BSP-Quickstart phyCORE-AM335x

In this manual copyrighted products are not explicitly indicated. The absence of the trademark (™) and copyright (©) symbols does not imply that a product is not protected. Additionally, registered patents and trademarks are similarly not expressly indicated in this manual.

The information in this document has been carefully checked and is believed to be entirely reliable. However, PHYTEC Messtechnik GmbH assumes no responsibility for any inaccuracies. PHYTEC Messtechnik GmbH neither gives any guarantee nor accepts any liability whatsoever for consequential damages resulting from the use of this manual or its associated product. PHYTEC Messtechnik GmbH reserves the right to alter the information contained herein without prior notification and accepts no responsibility for any damages that might result.

Additionally, PHYTEC Messtechnik GmbH offers no guarantee nor accepts any liability for damages arising from the improper usage or improper installation of the hardware or software. PHYTEC Messtechnik GmbH further reserves the right to alter the layout and/or design of the hardware without prior notification and accepts no liability for doing so.

	EUROPE	NORTH AMERICA
Address:	PHYTEC Technologie Holding AG Robert-Koch-Str. 39 55129 Mainz GERMANY	PHYTEC America LLC 203 Parfitt Way SW, Suite G100 Bainbridge Island, WA 98110 USA
Ordering Information:	+49 (6131) 9221-32 sales@phytec.de	1 (800) 278-9913 sales@phytec.com
Technical Support:	+49 (6131) 9221-31 support@phytec.de	1 (800) 278-9913 support@phytec.com
Fax:	+49 (6131) 9221-33	1 (206) 780-9135
Web Site:	http://www.phytec.de	http://www.phytec.com

3rd Edition: September 2013

Chapter 1 The Environment 5
1.1 Software Components 5
1.2 PTXdist 6
1.2.1 Main Parts of PTXdist 6
1.2.2 Extracting the sources 6
1.2.3 PTXdist installation 7
1.2.4 Configuring PTXdist 9
1.3 Toolchains 11
1.3.1 Building the Toolchain 12
1.3.2 Protecting the Toolchain 14
1.3.3 Building additional Toolchains 15
Chapter 2 Building phyCORE-AM335x's BSP 16
2.1 The Board Support Package 16
2.2 Selecting a Hardware Platform 16
2.3 Selecting a Toolchain 17
2.4 Building the Linux Kernel and its Root Filesystem 17
2.5 Building an Root Filesystem Image 18
Chapter 3 phyCORE-AM335x Bootloader preparation . 20
3.1 Updating Barebox 20
3.2 Updating using SD-Card instead of LAN 21
3.3 Booting from SPI NOR flash 23
Chapter 4 Booting Linux 24
4.1 Development Host Preparations 25
4.2 Stand-Alone Booting Linux 25
4.2.1 Preparations on the Embedded Board ... 26
4.2.2 Booting the Embedded Board 27
4.2.3 Using SD-Card instead of LAN 28
4.3 Remote-Booting Linux 28
4.3.1 Development Host Preparations 29
4.3.2 Booting the Embedded Board 29

Chapter 5 Accessing Peripherals 30
5.1 NAND Flash 31
5.2 Serial TTYs 32
5.3 Network 32
5.4 SPI Master 33
5.5 I²C Master 33
5.6 USB Host Controller 34
5.7 USB OTG 34
5.8 MMC/SD Card 35
5.9 GPIO 36
5.10 CAN Bus 36
5.11 Framebuffer 38
5.12 Touch 39
5.13 AUDIO support 39
5.13.1 Audio Sources and Sinks 40
5.13.2 Playback 41
5.13.3 Capture 41
5.14 Using Qt 41
5.15 CPU core frequency scaling 42
Chapter 6 Getting help 45
6.1 Mailing Lists 45
6.1.1 About PTXdist in Particular 45
6.1.2 About Embedded Linux in General 45
6.2 News Groups 46
6.2.1 About Linux in Embedded Environments 46
6.2.2 About General Unix/Linux Questions 46
6.3 Chat/IRC 46
6.4 phyCORE-AM335x Support 46
6.5 Commercial Support 47

Chapter 1 The Environment

1.1 Software Components

In order to follow this manual, some software archives are needed: BSP, toolchain, PTXdist, examples and so on. You should use 32-Bit Ubuntu 12.04 LTS and at first install the following packages:

```
sudo apt-get install libncurses5-dev gawk flex bison
sudo apt-get install texinfo quilt autoconf
```

Generally the central place for our BSPs is our ftp-server ftp://ftp.phytec.de/pub/Products/phyCORE-AM335x. In order to build a BSP you need the appropriate toolchain and you need the build tool PTXdist from our partner Pengutronix. The central place for toolchains is http://www.oselas.com and for PTXdist it is http://www.ptxdist.de. These websites provide all required packages and documentation (at least for software components that are available to the public). Usually you can find a copy of the particular needed PTXdist and toolchain together with the BSP on our ftp-server in the same directory, in order to make things easier.

To build BSP-phyCORE-AM335x-PD13.1.0, the following archives have to be available on the development host:

- ptxdist-2013.01.0.tar.bz2
- OSELAS.Toolchain-2012.12.1.tar.bz2
- phyCORE-AM335x-PD13.1.0.tar.gz

1.2 PTXdist

The most important software component which is necessary to build a BSP (board support package) is the PTXdist tool. The PTXdist build system must be used to create all images for our embedded devices based on Linux. In order to start development with PTXdist it is necessary that the software has been installed on the development system.

