Quickstart Manual

OSELAS.BSP( )

phyCORE-i.MX27

	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 (800) 0749832 order@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

1st Edition: July 2011

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-iMX27. 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 OSELAS.BSP-Phytec-phyCORE-i.MX27-PD11.1.0, the following archives have to be available on the development host:

- ptxdist-2010.11.1.tgz
- OSELAS.BSP-Phytec-phyCORE-i.MX27-PD11.1.0.tar.gz
- OSELAS.Toolchain-2011.02.0

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 an OSELAS.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 every OSELAS.BSP( ) board support package 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.

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-v5te-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-2011.02.0/*.

Chapter 2 Building phyCORE-i.MX27's BSP

2.1 The Board Support Package

In the directory */opt/PHYTEC_BSPs/OSELAS.BSP-Phytec-phyCORE-i.MX27-PD11.1.0* you will find the prebuild BSP with all its valid components. The most important ones are:

```
/opt/PHYTEC_BSPs/OSELAS.BSP-Phytec-phyCORE-i.MX27-PD11.1.0# ls -l
-rw-r--r--  1 ubuntu ubuntu  .......  ChangeLog.txt
drwxr-xr-x  5 ubuntu ubuntu  .......  configs
drwxr-xr-x  2 ubuntu ubuntu  .......  documentation
drwxr-xr-x  6 ubuntu ubuntu  .......  patches
drwxr-xr-x  8 ubuntu ubuntu  .......  projectroot
drwxr-xr-x  2 ubuntu ubuntu  .......  rules
drwxr-xr-x  2 ubuntu ubuntu  .......  tests
```

ChangeLog Here you can read what has changed in this release. Note: This file does not always exist.

configs A multiplatform BSP contains configurations for more than one target. This directory contains the platform configuration files.

documentation If this BSP is one of our OSELAS BSPs, this directory contains the Quickstart you are currenly reading in.

patches If some special patches are required to build the BSP for this target, then this directory contains these patches on a per package basis.

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-i.MX27. Although this has already been done on the LiveSystem, just for your understanding please change to directory */opt/PHYTEC_BSPs/OSELAS. BSP-Phytec-phyCORE-i.MX27-PD11.1.0* and type:

```
ptxdist platform configs/phyCORE-i.MX27-2011.02.0/platformconfig
```

You will see:

```
info: selected platformconfig:
'configs/phyCORE-i.MX27-2011.02.0/platformconfig'
```

Because we also have installed the OSELAS.Toolchain at its default location, PTXdist should also detect the proper toolchain while selecting the platform. In this case it will output:

```
found and using toolchain:
'/opt/OSELAS.Toolchain-2011.02.0/arm-v5te-linux-gnueabi/
gcc-4.5.2-glibc-2.13-binutils-2.21-kernel-2.6.36-sanitized/bin'
```

If it fails you can continue to select the toolchain manually as mentioned in the next section. If this autodetection was successful, 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 \ [Enter]
/opt/OSELAS.Toolchain-2011.02.0/arm-v5te-linux-gnueabi/\     [Enter]
gcc-4.5.2-glibc-2.13-binutils-2.21-kernel-2.6.36-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-i.MX27/images/* directory and the final root filesystem for the target board can be found in the *platform-phyCORE-i.MX27/root/* directory and a bunch of **.ipk* packets in the *platform-phyCORE-i.MX27/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.jffs2: root files inside a jffs2 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-i.MX27/images/*.

CAUTION

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-i.MX27/root/* will not be enclosed in the image.

Chapter 3 phyCORE-i.MX27 preparation

This step can be omitted if your phyCORE-i.MX27 already has the right boot loader: For this BSP the barebox-2011.03.0 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-i.MX27/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:

```
barebox:/ 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 *save* and reboot the board. Otherwise leave the editor with Strg-C.

If your current Barebox is older than version 2011.03.0, then enter:

```
barebox:/ tftp barebox-image
```

to load this binary image into the RAM disk of your target.

Now store the Barebox into the NAND flash,

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

Because the older Barebox version had been stored in the NOR flash, but now we're using NAND, prior to a reboot you need to set switches 4 and 6 of the DIP-switch S5 to boot from NAND. Otherwise the old barebox will still reappear.

Since version 2011.03.0 you can simply type:

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

CAUTION

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.

You have the possibility to load a splash screen that will be displayed during boot sequence of Barebox and Linux:

```
barebox:/ update -t splash -d nand -m tftp -f splash.bmp
```

The image file to be displayed must be of type *.bmp*, should have a resolution of 240x320 and may not be bigger than 256kByte.

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-i.MX27. 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.

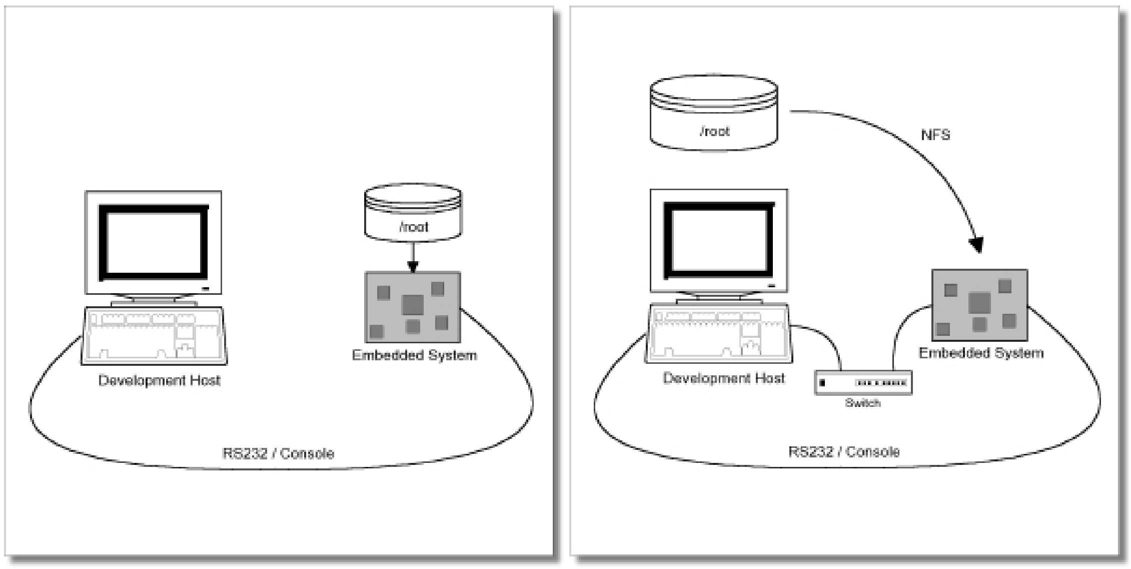


Figure 4.1: Booting the phyCORE-i.MX27: From its flash or from the host via network.

Figure 4.1 shows both methods. The main method used in the OSELAS.BSP-Phytec-phyCORE-i.MX27-PD11.1.0 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-i.MX27 with a serial nullmodem cable and via ethernet; the embedded board boots into the bootloader, then issues a TFTP request on the network and boots the kernel from the TFTP server on the host. Then, after decompressing the kernel into the RAM and starting it, the kernel mounts its root filesystem via NFS (Network File System) from the original location of the *platform-phyCORE-i.MX27/root/* directory in our PTXdist workspace.

The OSELAS.BSP-Phytec-phyCORE-i.MX27-PD11.1.0 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/OSELAS.BSP-Phytec-phyCORE-i.MX27-PD11.1.0/platform-
phyCORE-i.MX27/images# cp linuximage /tftpboot
/opt/PHYTEC_BSPs/OSELAS.BSP-Phytec-phyCORE-i.MX27-PD11.1.0/platform-
phyCORE-i.MX27/images# cp root.jffs2 /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-i.MX27. 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-i.MX27 can work independently from the development host. We can ”cut” the network (and serial cable) and the phyCORE-i.MX27 will continue to work.

4.2.1 Preparations on the Embedded Board

The phyCORE-i.MX27 uses Barebox as its bootloader. Barebox can be customized with environment variables and scripts to support any boot constellation. OSELAS.BSP-Phytec-phyCORE-i.MX27-PD11.1.0 comes with a predefined environment setup to easily bring up the phyCORE-i.MX27.

Usually the environment doesn’t have to be set manually on our target. 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:

```
barebox:/ 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 *save* 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:

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

barebox:/ update -t rootfs -d nand -m tftp -f root.jffs2
```

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.

4.3 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-i.MX27* 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 Preparations on the Embedded Board

The default environment settings coming with the OSELAS.BSP-Phytec-phyCORE-i.MX27-PD11.1.0 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:

```
barebox:/ edit /env/config
```

We move to the lines that define the *kernel_loc* and *rootfs_loc* variables. They can be defined to *nand*, *nor* 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-i.MX27.

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 *save* command is required.

4.3.3 Booting the Embedded Board

Now its time to boot the phyCORE-i.MX27. To do so, simply run:

```
barebox:/ boot
```

This command should boot the phyCORE-i.MX27 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-i.MX27 starter kit consists of the following individual boards:

1. The phyCORE-i.MX27 module itself, containing the controller, RAM, flash and several other peripherals.

2. A mapper board. The module is to be mounted on this.

3. The starter kit baseboard (PCM-970). Mount the mapper board on it.

To achieve maximum software re-use, the Linux kernel offers a sophisticated infrastructure, layering software components into board specific parts. The OSELAS.BSP( ) tries to modularize the kit features as far as possible; that means that when a customized 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-imx/mach-pcm038.c	for the CPU module
arch/arm/mach-imx/pcm970-baseboard.c	for the baseboard

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

CAUTION

Note that the huge variety of possibilities offered by the phyCORE-i.MX27 modules makes it difficult to have a completely generic implementation on the operating system side. Nevertheless, the OSELAS.BSP( ) can easily be adapted to customer specific variants. In case of interest, contact 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-i.MX27 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 JFFS2 filesystem. This filesystem uses compression and decompression on the fly, so there is a chance to bring more data into this device.

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

From linux userspace the NAND flash partitions can be seen as

- */dev/mtdblock5* (Barebox partition)
- */dev/mtdblock6* (Barebox environment partition)
- */dev/mtdblock7* (Splash Screen partition)
- */dev/mtdblock8* (Kernel partition)
- */dev/mtdblock9* (Linux rootfs partition)

Only the */dev/mtdblock9* on the phyCORE-i.MX27 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.

5.2 NOR Flash

Linux offers the Memory Technology Devices Interface (MTD) to access low level flash chips, directly connected to a SoC CPU.

Modern kernels offer a method to define flash partitions on the kernel command line, using the *mtdparts* command line argument:

```
mtdparts=physmap-flash.0:256k(uboot)ro,128k(ubootenv),2M(kernel),-
(root)
```

This line, for example, specifies several partitions with their size and name which can be used as */dev/mtd0*, */dev/mtd1* etc. from Linux. So if there is any need to change the partitioning layout, the Barebox environment is the only place where the layout has to be changed.

From userspace the NOR flash partitions can be accessed as

• */dev/mtdblock0* (e.g. Barebox partition)

• */dev/mtdblock1* (e.g. Barebox environment partition)

• */dev/mtdblock2* (e.g. Splash Screen partition)

• */dev/mtdblock3* (e.g. Kernel partition)

• */dev/mtdblock4* (e.g. Linux rootfs partition)

Note: This is an example only. Older BSPs than PD11.1.0 put their images into the 32MB NOR flash. Due to much more increased demands in graphic capabilities the current BSPs use NAND flash instead, which comes with 64MB at least.

You can type

```
flash_eraseall -j /dev/mtd4
```

in order to get a clean jffs2 formatted partition that can be mounted with

```
mkdir /mnt/nor
mount -t jffs2 /dev/mtdblock4 /mnt/nor
```

to your file system.

CAUTION Do not use the *flash_eraseall* command with the *-j* command line option for NAND memories. *-j* is intended only to be used for NOR memories!

5.3 Serial TTYs

The i.MX27 SoC supports up to 6 so called UART units. On the phyCORE-i.MX27 three UARTs are routed to the connectors and can be used in user's application:

• *ttymxc0* at connector P1 (bottom connector) used as the main kernel and control console.

• *ttymxc1* at connector P1 (top connector). Unused in this BSP

• *ttymxc2* at expansion connector. Unused in this BSP

5.4 Network

The phyCORE-i.MX27 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.

5.5 Camera Interface

The PCM-970 comes with a port for phyCAM-P cameras. The drivers are compatible to Video4Linux2 (v4l2) and thus they work for example with gstreamer. The PHYTEC camera series VM-006, VM-007, VM-009 and VM-010 are supported.

CAUTION Note that VM-009 may only be used on PCM-970 at least of revision PL1280.5. Otherwise the camera might be damaged due to too high voltages.

Detailed information about the phyCAM - interface and the PHYTEC camera boards you will find in the "phyCAM-P-S" manual L-748.

5.5.1 Camera Drivers

PHYTEC ships different kind of cameras with their development kits. The following camera drivers are available:

mt9v022 This driver supports Micron MT9V022 and MT9V024 picture sensors. If we have a camera of the VM-007 or VM-010 series, we will most probably need this driver. This driver is not capable to detect if the camera comes with a colour or monochrome sensor, so you must set the boot parameter *mt9v022.sensor_type=colour* within */env/config* of barebox’s environment for a colour camera and you must remove it for a b/w camera. mt9v024 uses the same driver as mt9v022.

mt9m001 This driver supports Micron MT9M001 b/w picture sensors, as built in cameras of the VM-006 series.

mt9m111 This driver supports Micron MT9M131 colour picture sensors, as built in cameras of the VM-009 series.

When loaded by command

```
modprobe mx2_camera
```

the drivers try to detect a camera, which they support. If the check is successful, the driver will register the camera device to the video subsystem and create a device node name */dev/video<x>* in the file system, where the *<x>* stands for the numbering of the device, starting with 2. If we have one camera only, which is mostly the case, it should be */dev/video2*. If our system misses this file, we have most probably no suitable driver loaded or our camera is not attached properly.

5.5.2 Using the Camera Support with GStreamer

A very easy way to handle pictures and videostreams from cameras is to use the popular Linux-tool GStreamer. Description of it follows in the next chapter. At this point it is only necessary to know that along with the OSELAS.BSP-Phytec-phyCORE-i.MX27-PD11.1.0 BSP comes a set of preconfigured scripts that use this tool. They are capable to do some basic guessing what kind of hard/software we have and then start the camera capturing with proper parameters.

We can find the convenience scripts on our target under */home/gstreamer-examples*.

