BSP-Quickstart phyFLEX-i.MX6

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 (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

2nd Edition: March 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. You should use 32-Bit Ubuntu 12.04 LTS and at first install the following packages:

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

Generally the central place for our BSPs is our ftp-server ftp://ftp.phytec.de/pub/Products/phyFLEX-iMX6. In order to build a BSP you need the appropriate toolchain and you need the build tool PTXdist from our partner Pengutronix. The central place for toolchains is http://www.oselas.com and for PTXdist it is http://www.ptxdist.de. These websites provide all required packages and documentation (at least for software components that are available to the public). Usually you can find a copy of the particular needed PTXdist and toolchain together with the BSP on our ftp-server in the same directory, in order to make things easier.

To build BSP-Phytec-phyFLEX-i.MX6-PD13.1.0, the following archives have to be available on the development host:

- ptxdist-2012.03.0.tar.bz2
- BSP-Phytec-phyFLEX-i.MX6-PD13.1.0.tgz
- OSELAS.Toolchain-2011.11.1.tar.bz2

This BSP does not work with alpha-versions of the phyFLEX-i.MX6 modules, thus those having number 1362.0 or 1362.0a printed on their plates!

1.2 PTXdist

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

1.2.1 Main Parts of PTXdist

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

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

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

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

1.2.2 Extracting the sources

To install PTXdist you need to extract the archive with the PTXdist software *ptxdist-2012.03.0.tar.bz2*.

The PTXdist packet is to be extracted into some temporary directory in order to be built before the installation, for example the *local/* directory in

the user's home. If this directory does not exist, we have to create it and change into it:

```
~# cd
~# mkdir local
~# cd local
```

Next step is to extract the archive:

```
~/local# tar -xjf ptxdist-2012.03.0.tar.bz2
```

If everything goes well, we now have a ptxdist-2012.03.0 directory, so we can change into it:

```
~/local# cd ptxdist-2012.03.0
```

1.2.3 PTXdist installation

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

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

```
~/local/ptxdist-2012.03.0# ./configure
```

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

```
[...]
checking whether /usr/bin/patch will work... yes
configure: creating ./config.status
config.status: creating Makefile
config.status: creating scripts/ptxdist_version.sh
config.status: creating rules/ptxdist-version.in
```

```
ptxdist version 2012.03.0 configured.
Using '/usr/local' for installation prefix.
Report bugs to ptxdist@pengutronix.de
```

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

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

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

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

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

```
~/local/ptxdist-2012.03.0# make
```

All program parts are being compiled, and if there are no errors we can now install PTXdist into it’s final location. In order to write to /usr/local, this step has to be performed as user *root*:

```
~/local/ptxdist-2012.03.0# sudo make install
```

```
[enter root password]
[...]
```

If we don’t have root access to the machine it is also possible to install into some other directory with the *--prefix* option. We need to take care that the *bin/* directory below the new installation dir is added to our *$PATH* environment variable (for example by exporting it in *~/.bashrc*).

The installation is now done, so the temporary folder may now be removed:

```
~/local/ptxdist-2012.03.0# cd
~# rm -rf local
```

1.2.4 Configuring PTXdist

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

Run PTXdist’s setup:

```
~# ptxdist setup
```

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

Proxy Setup

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

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

Source Archive Location

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

1.3 Toolchains

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

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

- `arm-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. It may be a good idea to do that over night, because it may take several hours, depending on the target architecture and development host power.

1.3.1 Building the Toolchain

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

PTXdist handles toolchain building as a simple project, like all other projects, too. So we can download the OSELAS.Toolchain bundle and build the required toolchain for the BSP.