1.2.1 Main Parts of PTXdist

The PTXdist Program: *ptxdist* is called to trigger any action, like building a software packet, cleaning up the tree etc. Usually the *ptxdist* program is used in a workspace directory, which contains all project relevant files.

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

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

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

1.2.2 Extracting the sources

To install PTXdist you need to extract the archive with the PTXdist software *ptxdist-2013.01.0.tar.bz2*.

The PTXdist packet is 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-2013.01.0.tar.bz2
```

If everything goes well, we now have a ptxdist-2013.01.0 directory, so we can change into it:

```
cd ptxdist-2013.01.0
```

1.2.3 PTXdist installation

Before PTXdist can be installed it has to be checked if all necessary programs such as *quilt* and *wget* are installed on the development host. The *configure* script will stop if it discovers that something is missing.

The PTXdist installation is based on GNU autotools, so the first thing to be done now is to configure the packet:

```
./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 2013.01.0 configured.
Using '/usr/local' for installation prefix.
Report bugs to ptxdist@pengutronix.de
```

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

The installation paths are configured in a way that several PTXdist versions can be installed in parallel. So if an old version of PTXdist is already installed there is no need to remove it. Later you will call the current version of PTXdist with command *ptxdist* and all other versions with command *ptxdist-<version>* with *<version>* set to the version you want to use.

Note that every BSP asks for a dedicated version of PTXdist. It may cause much work to try to build a BSP with a newer version of PTXdist than it requires.

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

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

```
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 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:

```
cd
rm -rf local
```

1.2.4 Configuring PTXdist

When using PTXdist for the first time, some setup properties have to be configured. Two settings are the most important ones: Where to store the source packages and if a proxy must be used to gain access to the world wide web.

Run PTXdist’s setup:

```
ptxdist setup
```

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

Proxy Setup

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

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

Source Archive Location

Whenever PTXdist downloads source archives it stores these archives in a project local manner. If we are working with more than one project, every project would download its own required archives. To share all source archives between all projects PTXdist can be configured to use only one archive directory for all projects it handles: Navigate to menu entry Source Directory and enter the path to the directory where PTXdist should store archives to share between projects.

1.3 Toolchains

Before we can start building our first userland we need a cross toolchain. On Linux, toolchains are no monolithic beasts. Most parts of what we need to cross compile code for the embedded target comes from the GNU Compiler Collection, *gcc*. The *gcc* packet includes the compiler frontend, *gcc*, plus several backend tools (*cc1*, *g++*, *ld* etc.) which actually perform the different stages of the compile process. *gcc* does not contain the assembler, so we also need the GNU Binutils package which provides lowlevel stuff.

Cross compilers and tools are usually named like the corresponding host tool, but with a prefix – the GNU target. For example, the cross compilers for ARM and powerpc may look like

- `arm-cortexa8-linux-gnu-gcc`
- `powerpc-unknown-linux-gnu-gcc`

With these compiler frontends we can convert e.g. a C program into binary code for specific machines. So for example if a C program is to be compiled natively, it works like this:

```
gcc test.c -o test
```

To build the same binary for the ARM architecture we have to use the cross compiler instead of the native one:

```
arm-cortexa8-linux-gnu-gcc test.c -o test
```

Also part of what we consider to be the ”toolchain” is the runtime library (*libc*, dynamic linker). All programs running on the embedded system are linked against the *libc*, which also offers the interface from user space functions to the kernel.

The compiler and *libc* are very tightly coupled components: The second stage compiler, which is used to build normal user space code, is being built against the *libc* itself. For example, if the target does not contain a hardware floating point unit, but the toolchain generates floating point code, it will fail. This is also the case when the toolchain builds code for i686 CPUs, whereas the target is i586.

So in order to make things working consistently it is necessary that the runtime *libc* is identical with the *libc* the compiler was built against.

PTXdist doesn’t contain a pre-built binary toolchain. Remember that it’s not a distribution but a development tool. But it can be used to build a toolchain for our target. Building the toolchain usually has only to be done once. It may be a good idea to do that over night, because it may take several hours, depending on the target architecture and development host power.

1.3.1 Building the Toolchain

If a toolchain is already installed which is known to be working, the toolchain building step with PTXdist may be omitted.

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 BSP.

A PTXdist project generally allows to build into some project defined directory. OSELAS.Toolchain projects that come with PTXdist are configured to install into */opt*.

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-2012.12.1
chown <username> /opt/OSELAS.Toolchain-2012.12.1
chmod a+rwx /opt/OSELAS.Toolchain-2012.12.1
```

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 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.

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. Please be sure that you take the hard float version (hf) in order to get best performance of the BSP later on:

```
tar -xjf OSELAS.Toolchain-2012.12.1.tar.bz2
cd OSELAS.Toolchain-2012.12.1
ptxdist select ptxconfigs/\     [Enter]
>         arm-cortexa8-linux-gnueabihf_gcc-4.7.3_glibc-2.16.0_binutils-
2.22_kernel-3.6-sanitized.ptxconfig
ptxdist --force go
```

Usually you would enter:

```
ptxdist go
```

But OSELAS.Toolchain-2012.12.1 needs ptxdist-2012.12.0 instead of ptxdist-2013.01.0 und thus this call would fail due to version conflicts.

In this special case using *--force* works and spares you the need to work with another version of ptxdist than later being needed by for building the BSP. But please note that in most case it is not a good idea to try builds with other versions than recommended.

On reasonably fast machines the time to build an OSELAS.Toolchain is something like around 30 minutes up to a few hours.

When the OSELAS.Toolchain project build is finished, PTXdist is ready for prime time and we can continue with our first project.

1.3.2 Protecting the Toolchain

All toolchain components are being 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.

1.3.3 Building additional Toolchains

The OSELAS.Toolchain-bundle comes with various predefined toolchains. Refer to 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 \     [Enter]
> ptxconfigs/any_other_toolchain_def.ptxconfig
ptxdist go
```

Chapter 2 Building phyCORE-AM335x's BSP

2.1 The Board Support Package

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

```
tar -zxf phyCORE-AM335x-PD13.1.0.tar.gz
cd phyCORE-AM335x-PD13.1.0
```

Some of the important components of the BSP that you will find here are:

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.

2.2 Selecting a Hardware Platform

Before we can build this BSP, we need to select the target to build for. In this case we want to build for the phyCORE-AM335x, so please type:

```
ptxdist platform configs/phyCORE-AM335x/platformconfig
```

You will see:

```
info: selected platformconfig:
'configs/phyCORE-AM335x/platformconfig'
```

If PTXdist automatically detects the proper toolchain while selecting the platform (does not work with alpha release), it will also output:

```
found and using toolchain:
'/opt/OSELAS.Toolchain-2012.12.1/arm-cortexa8-linux-gnueabihf/
gcc-4.7.3-glibc-2.16.0-binutils-2.22-kernel-3.6-sanitized/bin'
```

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

2.3 Selecting a Toolchain

If not automatically detected, one more step in selecting various configurations is to select the toolchain to be used to build everything for the target.

```
ptxdist toolchain /opt/OSELAS.Toolchain-2012.12.1/\     [Enter]
arm-cortexa8-linux-gnueabihf/\     [Enter]
gcc-4.7.3-glibc-2.16.0-binutils-2.22-kernel-3.6-sanitized/bin
```

2.4 Building the Linux Kernel and its Root Filesystem

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

```
ptxdist go
```

PTXdist does now automatically find out from the *selected_ptxconfig* and *selected_platformconfig* files which packages belong to the project and starts compiling their targetinstall stages (the linux kernel and those that actually put compiled binaries into the root filesystem). While doing this, PTXdist finds out about all the dependencies between the packets and brings them into the correct order.

While the command *ptxdist go* is running we can watch it building all the different stages of a packet. In the end the linux kernel can be found in *platform-phyCORE-AM335x/images/* directory and the final root filesystem for the target board can be found in the *platform-phyCORE-AM335x/root/* directory and a bunch of **.ipk* packets in the *platform-phyCORE-AM335x/packages/* directory, containing the single applications the root filesystem consists of.

Sometimes it can happen that an attempt to download a source package failes due to dead links. PHYTEC cannot guarantee the accessibility of all the webpages that are needed to build a BSP. You can check our ftp-server via

ftp://ftp.phytec.de/pub/BSP_PACKAGES/EXTERNALS

and then download it from there into your src-directory manually. After that, please restart build process by calling `ptxdist go` again. If you also cannot find it here, please tell us via support@phytec.de. We will then take it from our build-server and upload it onto our ftp-server for you.

Sometimes it might even happen that ptxdist can download a file that is claimed to be an archive, but then cannot decompress it. Reason is that the link didn't deliver the requested archive, but an HTML containing a message like: "Error - This file has been moved to ...". Please treat this case in the same way as if nothing had been delivered by the link.

2.5 Building an Root Filesystem Image

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

```
ptxdist images
```

PTXdist will then extract the content of priorly created *.*ipk* packages to a temporary directory and generate an image out of it. PTXdist supports several image types. What you need is:

• root.tgz: root files inside a plain gzip compressed tar ball.

• root.ubifs: root files inside an UBI filesystem.

• root.ubi: This is the physical UBI image to be flashed into the NAND.

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-phyCORE-AM335x/images/*.

Only the content of the *.*ipk* packages will be used to generate the image. This means that files which are put manually into the *platform-phyCORE-AM335x/root/* will not be enclosed in the image.

Chapter 3 phyCORE-AM335x Bootloader preparation

This step can be omitted if your phyCORE-AM335x already has the right boot loaders: For this BSP the Barebox version v2013.07.0 is used as second stage and as third stage boot loader.

3.1 Updating Barebox

Build the whole BSP with *ptxdist go* or just build the third stage boot loader with *ptxdist targetinstall barebox* and the second stage boot loader with *ptxdist targetinstall barebox_mlo*. When it's built go to directory *platform-phyCORE-AM335x/images* and copy the generated files *barebox-image* and *MLO* to your configured tftp exported directory. Then rename *barebox-image* to *barebox.bin* there.

On the target side first check for the correct network settings. Connect to the target with your favorite terminal application. After connecting the board with the power supply, the target starts booting. Press any key to stop autoboot, then type:

```
ifup eth0
devinfo eth0
```

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

```
ipaddr=192.168.3.11
netmask=255.255.255.0
gateway=192.168.3.10
serverip=192.168.3.10
```

If you need to change something, type

```
edit /env/network/eth0
```

Edit the settings, save them by leaving the editor with Strg-D, then type *saveenv* and reboot the board. Otherwise leave the editor with Strg-C.

Now get the new Barebox from your tftp-server into the module's RAM:

```
tftp barebox.bin
```

You also could use:

```
cp /mnt/tftp/barebox.bin .
```

It does automatically mount, so in this case no `ifup eth0` is needed.

