BSP-Quickstart phyCORE-OMAP44xx

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: July 2013

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. They are all already preinstalled on the LiveSystem.

Generally the central place for our BSPs is our ftp-server ftp://ftp.phytec.de/pub/Products/phyCORE-OMAP44xx. 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).

To build phyCORE-OMAP4-PD13.1.0, the following archives have to be available on the development host:

- ptxdist-2013.01.0.tgz
- OSELAS.Toolchain-2011.11.1
- phyCORE-OMAP4-PD13.1.0.tar.gz

If you do not use our LiveSystem, it is necessary to get them.

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 PTXdist installation

PTXdist has been installed into the standard path */usr/local/bin/ptxdist*. The installation paths are configured in a way that several PTXdist versions can be installed in parallel. So if you want to install a newer version of PTXdist, there is no need to remove the old one. You will then simply 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 the BSP asks for a dedicated version of PTXdist. It may cause much work to try to build the BSP with a newer version of PTXdist than it requires.

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-cortexa9-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-cortexa9-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 and has already been done on our LiveSystem. You will find it under */opt/OSELAS.Toolchain-2011.11.1/*.

Chapter 2 Building phyCORE-OMAP44xx's BSP

2.1 The Board Support Package

In the directory */opt/PHYTEC_BSPs/phyCORE-OMAP4-PD13.1.0* you will find the prebuild BSP with all its valid components. The most important ones 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.

tests Contains test scripts for automated target setup.

2.2 Selecting a Hardware Platform

Before we can build this BSP, we need to select one of the possible targets to build for. In this case we want to build for the phyCORE-OMAP44xx. Although this has already been done on the LiveSystem, just for your understanding please change to directory */opt/PHYTEC_BSPs/phyCORE-OMAP4-PD13.1.0* and type:

```
ptxdist platform configs/phyCORE-OMAP4-2013.01.0/platformconfig
```

You will see:

```
info: selected platformconfig:
'configs/phyCORE-OMAP4-2013.01.0/platformconfig'
```

PTXdist should also automatically detect the proper toolchain:

```
found and using toolchain:
'/opt/OSELAS.Toolchain-2011.11.1/arm-cortexa9-linux-gnueabi/
gcc-4.6.2-glibc-2.14.1-binutils-2.21.1a-kernel-2.6.39-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-2011.11.1/\     [Enter]
arm-cortexa9-linux-gnueabi/\     [Enter]
gcc-4.6.2-glibc-2.14.1-binutils-2.21.1a-kernel-2.6.39-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.

Note: Because the BSP has been prebuilt by us, the result is that *ptxdist go* will build nothing if you have not changed anything within the BSP.

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-OMAP4/images/* directory and the final root filesystem for the target board can be found in the *platform-phyCORE-OMAP4/root/* directory and a bunch of **.ipk* packets in the *platform-phyCORE-OMAP4/packages/* directory, containing the single applications the root filesystem consists of.

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.ubi: root files inside a ubi filesystem.

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

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-OMAP4/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-OMAP4/root/* will not be enclosed in the image.

Chapter 3 phyCORE-OMAP44xx preparation

This step can be omitted if your phyCORE-OMAP44xx already has the right boot loader: For this BSP the barebox version 2013.06.0 is required.

3.1 Updating Barebox

Build the whole BSP with *ptxdist go* or just build the barebox with *ptxdist targetinstall barebox*. When it's built copy the generated file *platform-phyCORE-OMAP4/images/barebox-image* to your configured tftp exported directory.

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:

```
edit /env/config
```

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

```
eth0.ipaddr=192.168.3.11
```

```
eth0.netmask=255.255.255.0
```

```
eth0.serverip=192.168.3.10
```

and dhcp must be disabled, that means commented out:

```
#ip=dhcp
```

If you need to change something, save your settings 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-image
```

After that store the Barebox into the NAND flash,

```
erase /dev/nand0.barebox.bb
cp barebox-image /dev/nand0.barebox.bb
```

The same can be done with one simple command:

```
update -t barebox -d nand -m tftp -f barebox-image
```

Note that if something goes wrong at this, you don’t have any bootloader anymore on your module. In this case you need to install a new one as being described in the Quickstart Manual.