```
~# ls gstreamer-examples
bwcam-fbdev_240x320            colcam-save_jpg_full_res
bwcam-fbdev_240x320-roi        colcam-save_raw_full_res
bwcam-mpeg4                    colcam-xv
bwcam-save_jpg_full_res        func.sh
bwcam-save_raw_full_res        more_mt9m131_scripts
bwcam-xv                       register-settings-mt9m001.txt
colcam-fbdev_240x320           register-settings-mt9m131.txt
colcam-fbdev_240x320-roi       register-settings-mt9v02x.txt
colcam-mpeg4
```

The files without extensions are the executable scripts. They all carry out the shell script *func.sh* that does general settings. Moreover they do camera specific settings by using PHYTEC's gstreamer plugin *i2c*, that reads the corresponding parameter file *register-settings-<driver>.txt*.

Start one of the *colcam*-scripts if you have a colour camera or one of the *bwcam*-scripts if you attached a b/w-camera to the board. At first you should try one of the *fbdev_240x320*-scripts in order to get the result being shown on the standard 240x320 display. For a PHYTEC VM-007 or VM-010 colour camera for example, start capturing by simply doing:

```
~# cd gstreamer-examples
~# ./colcam-fbdev_240x320
```

We will get the output:

```
mx2-camera mx2-camera.0: Using EMMA
camera 0-0: Probing 0-0
PCM970 camera: Found GPIO expander on camera, not using it
mx2-camera mx2-camera.0: Camera driver attached to camera 0
mt9m001 0-005d: No MT9M001 chip detected, register read fffffffb
mx2-camera mx2-camera.0: Camera driver detached from camera 0
camera 0-1: Probing 0-1
PCM970 camera: Found GPIO expander on camera, not using it
mx2-camera mx2-camera.0: Camera driver attached to camera 1
mt9v022 0-0048: Detected a MT9V02x chip ID 1313, colour sensor
mx2-camera mx2-camera.0: Camera driver detached from camera 1
camera 0-2: Probing 0-2
PCM970 camera: Found GPIO expander on camera, not using it
mx2-camera mx2-camera.0: Camera driver attached to camera 2
i2c i2c-0: Failed to register i2c client mt9m111 at 0x48 (-16)
mx2-camera mx2-camera.0: Camera driver detached from camera 2
camera 0-3: Probing 0-3
PCM970 camera: Found GPIO expander on camera, not using it
mx2-camera mx2-camera.0: Camera driver attached to camera 3
tw9910: probe of 0-0045 failed with error -22
mx2-camera mx2-camera.0: Camera driver detached from camera 3
mx2-camera mx2-camera.0: MX2 Camera (CSI) driver probed, clock frequency: 26600000
Camera mt9v02x attached.
starting gstreamer ...
Setting pipeline to PAUSED ...
PCM970 camera: Found GPIO expander on camera, not using it
mx2-camera mx2-camera.0: Camera driver attached to camera 1
i2c open /dev/video2
Pipeline is live and does not need PREROLL ...
Setting pipeline to PLAYING ...
New clock: GstSystemClock
```

The occuring error messages can be ignored: The drivers try to probe for a suitable camera module. We will only see them if we access the board through a serial terminal. If everything goes well, there won't be any further messages on the console and after a while the display should start showing camera's picture. We can stop capturing and displaying by pressing the well known Strg-C.

CAUTION When trying to capture for the first time, it might happen that the display will only show some coloured snow. In that case just stop the script and start it again.

The *save*-scripts the OSELAS.BSP-Phytec-phyCORE-i.MX27-PD11.1.0 BSP provide write a single picture into a file with fixed name in the same directory. For example you can copy such a file with ftp to an host in order to watch it there.

The *mpeg4*-scripts can be used for mp4-streaming from a colour camera over an IP-network. You need to adjust the IP-address of the destination within the script. Receiving the stream will be done with something like that:

```
gst-launch udpsrc caps = "application/x-rtp, media=(string)video" !
rtpmp4vdepay ! ffdec_mpeg4 ! decodebin ! ffmpegcolorspace ! ximagesink
```

There are also scripts for connecting to an X-Server, that has been enabled to receive data from the target:

- *bwcam-xv* for b/w camera + X enabled display
- *colcam-xv* for colour camera + X enabled display

Before calling such a script you must declare the IP-address of the X-Server:

```
export DISPLAY=<HOST IP>:0
```

Furthermore there are some subdirectories within the example directory. They provide more GStreamer scripts for dedicated sensors/cameras in order to show sensor specific functions.

More detailed description of the internals of these scripts and the content of the parameter files will be found in our Application Note LAN-052. Currently this AppNote is only available in German. It is located on the LiveSystem in path */opt/PHYTEC_Tools/Documentation*.

5.5.3 Using the Camera Support with Video4Linux 2

GStreamer uses the Video4Linux 2 Interface (V4L2) in order to access the camera drivers. Also for writing your own applications using V4L2 is recommended. This interface offers the possibility to query which functions are being supported (Capability Querying). For a description of all the functionality please refer to the common documentations of V4L2 and to its API Specification.

5.6 Basic Usage of GStreamer

GStreamer is a streaming media framework, based on graphs of filters which operate on media data. GStreamer consists of several command line tools and sets of plugins. Generally there are three kinds of plugins:

Source Plugins These plugins access directly media sources on the system and pass the media data on to further plugins. For example, we can use the v4l2src plugin to access a camera.

Pipe Plugins These plugins are like double ended tubes. We can put data in one side and take the processed data from the other side.

Sink Plugins These plugins take the media data and put them to ”sinks”, which are output devices like framebuffer device, x.org client or even Network sockets.

Using GStreamer on command line is like building a pipeline. To launch the pipeline, we need the command-line tool *gst-launch*.

5.6.1 Simple Usage Example

To get a feeling how this works we can use the built-in fake plugins to build a pipeline, which simply process some empty buffers:

```
~# gst-launch fakesrc num-buffers=1 ! fakesink
```

This will result to output that looks simlar to this:

```
Setting pipeline to PAUSED ...

Pipeline is PREROLLING ...

Pipeline is PREROLLED ...

Setting pipeline to PLAYING ...

New clock: GstSystemClock

Got EOS from element "pipeline0".

Execution ended after 5943451 ns.

Setting pipeline to PAUSED ...

Setting pipeline to READY ...

Setting pipeline to NULL ...

Freeing pipeline ...
```

All plugins are connected with an exclamation mark (!)

If this works properly, we can go on to do some real work in the next section.

5.6.2 Simple Monochrome Usage Example

When the camera drivers have been loaded, the following line will grab the video stream from a monochrome camera like the VM-006 and put it on the framebuffer device:

```
~# gst-launch v4l2src device=/dev/video2 ! video/x-raw-gray,width=1280,height=1024 ! ffmpegcolorspace ! fbdevsink
```

Three plugins are used here:

1. *v4l2src* plugin grabs the raw frames from the camera using the Video4Linux2 API and the given device node.

2. *video/x-raw-gray,width=1280,height=1024* in the pipeline is a capability filter. It sets a mime type to specify a desired video format (in this case grayscale) and the frame size (in this case 1280x1024). It must match to the capabilities of the camera because otherwise you'll get the error message *Could not negotiate format*.

3. *ffmpegcolorspace* takes the stream and converts it to suitable colourspace

4. *fbdevsink* takes the converted stream and displays it on a framebuffer device

5.6.3 Simple Colour Usage Example

If we have a camera with a colour sensor like the VM-009 instead, we can use the following line to process the videostream:

```
~# gst-launch v4l2src device=/dev/video2 ! video/x-raw-rgb,width=1280,height=1024 ! ffmpegcolorspace ! fbdevsink
```

If you use a camera that delivers bayer pattern instead of RGB values like the PHYTEC VM-007-COL or VM-010-COL, there is a plugin named *bayer2rgb* to convert the raw content provided by the camera sensor into an RGB signal:

```
~# gst-launch v4l2src device=/dev/video2 ! video/x-raw-
bayer,width=752,height=480 ! bayer2rgb bg_first=false !
ffmpegcolorspace ! fbdevsink
```

The *bayer2rgb* plugin uses a parameter *bg_first* to define if the first line of the bayer pattern provided by the sensor is a blue/green line or a green/red line. We can try to change this to true if we experience any troubles with our colourspace.

5.7 Advanced Usage of GStreamer

5.7.1 Manually Setting the Frame Rate and Framesize

Along with the mimetype definition we can also optionally set information like frame size and rate. e.g.