After that store the Barebox into the NAND flash:

```
erase /dev/nand0.barebox.bb
cp barebox.bin /dev/nand0.barebox.bb
```

Note that the command `update` known from former BSP versions is not available anymore.

For the second stage bootloader the commands are:

```
cp /mnt/tftp/MLO .
erase /dev/nand0.xload.bb
cp MLO /dev/nand0.xload.bb
```

Note that if something goes wrong at this, you don't have any bootloader anymore on your module. In this case you need to boot from an SD-card as being described in the following chapter.

3.2 Updating using SD-Card instead of LAN

Alternatively of using LAN you can also update the images using an SD-card. Mounting it will be done automatically when you boot from it.

In order to boot from SD-card, set DIP switch S5 as follows:

```
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
-------------------------------
|on |on |on |on |off|on |off|off|
```

A description of how to create a bootable SD-card and a script that is needed therefor can be found on http://www.phytec.eu under *Support* / *FAQ/Download* / *phyCORE-AM335x*. But please note that the Barebox image must be named *barebox.bin* now, not *barebox.img* anymore as for earlier BSP versions.

You could also mount manually by commands:

```
mci0.probe=1
mkdir /mnt/disk
mount /dev/disk0.0 /mnt/disk
```

Update x-loader *MLO* by typing:

```
erase /dev/nand0.xload.bb
cp /mnt/disk/MLO /dev/nand0.xload.bb
```

Update barebox by typing:

```
erase /dev/nand0.barebox.bb
cp /mnt/disk/barebox.bin /dev/nand0.barebox.bb
```

3.3 Booting from SPI NOR flash

The *MLO* for the SPI NOR flash has to be byte-swapped. So if you want to boot from SPI NOR flash and thus need to flash *MLO* into it, this will be done like this:

```
ifup eth0
tftp MLO
erase /dev/m25p0.xload
barebox_update -t MLO.spi MLO
```

After that use the following commands in order to flash the barebox:

```
tftp barebox.bin
erase /dev/m25p0.barebox
cp barebox.bin /dev/m25p0.barebox
```

In order to boot from SPI NOR flash, set DIP switch S5 as follows:

```
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
---------------------------------
|on |on |off|off|on |on |off|off|
```

Chapter 4 Booting Linux

Now that there is a linux kernel and a root filesystem in our workspace we'll have to make them visible to the phyCORE-AM335x. There are two possibilities to do this:

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

2. Booting from the development host via network.

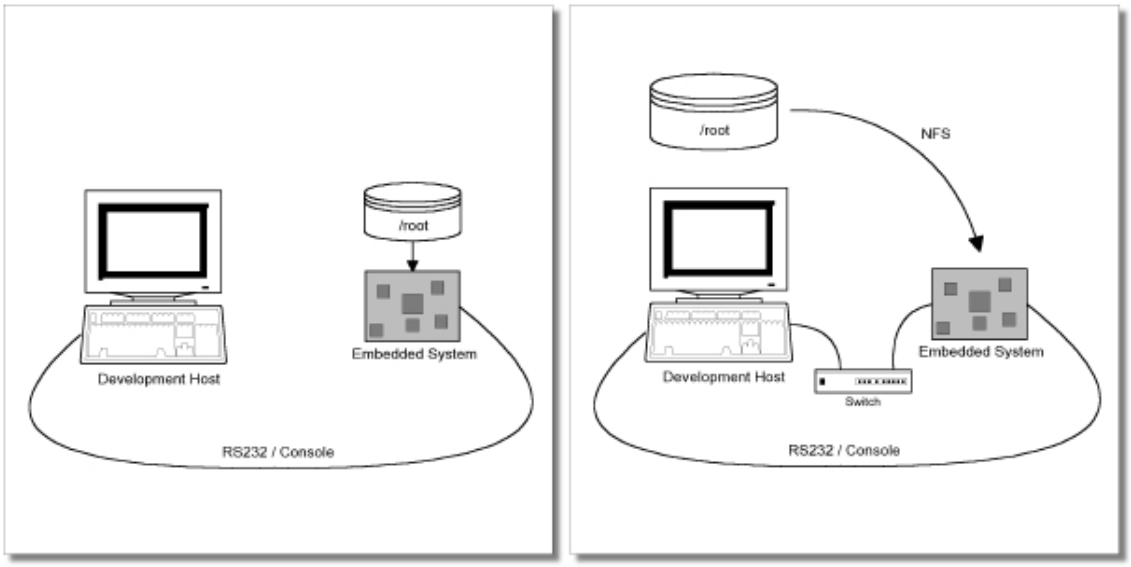

Figure 4.1: Booting the phyCORE-AM335x: From its flash or from the host via network.

Figure 4.1 shows both methods. The main method used in the BSP-phyCORE-AM335x-PD13.1.0 BSP is to provide all needed components to run on the target itself. The linux kernel image and the root filesystem image are persistent in the media the target features. This means the only connection needed is the nullmodem cable to see what is happening on our target. We call this method *standalone*.

The other method is to provide needed components via network. In this case the development host is connected to the phyCORE-AM335x with a

serial nullmodem cable and via ethernet; the embedded board boots into the bootloader, then issues a TFTP request on the network and boots the kernel from the TFTP server on the host. Then, after decompressing the kernel into the RAM and starting it, the kernel mounts its root filesystem via NFS (Network File System) from the original location of the *platform-phyCORE-AM335x/root/* directory in our PTXdist workspace.