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

Usually the */opt* directory is not world writeable. So in order to build our OSELAS.Toolchain into that directory we need to use a root account to change the permissions. PTXdist detects this case and asks if we want to run *sudo* to do the job for us. Alternatively we can enter:

```
mkdir /opt/OSELAS.Toolchain-2011.11.1
chown <username> /opt/OSELAS.Toolchain-2011.11.1
chmod a+rwx /opt/OSELAS.Toolchain-2011.11.1
```

We recommend to keep this installation path as PTXdist expects the toolchains at */opt*. Whenever we go to select a platform in a project, PTXdist tries to find the right toolchain from the platform configuration settings and a toolchain at */opt* that matches to these settings. But that's for our convenience only. If we decide to install the toolchains at a different location, we still can use the toolchain parameter to define the toolchain to be used on a per project base.

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

```
tar -xjf OSELAS.Toolchain-2011.11.1.tar.bz2
cd OSELAS.Toolchain-2011.11.1
ptxdist select ptxconfigs/\     [Enter]
>           arm-cortexa9-linux-gnueabi_gcc-4.6.2_glibc-2.14.1_binutils-
2.21.1a_kernel-2.6.39-sanitized.ptxconfig
```

```
ptxdist go
```

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

Another possibility is to read the next chapters of this manual, to find out how to start a new project.

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

1.3.2 Protecting the Toolchain

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

1.3.3 Building additional Toolchains

The OSELAS.Toolchain-bundle comes with various predefined toolchains. Refer to the *ptxconfigs/* folder for other definitions. To build additional toolchains we only have to clean our current toolchain project, removing the current *selected_ptxconfig* link and creating a new one.

```
ptxdist clean
rm selected_ptxconfig
ptxdist select \     [Enter]
> ptxconfigs/any_other_toolchain_def.ptxconfig
ptxdist go
```

Chapter 2 Building phyFLEX-i.MX6's BSP

2.1 The Board Support Package

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

```
~# tar -zxf BSP-Phytec-phyFLEX-i.MX6-PD13.1.0.tgz
~# cd BSP-Phytec-phyFLEX-i.MX6-PD13.1.0
```

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

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

projectroot Contains files and configuration for the target's runtime. A running GNU/Linux system uses many text files for runtime configuration. Most of the time the generic files from the PTXdist installation will fit the needs. But if not, customized files are located in this directory.

rules If something special is required to build the BSP for the target it is intended for, then this directory contains these additional rules.

2.2 Selecting a Hardware Platform

Before we can build this BSP, we need to select the target to build for. In this case we want to build for the phyFLEX-i.MX6, so please type:

```
ptxdist select configs/ptxconfig
ptxdist platform configs/phyFLEX-i.MX6/platformconfig
```

You will see:

```
info: selected platformconfig:
'configs/phyFLEX-i.MX6/platformconfig'
```

If PTXdist automatically detects the proper toolchain while selecting the platform, it will also output:

```
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.

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-phyFLEX-i.MX6/images/* directory and the final root filesystem for the target board can be found in the *platform-phyFLEX-i.MX6/root/* directory and a bunch of **.ipk* packets in the *platform-phyFLEX-i.MX6/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.tgz: root files inside a plain gzip compressed tar ball.

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

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

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

```
ptxdist platformconfig
```

Then select the submenu *image creation options*. The generated image will be placed into *platform-phyFLEX-i.MX6/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-phyFLEX-i.MX6/root/* will not be enclosed in the image.

Chapter 3 phyFLEX-i.MX6 Bootloader preparation

This step can be omitted if your phyFLEX-i.MX6 already has the right boot loader: For this BSP the barebox version v2012.02.0 is required.

3.1 Updating Barebox

This BSP does not work with alpha-versions of the phyFLEX-i.MX6 modules, thus those having number 1362.0 or 1362.0a printed on their plates!

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-phyFLEX-i.MX6/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.

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

```
tftp barebox-image
```

After that store the Barebox into the SPI NOR flash:

```
erase /dev/nor0.barebox
cp barebox-image /dev/nor0.barebox
```

The same can be done with one simple command:

```
update -t barebox -d nor -m tftp -f barebox-image
```

Please ensure that the module boots from SPI NOR flash. From the DIP-switch S3 either switch 1 or switch 2 should be set to ON and all others to OFF.

Note that if something goes wrong at this, you don't have any bootloader anymore on your module. In this case you need to boot from an SD-card into Barebox (set DIP-switch S3 to ON-ON-OFF-OFF) and then do:

```
update –t barebox –d nor –m tftp –f barebox-image
```

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 phyFLEX-i.MX6. 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.

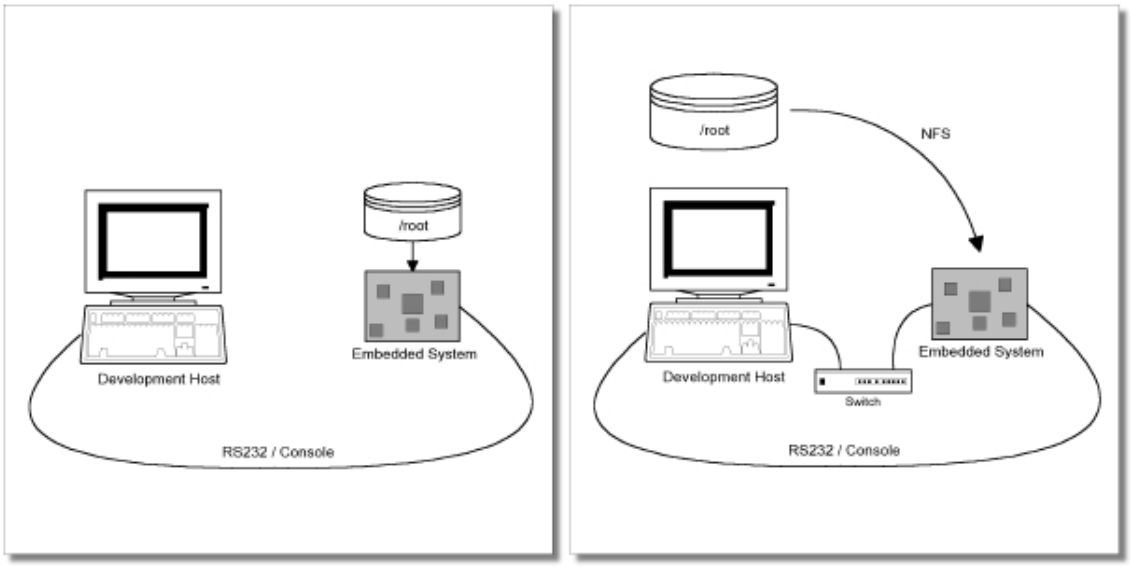

Figure 4.1: Booting the phyFLEX-i.MX6: From its flash or from the host via network.

Figure 4.1 shows both methods. The main method used in the BSP-phyFLEX-i.MX6-PD13.1.0 BSP is to provide all needed components to run on the target itself. The linux kernel image and the root filesystem image are persistent in the media the target features. This means the only connection needed is the nullmodem cable to see what is happening on our target. We call this method *standalone*.

The other method is to provide needed components via network. In this case the development host is connected to the phyFLEX-i.MX6 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-phyFLEX-i.MX6/root/* directory in our PTXdist workspace.

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

4.1 Development Host Preparations

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

If you built your own images, please copy them from the BSP's directory *platform-phyFLEX-i.MX6/images* now to here.

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

4.2 Stand-Alone Booting Linux

To use the the target standalone, the kernel and the rootfs have to be made persistent in the onboard media of the phyFLEX-i.MX6. The following sections describe the steps necessary to bring kernel and rootfs into the onboard NAND type flash.

After that, the phyFLEX-i.MX6 can work independently from the development host. We can "cut" the network (and serial cable) and the phyFLEX-i.MX6 will continue to work.

4.2.1 Preparations on the Embedded Board

The phyFLEX-i.MX6 uses Barebox as its bootloader. Barebox can be customized with environment variables and scripts to support any boot constellation. BSP-Phytec-phyFLEX-i.MX6-PD13.1.0 comes with a predefined environment setup to easily bring up the phyFLEX-i.MX6.

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 *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 the Linux kernel on the target:

```
update -t kernel -d nand –m tftp –f uImage-pfla02
```

In principle Linux's root file system can be flashed into NAND the same way:

```
update -t rootfs -d nand –m tftp –f root-pfla02.ubi
```

But you should know that ubifs keeps erase counters within the NAND in order to be able to balance write cycles equally over all NAND sectors. So if there's already an ubifs on your module and you want to replace it by a new one, using *update* will also erase these erase counters, and this should be avoided. Currently in order to achieve this you need to boot into Linux and flash the ubifs from there. So for one time you need to boot the rootfs from SD card, using the following setting (note that `mmcblk0p2` depends on the partition on the SD card where the rootfs is located):

```
rootfs_loc=disk

rootfs_type=ext3

rootfs_part_linux_dev=mmcblk0p2
```

Then use an SD card in order to boot Linux from there. Note: The default login account is *root* with an empty password. Now use an ftp connection in order to copy image *root.ubi* into it. Finally you can flash the image by typing:

```
ubiformat -y /dev/mtd8 -f root-pfla02.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.

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-phyFLEX-i.MX6* 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 BSP-Phytec-phyFLEX-i.MX6-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*, *nor*, *disk* 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 phyFLEX-i.MX6.

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 phyFLEX-i.MX6. To do so, simply type:

```
boot
```

This command should boot the phyFLEX-i.MX6 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 phyFLEX-i.MX6 starter kit consists of the following individual boards:

1. The phyFLEX-i.MX6 module itself, containing the controller, RAM, flash and several other peripherals.

2. The starter kit baseboard (PBA-B-01).

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-mx6/board-mx6q_phyflex.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 phyFLEX-i.MX6 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 and SPI NOR Flash

The phyFLEX-i.MX6 module comes with NAND and SPI NOR 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 flash partitions can be seen as

- */dev/mtdblock0* (Barebox partition SPI NOR)
- */dev/mtdblock1* (Barebox environment partition SPI NOR)
- */dev/mtdblock2* (Splash screen partition SPI NOR)
- */dev/mtdblock3* (Kernel partition SPI NOR)
- */dev/mtdblock4* (Barebox partition NAND)
- */dev/mtdblock5* (Barebox environment partition NAND)
- */dev/mtdblock6* (Splash screen partition NAND)
- */dev/mtdblock7* (Kernel partition NAND)
- */dev/mtdblock8* (Linux rootfs partition NAND)

Only the */dev/mtdblock8* on the phyFLEX-i.MX6 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 positions and sizes of the partitions are:

```
NOR   0x00000000 - 0x0007FFFF: "Barebox"        /dev/mtdblock0
NOR   0x00080000 - 0x0009FFFF: "Barebox Env"    /dev/mtdblock1
NOR   0x000A0000 - 0x0029FFFF: "Splash"         /dev/mtdblock2
NOR   0x002A0000 - 0x00FFFFFF: "Kernel"         /dev/mtdblock3

NAND  0x00000000 - 0x0007FFFF: "Barebox"        /dev/mtdblock4
NAND  0x00080000 - 0x0009FFFF: "Barebox Env"    /dev/mtdblock5
NAND  0x000A0000 - 0x0029FFFF: "Splash"         /dev/mtdblock6
NAND  0x002A0000 - 0x00A9FFFF: "Kernel"         /dev/mtdblock7
NAND  0x00AA0000 - 0x3FFFFFFF: "File System"    /dev/mtdblock8
```

5.2 Serial TTYs

The i.MX6 SoC supports up to 2 so called UART units. On the phyFLEX-i.MX6 all two UARTs are routed to the Molex connectors. At UART0 connector X51 you'll find *ttyO0* which is the standard console.

5.3 Network

The phyFLEX-i.MX6 module features ethernet, which is being used to provide the *eth0* network interface. The interface offers a standard Linux network port at POE connector X28 which can be programmed using the BSD socket interface.

5.4 SPI Master

The i.MX6 provides five SPI busses. Some of them are available on the PBA-B-01 baseboard, one is used for the SPI NOR.