```
~# gst-launch v4l2src device=/dev/video2 \      [Enter]
> ! video/x-raw-gray,width=240,height=320,framerate=30/1 \      [Enter]
> ! ffmpegcolorspace ! fbdevsink
```

Doing this will set a fix value for the input frame format. Such options may come handy if we e.g. have trouble with the size of the input stream. Note:

• If we adjust the width and height ourself, the shown region starts at the upperlefter corner of the frame captured by the camera.

• The frame size we can choose depends on the one our camera can provide. If we choose any size which extends the picture range of the camera, GStreamer will fail to start with an output like this:

```
ERROR: from element /pipeline0/v4l2src0: Could not negotiate format
```

Some cameras do not provide their framesize properly to the system or just simply provide a frame bigger than the system can process. In this case the command will fail and we have to define the framesize manually.

5.7.2 Manipulate Input Frame Size with Plugins

Normally the onboard framebuffer device has smaller size than the picture captured by the camera. Because of that, only a part of the captured video might be visible, starting at the upleft corner, on the display. To get the picture we actually want on our display, we can use various plugins:

• we can crop the videosignal using the *videocrop* plugin

```
~# gst-launch v4l2src device=/dev/video2 \     [Enter]
> ! video/x-raw-gray \     [Enter]
> ! videocrop left=250 right=250 top=80 bottom=80 \     [Enter]
> ! ffmpegcolorspace ! fbdevsink
```

Will ”chop out” 432 pixel in the horizontal and 240 pixel in the vertical direction of the input frame, so that we will get a 320x240 sized frame which is positioned in the central of the input frame. This command line will work with a PHYTEC VM-007 (MT9V022) and a VM-010 (MT9V024) camera, which provide a 752(H)x480(V) sized frame at default. We should consult the datasheet of our display and our camera to find out the correct crop parameter for the camera kit.

• If we’d rather prefer not to crop out regions of the input frame, we can use the *videoscale* plugin to resize the input frame

```
~# gst-launch v4l2src device=/dev/video2 ! video/x-raw-gray !
ffmpegcolorspace ! videoscale ! video/x-raw-yuv,width=320,height=240 !
ffmpegcolorspace ! fbdevsink
```

With this command we can resize the video frame to 320x240. Notice:

– Unlike the *videocrop* plugin, *videoscale* cannot process raw data provided by *v4l2src* directly. So we put a *ffmpegcolorspace* between the *v4l2src* and *videoscale*.

– Aspect ration in the resized frame should be proportional to the original size. Otherwise we might get a disorted image.

5.7.3 Manipulate Picture's Orientation

Further we can flip and rotate our video with the *videoflip* plugin. We take the command above as example:

```
~# gst-launch v4l2src device=/dev/video2 \      [Enter]
> ! video/x-raw-gray ! ffmpegcolorspace \     [Enter]
> ! videoscale ! video/x-raw-yuv,width=320,height=240 \     [Enter]
> ! videoflip method=clockwise \     [Enter]
> ! ffmpegcolorspace ! fbdevsink
```

The addtional videoflip plugin in the pipeline flips the video 90 degrees clockwise.

5.7.4 Using other Sinks Than the Framebuffer

Beside the local framebuffer there are additional locations where to show the video stream.

ximagesink and *xvimagesink*

If the support for X protocols is turned on in our OSELAS.BSP-Phytec-phyCORE-i.MX27-PD11.1.0, we can use it to display our camera stream on a remote host, which runs a common X server like X.Org. To do this we run the following steps:

On our host:

```
user@host ~ xhost +
access control disabled, clients can connect from any host
```

CAUTION

Note: With this command we will grant access from any X-Clients in our network to our local X-Server. This could be a security issue. Hence we might probably want to consult our system administrator first before doing this.

On target:

```
~# export DISPLAY=[IP of our host in here]:0
```

Now any X-related application started on our target will be shown on our host and we can start our Gstreamer with the *ximagesink*

```
~# gst-launch v4l2src device=/dev/video2 ! video/x-raw-gray !
ffmpegcolorspace ! ximagesink
```

Notice that displaying our videostream like this we might get very low framerate due to high network load.

udpsink

With GStreamer we can also stream our video with various streaming protocols through the network. However we have to encode the stream with a video codec first. Currently only hardware encoding is supported. If our target system supports such a feature, we can start streaming on the target with following command:

```
~# gst-launch v4l2src device=/dev/video2 ! video/x-raw-gray !
ffmpegcolorspace ! mfw_vpuencoder codec-type=std_avc bitrate=32767
gopsize=10 ! rtph264pay ! udpsink host=[HOST IP ADDRESS] port=5555
```

On the host we can use the following command to decode this video stream:

```
user@host ~ gst-launch udpsrc port=5555 caps = "application/x-rtp,
media=(string)video" ! rtph264depay ! ffdec_h264 ! xvimagesink
```

5.7.5 Using Hardware Encoding for Video Playback

Freescale’s hardware (VPU) encoder plugin *mfw_vpuencoder* can also be used for playing video files. An example is shown in the script *video* you’ll find in directory */usr/bin*. Simply put any mp4-coded avi video file on the phyCORE-i.MX27 and call