The BSP-phyCORE-AM335x-PD13.1.0 provides both methods. The latter one is especially for development purposes, as it provides a very quick turnaround while testing the kernel and the root filesystem.

4.1 Development Host Preparations

On the development host a TFTP server must be installed and configured. Usually TFTP servers are using the */tftpboot* directory to fetch files from.

If you built your own images, please copy them from the BSP's directory *platform-phyCORE-AM335x/images* now to here.

We also need a network connection between the embedded board and the TFTP server. The server should be set to IP *192.168.3.10* and netmask *255.255.255.0*.

4.2 Stand-Alone Booting Linux

To use the the target standalone, the kernel and the rootfs have to be made persistent in the onboard media of the phyCORE-AM335x. The following sections describe the steps necessary to bring kernel and rootfs into the onboard NAND type flash.

After that, the phyCORE-AM335x can work independently from the development host. We can "cut" the network (and serial cable) and the phyCORE-AM335x will continue to work.

4.2.1 Preparations on the Embedded Board

The phyCORE-AM335x uses Barebox as its bootloader. Barebox can be customized with environment variables and scripts to support any boot constellation. BSP-phyCORE-AM335x-PD13.1.0 comes with a predefined environment setup to easily bring up the phyCORE-AM335x.

Usually the environment doesn't have to be set manually on our target. Due to the fact that some of the values of these Barebox environment variables must meet our local network environment and development host settings you need to define them prior to the next steps.

Connect to the target with your favorite terminal application. After connecting the board with the power supply, the target starts booting. Press any key to stop autoboot, then type:

```
ifup eth0
devinfo eth0
```

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

```
ipaddr=192.168.3.11
netmask=255.255.255.0
gateway=192.168.3.10
serverip=192.168.3.10
```

If you need to change something, type

```
edit /env/network/eth0
```

Edit the settings, save them by leaving the editor with Strg-D, then type *saveenv* and reboot the board. Otherwise leave the editor with Strg-C.

Now get the Linux kernel from your tftp-server and store it into the NAND flash:

```
erase /dev/nand0.kernel.bb
cp /mnt/tftp/linuximage /dev/nand0.kernel.bb
```

Note that the command `update` known from former BSP versions is not available anymore.

For flashing Linux's root file system into NAND, please use:

```
ubiformat /dev/nand0.root
ubiattach /dev/nand0.root
ubimkvol /dev/ubi0 root 0
cp /mnt/tftp/root.ubifs /dev/ubi0.root
```

Note that you should not flash Linux's root file system into NAND the same way as you did with Linux kernel. Ubifs keeps erase counters within the NAND in order to be able to balance write cycles equally over all NAND sectors. So if there's already an ubifs on your module and you want to replace it by a new one, using *erase* and *cp* will also erase these erase counters, and this should be avoided.

4.2.2 Booting the Embedded Board

By default kernel and root filesystem will always be booted from the same device from that you booted the bootloader. I.e. if you boot from NAND, the board will also boot kernel and root filesystem from NAND. If you boot from SD-card, kernel and root filesystem will also be booted from SD-card. If you boot from SPI-NOR, also the kernel will be booted from SPI-NOR. Because the root filesystem is too big to fit into SPI-NOR, it will be taken from NAND in this case.

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

This boot behaviour is set by a line in file */env/config*:

```
global.boot.default=$bootsource
```

You can replace *$bootsource* by the names of the scripts you'll find in */env/boot*. For example, if you want to have kernel and root filesystem always booted from NAND, regardless of from where the board booted, modify the line like this:

```
global.boot.default=nand
```

4.2.3 Using SD-Card instead of LAN

Alternatively of using LAN you can also update the images using an SD-card. Mounting it will be done automatically when you boot from it (see chapter 3.2). Update kernel by typing:

```
erase /dev/nand0.kernel.bb
cp /mnt/disk/linuximage /dev/nand0.kernel.bb
```

Update root filesystem by typing:

```
ubiformat /dev/nand0.root
ubiattach /dev/nand0.root
ubimkvol /dev/ubi0 root 0
cp /mnt/disk/root.ubifs /dev/ubi0.root
```

4.3 Remote-Booting Linux

The next method we want to try after building the linux kernel and the root filesystem is the network-remote boot variant. This method is especially intended for development as everything related to the root filesystem happens on the host only. It's the fastest way in a phase of a project, where things are changing frequently. Any change made in the local *root/*

directory of the corresponding *platform-phyCORE-AM335x* directory simply ”appears” on the embedded device immediately.

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

4.3.1 Development Host Preparations

The NFS server is not restricted to a certain filesystem location, so all we have to do on most distributions is to modify the file */etc/exports* and export our root filesystem to the embedded network. In this example file the whole work directory is exported, and the ”lab network” between the development host is 192.168.3.10, so the IP addresses have to be adapted to the local needs:

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

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

4.3.2 Booting the Embedded Board

Restart the board and stop autoboot by pressing *m*. You’ll get a menu:

```
Welcome to Barebox
     1: Boot: nand (UBI)
     2: Boot: kernel & rootfs on SD card
     3: Boot: network (tftp, nfs)
     4: Boot: SPI NOR Flash
     5: Settings
     6: Shell
     7: Reset
```

Press *3* and then the enter key in order to boot the board from network.