The splash screen that Barebox can show can be updated with command:

```
update -t splash -d nand -m tftp -f Splashscreen_800x600.bmp
```

3.2 Updating X-Loader

The X-Loader the OMAP needs is just a little Barebox. You can update it with command:

```
update -t xload -d nand –m tftp -f MLO
```

3.3 Updating using SD-Card instead of LAN

Alternatively of using LAN you can also update the images using an SD-Card which has been formatted with FAT. Move the images to your SD-Card and put the card into slot SDMMC1. Then type:

```
mkdir /boot
mount /dev/disk0.0 fat /boot
```

With an `ls /boot` you should now get the images listed. Updating X-Loader and Barebox can now be done by typing:

```
nand_bootstrap
```

This is a script that does the following (could also be typed manually):

```
gpmc_nand0.eccmode=${xload_eccmode}
erase /dev/nand0.xload.bb
cp /boot/MLO /dev/nand0.xload.bb

gpmc_nand0.eccmode=${barebox_eccmode}
erase /dev/nand0.barebox.bb
cp /boot/barebox-image /dev/nand0.barebox.bb
```

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-OMAP44xx. 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.

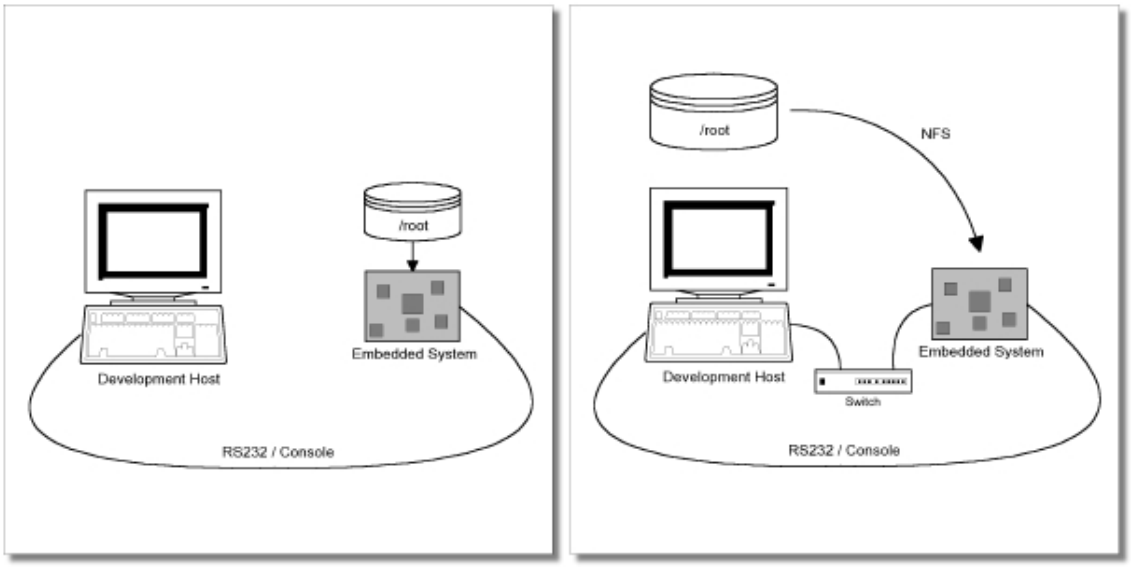

Figure 4.1: Booting the phyCORE-OMAP44xx: From its flash or from the host via network.

Figure 4.1 shows both methods. The main method used in the BSP phyCORE-OMAP4-PD13.1.0 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-OMAP44xx 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-OMAP44xx/root/* directory in our PTXdist workspace.

The BSP phyCORE-OMAP4-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. This has already been done on our LiveSystem.

Usually TFTP servers are using the */tftpboot* directory to fetch files from. The images of kernel and rootfs that have been prebuilt by us are already located there.

If you built your own images, please copy them now to here:

```
/opt/PHYTEC_BSPs/phyCORE-OMAP4-PD13.1.0/platform-phyCORE-
OMAP44xx/images# cp linuximage /tftpboot
/opt/PHYTEC_BSPs/phyCORE-OMAP4-PD13.1.0/platform-phyCORE-
OMAP44xx/images# cp root.ubi /tftpboot
```

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-OMAP44xx. The following sections describe the steps necessary to bring kernel and rootfs into the onboard NAND type flash.

Only for preparation we need a network connection to the embedded board and a network aware bootloader which can fetch any data from a TFTP server.