```
video filename.avi
```

in order to see and to listen. There is an example avi file named *big_buck_bunny_320x192-MP4-MP3-1m.avi* on the LiveSystem in path */opt/PHYTEC_Tools* that you can put onto the target with ftp.

5.8 SPI Master

The phyCORE-i.MX27 supports an SPI bus, based on the i.MX27’s integrated SPI controller. It is connected to the onboard devices using the standard kernel method, so all methods described here are not special to the phyCORE-i.MX27.

Connected devices can be found in the sysfs at the path */sys/bus/spi/devices*. It depends on the corresponding SPI slave device driver if it provides access to the SPI slave device through this way (sysfs), or any different kind of API.

This BSP currently supports one dedicated SPI bus. Its used to control the MC13783, the main peripheral controller.

5.8.1 SPI PMIC MC13783

The PMIC is used for power management, touch, audio, analogue input and RTC */dev/rtc0*.

With jumper JP29 on the PCM-970 being closed to 2+3 a goldcap capacitor will still supply the PMIC and thus its RTC for several minutes after power off. With JP29 being closed to 1+2 backup supply can be done by connecting a Li-Cell at connector X20. Thus a Li-Cell should typically have 4.6V. For more information refer to the data sheet of MX13783.

The PMIC has an internal timer that determines how long backup supply of other devices like the I^{2}C Realtime Clock Chip RTC8564 will take. Barebox sets this timer to 8 seconds.

Note: Only after this time the phyCORE-i.MX27 can restart again if main power is reconnected.

5.9 I²C Master

The i.MX27 processor based phyCORE-i.MX27 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.

5.9.1 I²C Realtime Clock RTC8564

The RTC8564 clock chip on the phyCORE-i.MX27 can be accessed as */dev/rtc1*. It also has the entry */sys/class/rtc/rtc1* in the sysfs filesystem, where you can read for example the *name*.

5.9.2 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/1-0052/eeprom* and use *fseek()* and *read()* to get the values.

5.10 Real Time Clocks

The Real Time Clock framework of the kernel can access the RTC of the PMIC MC13783 as well as the RTC8564 clock chip using the same tools as for any other real time clock.

Date and time can be manipulated with the *hwclock* tool, using the *-w* (systohc) and *-s* (hctosys) options. By default it will use PMIC's RTC */dev/rtc0*. With option *–f /dev/rtc1* set it will use the RTC8564 clock chip instead. For more information about this tool refer to the manpage of *hwclock*.

OSELAS.BSP-Phytec-phyCORE-i.MX27-PD11.1.0 tries to set up the date at system startup. If there was a powerfail *hwclock* will state:

```
hwclock: RTC_RD_TIME: No data available
```

In this case set the date manually (see *man date*) and run *hwclock -w -u* to store the new date into the RTC.

5.11 Framebuffer

This driver gains access to the display via device node */dev/fb0*. For this BSP the Hitachi TX090 display with a resolution of 240x320 is default. Older kits came with a Sharp LQ035Q7. Selection of this display can be done in the Barebox in script */env/config*. There you will find the lines

```
# Displays

display=TX090

#display=Sharp-LQ035Q7

bootargs="mem=122M console=ttymxc0,115200 video=imxfb:$display"
```

Just change the commentation in order to switch over to the other display.

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.

5.12 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 USB Host Controller

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

The OSELAS.BSP-Phytec-phyCORE-i.MX27-PD11.1.0 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.

5.14 OneWire Interface

Any detected 1W device will be mapped to the *sysfs* filesystem. For example a connected temperature sensor could be accessed via this entry:

```
/sys/bus/w1/devices/10-000801018ed7/w1_slave
```

A simple *cat* command can give you the follwing output:

```
root@phyCORE:~ cat /sys/bus/w1/devices/10-000801018ed7/w1_slave
2e 00 4b 46 ff ff 0e 10 91 : crc=91 YES
2e 00 4b 46 ff ff 0e 10 91 t=22875
```

5.15 MMC/SD Card

The phyCORE-i.MX27 in conjunction with its baseboard and mapper supports Secure Digital Cards and Multi Media Cards to be used as general purpose blockdevices. These devices can be used in the same way as any other blockdevice.

CAUTION

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

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

CAUTION

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.

There are different possibilities of how to handle write-protection of MMC/SD cards that can be configured by boot parameter in Barebox’s environment. If you set the boot argument *mmc_wp=0*, then there’s no write protection at all, thus the card is always writable. By default this argument is set to *1*. Because baseboards of revision up to 1280.3 don’t have write protection recognition, in this case the card is always only readable and never writable.

Baseboards of revision 1280.4 and higher have a jumper JP50. If you leave this jumper open, behaviour is the same than if you have a baseboard of lower revision. If you close this jumper, then the write protect switch of the card will be recognized.

5.16 CAN Bus

The phyCORE-i.MX27 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.phyCORE-i.MX27* instead and rebuild the BSP.

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

```
root@phyCORE:~ 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.