If you want to configure *env/config* for always booting from network, you need to set the following entry to:

```
global.boot.default=net
```

Chapter 5 Accessing Peripherals

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

Phytec's phyCORE-AM335x starter kit consists of the following individual boards:

1. The phyCORE-AM335x module itself, containing the controller, RAM, flash and several other peripherals.

2. The starter kit baseboard (PCM-953).

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

arch/arm/mach-omap2/board-pcm051.c

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

Note that the huge variety of possibilities offered by the phyCORE-AM335x modules makes it difficult to have a completely generic implementation on the operating system side. Nevertheless, the BSP can easily be adapted to customer specific variants. In case of interest, contact our sales department (sales@phytec.de) and ask for a dedicated offer.

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

5.1 NAND Flash

The phyCORE-AM335x module comes with NAND memory to be used as media for storing linux and its root filesystem, including applications and their data files. This type of media will be managed by the UBI filesystem. This filesystem uses compression and decompression on the fly, so there is a chance to bring more data into this device.

From Linux userspace the NAND flash partitions can be seen as

- */dev/mtdblock5* (X-Loader partition)
- */dev/mtdblock6* (X-Loader backup 1 partition)
- */dev/mtdblock7* (X-Loader backup 2 partition)
- */dev/mtdblock8* (X-Loader backup 3 partition)
- */dev/mtdblock9* (Barebox partition)
- */dev/mtdblock10* (Barebox environment partition)
- */dev/mtdblock11* (Kernel partition)
- */dev/mtdblock12* (Linux rootfs partition)

Only the */dev/mtdblock12* on the phyCORE-AM335x has a filesystem, so the other partitions cannot be mounted into the rootfs. The only way to access them is by pushing a prepared flash image into the corresponding */dev/mtd* device node.

X-loader is the Second Stage Bootloader, that will be loaded by the internal bootloader (1st stage Bootloader). In case that this block is damaged, the internal bootloader searches for a loadable backup partition.

The positions and sizes of the partitions are:

```
0x00000000 - 0x0001FFFF: "xload"          /dev/mtdblock5
0x00020000 - 0x0003FFFF: "xload_backup1"  /dev/mtdblock6
0x00040000 - 0x0005FFFF: "xload_backup2"  /dev/mtdblock7
0x00060000 - 0x0007FFFF: "xload_backup3"  /dev/mtdblock8
0x00080000 - 0x000FFFFF: "barebox"        /dev/mtdblock9
0x00100000 - 0x0011FFFF: "bareboxenv"     /dev/mtdblock10
0x00120000 - 0x0091FFFF: "kernel"         /dev/mtdblock11
0x00920000 - 0x1FFFFFFF: "root"           /dev/mtdblock12
```

5.2 Serial TTYs

The AM335x SoC supports up to 6 so called UART units. On the phyCORE-AM335x all six UARTs are routed to the Molex connectors. At connector X18 you'll find *ttyO0* which is the standard console.

Only */dev/ttyO0* has been implemented within this BSP.

5.3 Network

The phyCORE-AM335x module features ethernet 10/100Mbit, which is being used to provide the *eth0* network interface. By default it has the static IP 192.168.3.11.

The module also features gigabit ethernet, which is being used to provide *eth1* network interface. By default it has the static IP 192.168.4.11.

Both interfaces offer a standard Linux network port which can be programmed using the BSD socket interface.

5.4 SPI Master

The phyCORE-AM335x provides two SPI busses, based on the AM335x's integrated SPI controllers. A NOR flash has been connected to CS0 of SPI0, that can even be used for booting.

From Linux userspace the NOR flash partitions can be seen as

• */dev/mtdblock0* (X-Loader partition)

• */dev/mtdblock1* (Barebox partition)

• */dev/mtdblock2* (Barebox environment partition)

• */dev/mtdblock3* (Kernel partition)

• */dev/mtdblock4* (Linux rootfs partition)

Please note that the Linux rootfs partition is too small for the kit's rootfs.

The positions and sizes of the partitions are:

```
0x00000000 - 0x0001FFFF: "xload"          /dev/mtdblock0
0x00020000 - 0x0009FFFF: "barebox"        /dev/mtdblock1
0x000A0000 - 0x000BFFFF: "bareboxenv"     /dev/mtdblock2
0x000C0000 - 0x004BFFFF: "kernel"         /dev/mtdblock3
0x004C0000 - 0x007FFFFF : "root"          /dev/mtdblock4
```

5.5 I²C Master

The RTC RV-4162-C7 on the phyCORE-AM335x can be accessed as */dev/rtc0*. It also has the entry */sys/class/rtc/rtc0* in the sysfs filesystem, where you can read for example the *name*.

Date and time can be manipulated with the *hwclock* tool, using the *-w* (systohc) and *-s* (hctosys) options. For more information about this tool refer to the manpage of *hwclock*.

To set the date first use *date* (see *man date* on the PC) and then run *hwclock -w -u* to store the new date into the RTC.

Please note that in case you need to use RTC's interrupt, jumper JP23 needs to be closed on the baseboard. But this is only functional if you don't already are using this interrupt line for LCD-018 touch.

5.6 USB Host Controller

The AM335x CPU embedds a USB 2.0 EHCI controller that is also able to handle low and full speed devices (USB 1.1).

The BSP-phyCORE-AM335x-PD13.1.0 includes support for mass storage devices and keyboards. Other USB related device drivers must be enabled in the kernel configuration on demand.