After preparation is done, the phyCORE-OMAP44xx can work independently from the development host. We can ”cut” the network (and serial cable) and the phyCORE-OMAP44xx will continue to work.

4.2.1 Preparations on the Embedded Board

The phyCORE-OMAP44xx uses Barebox as its bootloader. Barebox can be customized with environment variables and scripts to support any boot constellation. BSP phyCORE-OMAP4-PD13.1.0 comes with a predefined environment setup to easily bring up the phyCORE-OMAP44xx.

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:

```
edit /env/config
```

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

```
eth0.ipaddr=192.168.3.11
eth0.netmask=255.255.255.0
eth0.serverip=192.168.3.10
```

and dhcp must be disabled, that means commented out:

```
#ip=dhcp
```

If you need to change something, save your settings by leaving the editor with Strg-D, then type *saveenv* and reboot the board. Otherwise leave the editor with Strg-C.

Now you can use the *update* script within barebox in order to flash both images on the target:

```
update -t kernel -d nand –m tftp -f linuximage

update -t rootfs -d nand -m tftp -f root.ubi
```

4.2.2 Booting the Embedded Board

After the next reset or powercycle of the board, it should boot the kernel from the flash, start it and mount the root filesystem also from flash.

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

4.2.3 Using SD-Cards instead of LAN

See chapter 3.3 for how to mount SD-Cards within the Barebox. So if you put your images onto an SD-Card instead on a tftp server and mount it, updating the kernel and the rootfs can be done simply:

```
erase /dev/nand0.kernel.bb

cp /boot/linuximage /dev/nand0.kernel.bb

erase /dev/nand0.root.bb

cp /boot/root.ubi /dev/nand0.root.bb
```

4.2.4 Remarks upon UBI file system

One of the main tasks of file systems for flash memory like UBI is to balance write cycles equably over all blocks of the partition, because the amount of possible erases of each block is limited. So UBI has an erase counter for every block. But when you erase the whole partition by using `update -t rootfs -d nand` or `erase /dev/nand0.root.bb`, you also erase the erase counters.

If you want to update an UBI file system without resetting its erase counters, you need to use `ubiformat` instead:

```
ubiformat /dev/nand0.root -f /boot/root.ubi
```

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-OMAP44xx* 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.

Hint: Barebox of the phyCORE-OMAP44xx uses UDP for NFS, not TCP!

4.3.2 Preparations on the Embedded Board

The default environment settings coming with the BSP phyCORE-OMAP4-PD13.1.0 has the possibility to boot from the internal flash or from the network. Configuration happens in the file */env/config*. As Barebox uses a full shell like console you can edit this file to configure the other scripts (the boot script for example).

To edit this configuration file we run the *edit* command on it:

```
edit /env/config
```

We move to the lines that define the *kernel_loc* and *rootfs_loc* variables. They can be defined to *nand* or *net*. In this example we change them to *net* to load all parts from the network. When we do that, we also have to configure the network setup a few lines above in this file. We setup these values to the network we want to run the phyCORE-OMAP44xx.

Leaving this editor with saving the changes happens with CTRL-D. Leaving it without saving the changes happens with CTRL-C.

Note: Saving here means the changes will be saved to the RAM disks Barebox uses for the environment. To store it to the persistent memory, an additional *saveenv* command is required.

4.3.3 Booting the Embedded Board

Now its time to boot the phyCORE-OMAP44xx. To do so, simply run:

```
boot
```

This command should boot the phyCORE-OMAP44xx into the login prompt.

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

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-OMAP44xx starter kit consists of the following individual boards:

1. The phyCORE-OMAP44xx module itself, containing the controller, RAM, flash and several other peripherals.

2. The starter kit baseboard (PCM-959).

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-omap4pcm049.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-OMAP44xx 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-OMAP44xx 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/mtdblock0* (Xloader partition)

• */dev/mtdblock1* (Barebox partition)

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

• */dev/mtdblock3* (Kernel partition)

• */dev/mtdblock4* (Splash picture partition)

• */dev/mtdblock5* (Linux rootfs partition)

Only the */dev/mtdblock4* on the phyCORE-OMAP44xx 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.