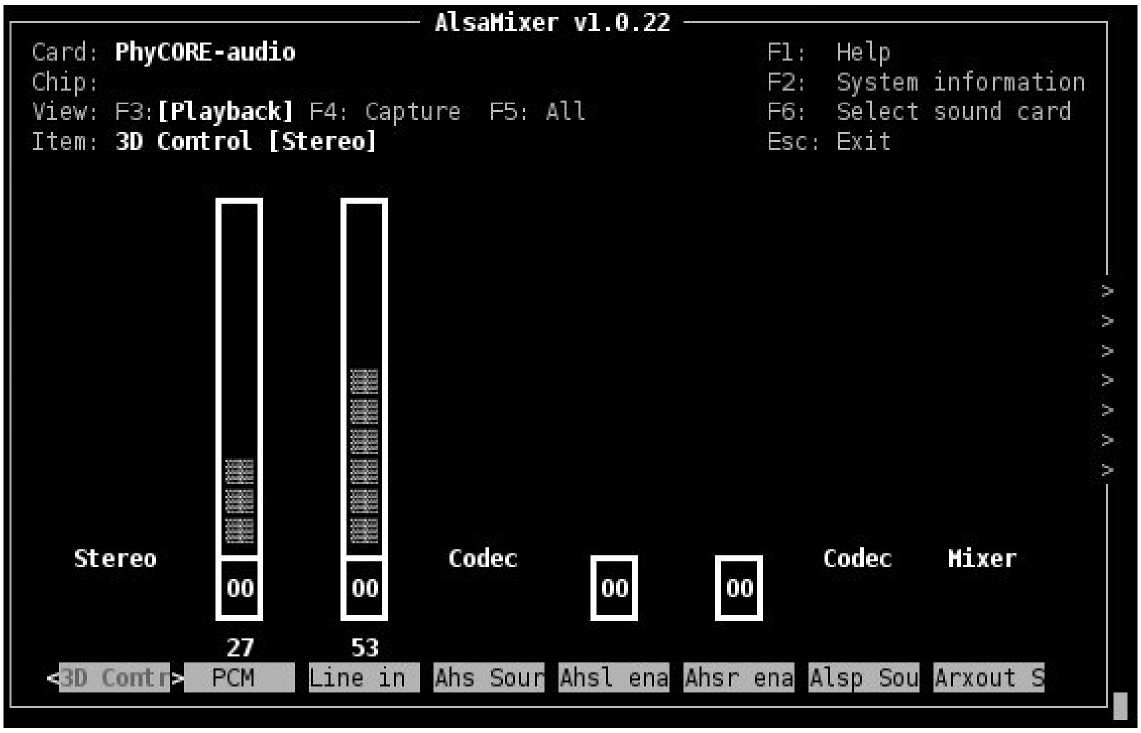
5.17 AUDIO support

Audio support on the module is done via the AC97 interface.

5.17.1 Audio Sources and Sinks

The PCM-970 has three jacks for Line Out, Line In and the Handset microphone can be connected through three jacks.

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.



```
AlsaMixer v1.0.22
Card: PhyCORE-audio                                    F1:  Help
Chip:                                                  F2:  System information
View: F3:[Playback] F4: Capture  F5: All               F6:  Select sound card
Item: 3D Control [Stereo]                              Esc: Exit

   Stereo                    Codec                  Codec   Mixer
             00      00              00      00
             27      53
 <3D Contr>  PCM    Line in  Ahs Sour Ahsl ena Ahsr ena Alsp Sou Arxout S
```

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.

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.

5.17.2 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/sounds/KDE_Startup.wav
```

5.17.3 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
```

Note: Sample rate is restricted by the external PMIC device. Only 8kHz and 16kHz are possible.

Capturing can be stopped again using Strg-C.

5.18 Using Qt

Since PD11.1.0 Nokia's Qtopia is supported by our BSPs. Qt is very commonly used for embedded systems. 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 4.6.3. It's the fluidlauncher, located in */usr/bin/qt4-demos/embedded/fluidlauncher*. Just go there and start it with

```
./fluidlauncher -qws
```

This Qt application will present three other Qt applications: 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 three 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 three 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.

5.19 CPU core frequency scaling

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

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

performance Always selects the highest possible CPU core frequency.

powersave Always selects the lowest possible CPU core frequency.

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

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

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

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

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-i.MX27 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-i.MX27*

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:	Quickstart Manual OSELAS.BSP( ) phyCORE-i.MX27
Document Number:	L-766e_1 **July** 2011

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