Due to *udev*, connecting various mass storage devices get unique IDs and can be found in */dev/disks/by-id*. These IDs can be used in */etc/fstab* to mount different USB memory devices in a different way.

5.7 USB OTG

The BSP-phyCORE-AM335x-PD13.1.0 includes support for USB OTG. In order to activate it you need to load an appropriate module with *modprobe*, for example the ethernet gadget *g_ether*.

For using USB OTG as host, please load module *g_zero*.

5.8 MMC/SD Card

The phyCORE-AM335x in conjunction with its baseboard supports a slot for 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 unplugg the device while it's still mounted. This may result in data loss.

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

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

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

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

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

The cards are always mounted as being writable. Setting of write-protection of MMC/SD cards is not recognized.

5.9 GPIO

For setting and resetting the green user-LED1 on the baseboard just enter:

```
cd /sys/class/gpio
echo 62 > export
cd gpio62
echo out > direction
echo 1 > value
echo 0 > value
```

The yellow user-LED2 can be accessed the same way using GPIO 63.

5.10 CAN Bus

The phyCORE-AM335x provides a CAN feature, which is supported by drivers using the proposed Linux standard CAN framework ”Socket-CAN”. Using this framework, CAN interfaces can be programmed with the BSD socket API.

The CAN (Controller Area Network) 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. The Socket-CAN interface behaves like an ordinary Linux network device, with some additional features special to CAN. Thus for example you can use

```
ifconfig -a
```

in order to see if the interface is up or down, but the given MAC and IP addresses are arbitrary and obsolete.

Configuration happens within the script */etc/network/can-pre-up*. This script will be called when */etc/init.d/networking* is running at system start up. To change default used bitrates on the target change the variable *BITRATE* in */etc/network/can-pre-up*.

For a persistent change of the default bitrates change the local *projectroot/etc/network/can-pre-up* instead and rebuild the BSP.

The information for *can0* looks like

```
root@phyCORE-AM335x:~ 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:1284643 errors:0 dropped:0 overruns:0 frame:0
          TX packets:67450 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:10
          RX bytes:5138560 (4.9 MiB)  TX bytes:269767 (263.4 KiB)
          Interrupt:211
```

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). The following output parameters contain useful information:

Field	Description
can0	Interface Name
NOARP	CAN cannot use ARP protocol
MTU	Maximum Transfer Unit
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

You can send messages with *cansend* or receive messages with *candump*:

```
cansend can0 0x03 0x12 0x06
candump can0
```

See *cansend --help* and *candump --help* help messages for further information about using and options.

5.11 Framebuffer

This driver gains access to the display via device node */dev/fb0* for PHYTEC display connector.

The BSP is already prepared for use with the PrimeView displays PD050VL1 (640x480), PM070WL4 (800x480), PD104SLF (800x600) and ETM0700G0DH6 (800x480). Selection of this display can be done in the Barebox in script */env/video/display* simply by modifying comments of the lines

```
#Displays
#display=PV_PD050VL1
#display=PV_PM070WL4
#display=PV_PD104SLF
display=ETM0700G0DH6
```

A simple test of the framebuffer feature can then be run with:

```
fbtest
```

This will show various pictures on the display.

You can check your framebuffer resolution with the command

```
fbset
```

NOTE: *fbset* cannot be used to change display resolution or colour depth. Depending on the framebuffer device different kernel command line are mostly needed to do this. Please refer to the manual of your display driver for more details.

5.12 Touch

A simple test of this feature can be run with

```
ts_calibrate
```

to calibrate the touch and with

```
ts_test
```

to do a simple application using this feature.

5.13 AUDIO support

Audio support on the module is done via the I2S interface and controlled via I2C.

5.13.1 Audio Sources and Sinks

The baseboard has three jacks for Microphone, Headset Speaker and Line Out. Because microphone input is mono only, jumper JP5 selects which channel of the stereo jack will be used.

Enabling and disabling input and output channels can be done with the *alsamixer* program. F3 key selects screen *Playback* and F4 key selects screen *Capture*. Tabulator key toggles between these screens.