The following line in the environment of the bootloader defines the sizes of the partitions

```
nand_parts="128k(xload)ro,512k(barebox),128k(bareboxenv),4M(kernel),4M(splash)
,-(root)"
```

which results in:

```
0x00000000 - 0x0001FFFF: "xload"          /dev/mtdblock0
0x00020000 - 0x0009FFFF: "barebox"        /dev/mtdblock1
```

```
0x000A0000 - 0x000BFFFF: "bareboxenv"     /dev/mtdblock2
0x000C0000 - 0x004BFFFF: "kernel"         /dev/mtdblock3
0x004C0000 - 0x008BFFFF: "kernel"         /dev/mtdblock4
0x008C0000 - 0x1FFFFFFF: "root"           /dev/mtdblock5
```

5.2 Serial TTYs

The OMAP44xx SoC supports up to 4 so called UART units. On the phyCORE-OMAP44xx all four UARTs are routed to the Molex connectors. Available for user's applications are:

• *ttyO1* at connector P1 (bottom connector). Unused in this BSP.

• *ttyO2* at connector P1 (top connector) used as the main kernel and control console.

5.3 Network

The phyCORE-OMAP44xx module features ethernet, 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.

5.4 I²C Master

The OMAP44xx processor based phyCORE-OMAP44xx 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.

5.4.1 I²C PMIC TWL6030

The RTC of the PMIC TWL6030 on the phyCORE-OMAP44xx 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.

5.4.2 I²C Device 24WC32W

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

This type of memory is accessible through the sysfs filesystem. To read the EEPROM content simply *open()* the entry */sys/bus/i2c/devices/1-0050/eeprom* and use *fseek()* and *read()* to get the values.

5.5 Framebuffer

This driver gains access to the display via device node */dev/fb0*. For this BSP the DVI connector is default. In order to get it set up properly, please turn on the attached monitor before booting the phyCORE-OMAP44xx.

The BSP is also already prepared for use with the following displays:

PrimeView	PD050VL1	640x480
PrimeView	PM070WL4	800x480
PrimeView	PD104SLF	800x600
EDT	ETM0350G0DH6	320x240
EDT	ETM0430G0DH6	480x272
EDT	ETMV570G0DHU	640x480
EDT	ETM0700G0DH6	800x480

Selection of one of these displays can be done in the Barebox in script */env/config*. There you will find the lines

```
#Displays
# Splashscreen-Display can be either '', 'pd050vl1', 'pm070wl4', 'pd104slf',
# 'edt_etm0350G0dh6', 'edt_etm0430G0dh6', 'edt_etmv570G2dhu' or 'edt_etm0700G0dh6'
# to use dvi output in kernel set 'display=dvi' and
# dvi_resolution to '640x480-60' '800x600-60' or '1024x768-60'

display=edt_etm0700G0dh6
#dvi_resolution=1024x768-60
```

Just change the entry in order to switch over to another display. Please note that currently it’s not possible to drive DVI output and another display at the same time. The default setup of DVI resolution is 1024x768 at 60MHz, but as you can see this can also easily being changed the same way.

A simple test of the framebuffer 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 are mostly needed to do this. Please refer to the manual of your display driver for more details.

5.6 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.

5.7 USB Host Controller

The OMAP44xx 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-OMAP4-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.

Please note that USB OTG is currently unable to switch from device to host, automatically. After booting USB OTG is set to device. In order to set it to host you need to load the *g_zero gaget*:

```
modprobe g_zero
```

Additionally you need to close J9 if you want to use mass storage devices.

5.8 MMC/SD Card

The phyCORE-OMAP44xx in conjunction with its baseboard supports two slots 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* or

/dev/mmcblk1 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* or */dev/mmcblk1* 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 AUDIO support

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

5.9.1 Audio Sources and Sinks

The baseboard has three jacks for Microphone, Headset Speaker and Line Out. Jumpers JP15 and JP16 must both be set in position 1+2 in order to use the headset speaker jack.

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.

