net:6667

Create a connection to the **irc.freenode.net:6667** server and enter the chat group **#ptxdist**. This is an english language group to answer questions about PTXdist. Best time to meet somebody in there is at europeen daytime.

3.4 Miscellaneous

Online Linux Kernel Cross Reference

A powerful cross reference to be used online.

http://lxr.linux.no/blurb.html

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

3.5 phyCORE-i.MX27 specific 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

how to subscribe to this list.

3.6 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