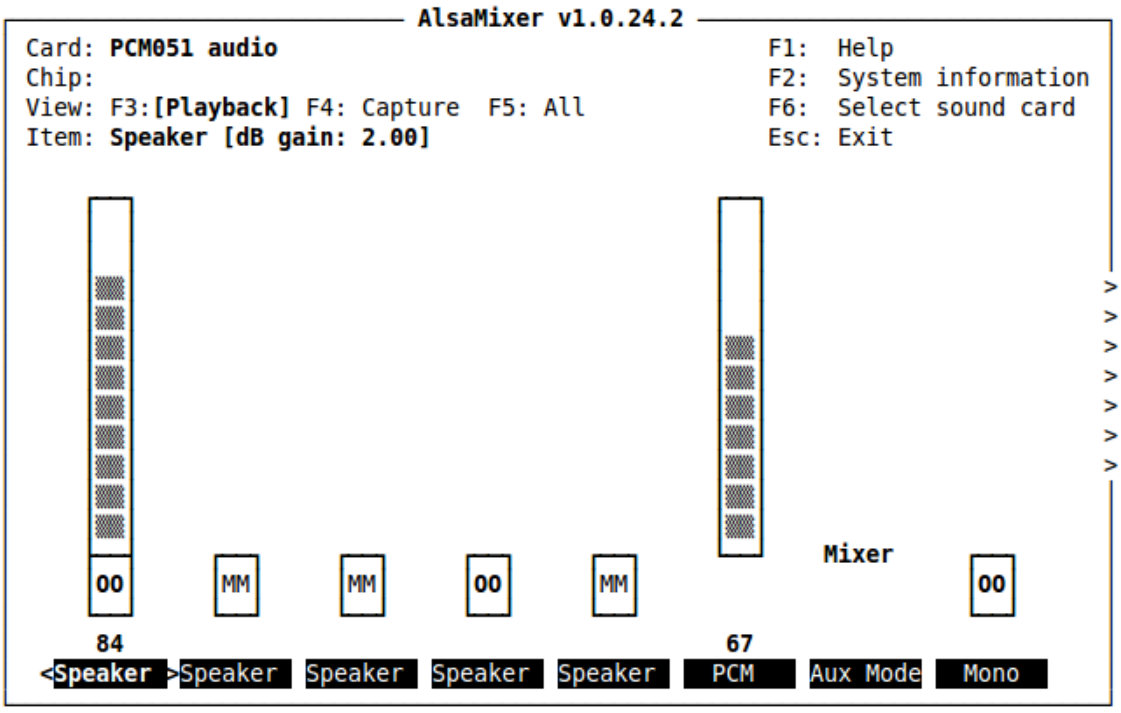

Figure 5.1: Playback controls

With the keys cursor left and cursor right you can step through the different channels. There are much more channels than fit onto one screen, so the screen will scroll if your cursor reaches the right or left edge of it. In case you get trouble in the display during scrolling, please use *telnet* instead of *microcom*.

alsamixer can be left by pressing the *ESC* key. If you want these settings to be persistent you can run a *alsactl store* now. This will store the current

audio mixer settings into the file */etc/asound.state*. At the next system start these settings will be restored by the */etc/init.d/alsa-utils* script.

5.13.2 Playback

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

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

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

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

```
aplay /usr/share/supertux/sounds/coffee.wav
```

5.13.3 Capture

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

The following example will capture the current stereo input source with 48kHz 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 48000 -f S16_LE test.wav
```

Capturing can be stopped again using Strg-C.

5.14 Using Qt

Nokia's Qtopia is very commonly used for embedded systems and it's supported by this BSP. Please visit http://qt.nokia.com in order to get all the documentation that are available about this powerful cross-platform GUI toolkit.

Within the BSP come some demos that show what is possible with Qt version 4.8.4. They're located in */usr/bin/qt4-demos*. In order to try them you need a touch display attached to the board. Please go into this directory with

```
export QWS_MOUSE_PROTO=tslib:/dev/input/event0
```

```
cd /usr/bin/qt4-demos
```

and then start demos with commands like

```
spreadsheet/spreadsheet -qws
```

Each of the Qt applications shows a standardized little green Qt-icon at its upper left side that offers standard window functions like minimize, maximize and close.

Demo `fluidlauncher` will be started automatically after booting.

5.15 CPU core frequency scaling

The phyCORE-AM335x supports dvfs (dynamic voltage and frequency scaling). Four different frequencies are supported. Type:

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies
```

and you'll get them all listed. In case you have an AM3359 with a maximum of 800MHz these are:

```
300000 600000 720000 800000
```

The voltages are scaled according to the setup of the frequencies.

You can decrease the maximum frequency (e.g. to 720000)

```
echo 720000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
```

or increase the minium frequency (e.g. to 600000)

```
echo 600000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq
```

Asking for the current frequency will be done with:

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
```

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

performance Always selects the highest possible CPU core frequency.

powersave Always selects the lowest possible CPU core frequency.

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.

userspace Allows the user or userspace program running as root to set a specific frequency (e.g. to 600000):

```
echo 600000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
```

So when you type

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_governors
```

you’ll get the result:

```
userspace powersave ondemand performance
```

In order to ask for the current governor, type

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

and you'll normally get:

```
ondemand
```

Switching over to another governor (e.g. *userspace*) will be done with:

```
echo userspace > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

For more detailed information about the govenors refer to the linux kernel documentation in:

Documentation/cpu-freq/governors.txt.

Chapter 6 Getting help

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

6.1 Mailing Lists

6.1.1 About PTXdist in Particular

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

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

how to subscribe to this list. If you want to search through the mailing list archive, visit

http://www.mail-archive.com/

and search for the list ptxdist. Please note again that this mailing list is just related to the PTXdist as a software. For questions regarding your specific BSP, see the following items.

6.1.2 About Embedded Linux in General

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

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

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

6.2 News Groups

6.2.1 About Linux in Embedded Environments

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

comp.os.linux.embedded

6.2.2 About General Unix/Linux Questions

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

de.comp.os.unix.programming

6.3 Chat/IRC

About PTXdist in particular

irc.freenode.net:6667

Create a connection to the *irc.freenode.net:6667* server and enter the chatroom *#ptxdist*. This is an English room to answer questions about PTXdist. Best time to meet somebody there is at European daytime.

6.4 phyCORE-AM335x Support

support@phytec.de

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

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

and then navigate to *Support / FAQ / Modules / phyCORE-AM335x*

6.5 Commercial Support

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

Contact us at:

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

or by electronic mail:

sales@phytec.de

Document:	BSP-Quickstart phyCORE-AM335x
Document Number:	L-775e_3 **September** 2013

How would you improve this manual?

Did you find any mistakes in this manual? page

Submitted by:

Customer number: _____

Name: _____

Company: _____

Address: _____

Return to:

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

Fax: +49 (6131) 9221-26