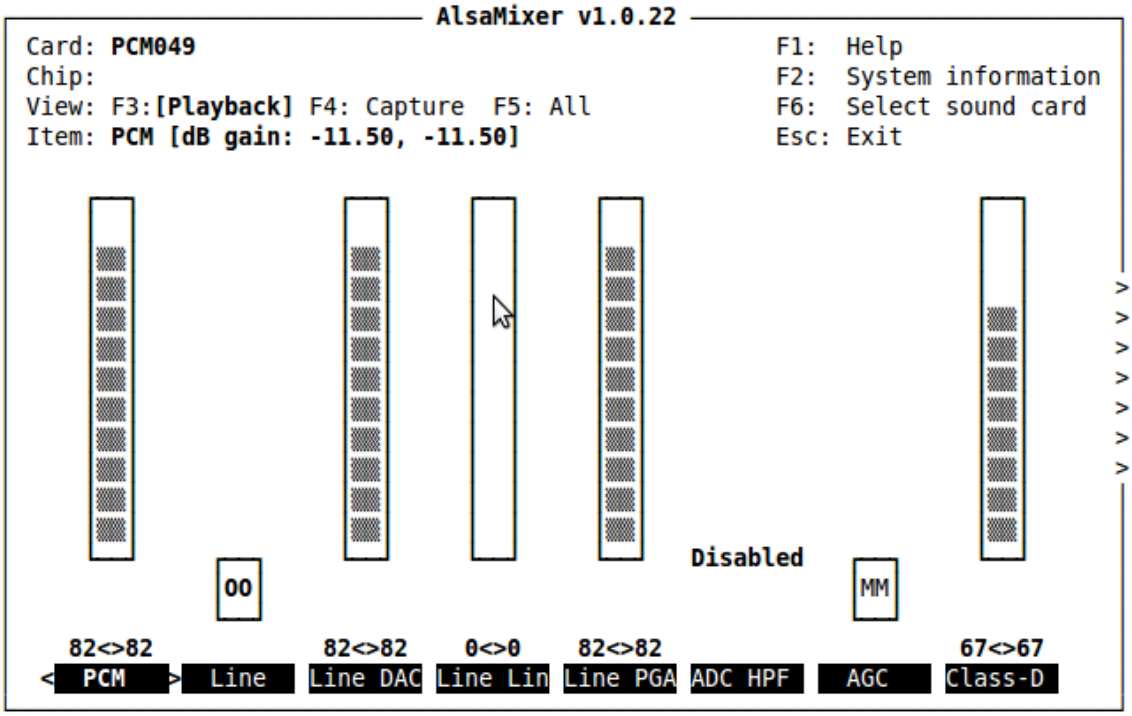

Figure 5.1: Playback controls

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.

5.9.2 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
```

5.9.3 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 16kHz sample rate and will create an audio file in *WAV* format (signed 16 bit per channel, 32 bit per sample):

```
~# arecord -t wav -c 2 -r 16000 -f S16_LE /demo.wav
```

The following example will capture the current stereo input source with 8kHz sample rate and will create an audio file in *WAV* format (signed 16 bit per channel, 32 bit per sample):

```
~# arecord -t wav -c 2 -r 8000 -f S16_LE /demo.wav
```

Capturing can be stopped again using Strg-C.

5.10 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 comes a demo that shows what is possible with Qt version 4.8.4. It's the fluidlauncher, located in */usr/bin/qt4-demos/embedded/fluidlauncher*. It will start automatically at the end of the boot sequence.

This Qt application will present some other Qt applications like an calculator, an imageviewer and an text editor. Touch the Qt application presented in the middle of the screen in order to start it or touch one of the Qt applications beside it in order to move this into the middle so that it can be started.

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. Closing the application will bring you back to the fluidlauncher.

Placed right beside the several demo Qt applications you'll find the dummy application *Exit Embedded Demo*. Move this into the middle of the screen and then touch it in order to close the fluidlaucher again.

5.11 CPU core frequency scaling

The phyCORE-OMAP44xx 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.

OMAP4430: `300000 600000 800000 1008000`
OMAP4460: `350000 700000 920000 1200000`

(OMAP4460 might have `1500000`)

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

You can decrease the maximum frequency (e.g. to 800000)

```
echo 800000 > /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
```

or set a frequency (e.g. to 600000)

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

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.

userspace Allows the user or userspace program running as root to set a specific frequency.

interactive Allows the user to get a full dynamic cpu frequency capable system by simply loading your cpufreq low-level hardware driver, using this governor for latency-sensitive workloads. This is the default governor when the system starts up.

So when you type

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

you'll get the result:

```
userspace powersave ondemand performance interactive
```

In order to ask for the current governor, type

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

and you'll normally get:

```
interactive
```

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-OMAP44xx 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-OMAP44xx*

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-OMAP44xx
Document Number:	L-768e_3 **July** 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