5.5 I²C Master

The i.MX6 provides three I²C busses. At I²C1 you'll find an EEPROM 24C32. 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/0-0050/eeprom* and use *fseek()* and *read()* to get the values.

5.6 USB Host Controller

The phyFLEX-i.MX6 provides one USB-host and one USB-OTG. Both are USB 2.0. The baseboard has an USB-hub for the host port that already is prepared for USB 3.0.

You can switch over the USB-OTG to host mode by setting jumper JP5.

The BSP-Phytec-phyFLEX-i.MX6-PD13.1.0 includes support for mass storage devices and keyboards. Other USB related device drivers must be enabled in the kernel configuration on demand.

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

5.7 MMC/SD Card

The phyFLEX-i.MX6 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.

If you have a baseboard of version at least PL1364.2 jumper JP13 must be set to 1+2 in order to supply SD0 with power.

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<Y1>
/dev/mmcblk1p<Y2>
```

<Y1> and *<Y2>* count the partition numbers starting from 1 to the max count of partitions on the 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.

Furthermore SD0 can be used for booting if you enter an appropriate formatted SD card and set DIP-switch S3 to ON-ON-OFF-OFF.

5.8 GPIO

For setting and resetting the User-LED “USER_LED_GPIO” on the baseboard just enter

```
cd /sys/class/gpio
echo 56 > export
cd gpio56
echo out > direction
echo 1 > value
echo 0 > value
```

The pins on GPIO-port X54 can be accessed using the following numbers:

Pin 0 - GPIO 136 # used as interrupt for LCD-017
Pin 1 - GPIO 135
Pin 2 - GPIO 114
Pin 3 - GPIO 115
Pin 4 - GPIO 6
Pin 5 - GPIO 9
Pin 6 - GPIO 204 # used as interrupt for LCD-018
Pin 7 - GPIO 205
Pin 8 - GPIO 101
Pin 9 - GPIO 55
Pin 10 - GPIO 56

5.9 SATA

You can attach a SATA hard disc drives at connector X62. Power is provided by connector X61.

The drive will be presented as */dev/sda1* and thus can be mounted with

```
mount /dev/sda1 /mnt
```

5.10 Framebuffer

This driver gains access to the display via device node */dev/fb0* for PHYTEC display connector or */dev/fb2* for HDMI. For this BSP the PHYTEC display connector is default. In order to switch on DVI additionaly please use command

```
echo "0" > /sys/devices/platform/mxc_sdc_fb.1/graphics/fb2/blank
```

The default setup of DVI resolution is 1280x1024 at 60MHz. You can change this in Barebox. In its script */env/config* add the line

```
bootargs="$bootargs video=mxcfb2:1920x1080M,if=RGB24"
```

and adjust the resolution due to your needs.

The BSP is already prepared for use with the PrimeView displays PD050VL1 (640x480), PD035VL1 (640x480), PD104SLF (800x600), PM070WL4 (800x480) and ETM0700G0DH6 (800x480). Selection of this display can be done in the Barebox in script */env/config*. There you will find the lines

```
#Displays
#display=Primeview-PD050VL1
#display=Primeview-PD035VL1
#display=Primeview-PD104SLF
#display=Primeview-PM070WL4
display=ETM0700G0DH6
```

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

```
~# fbtest
```

This will show various pictures on the display.

You can check your framebuffer resolution with the command

```
~# fbset
```

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

5.11 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.

5.12 Using Qt

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

Within the BSP come some demos that show what is possible with Qt version 4.7.4. In order to try them you need a touch display attached to the board.

The demo *fluidlauncher* will start automatically. From command line it can be stopped with

```
systemctl stop qt-demo-startup.service
```

If you want to prevent it from being autostarted, you can do it with

```
systemctl disable qt-demo-startup.service
```

Activating autostart again can be done with

```
systemctl enable qt-demo-startup.service
```

And if you want to start it manually, do

```
systemctl start qt-demo-startup.service
```

More demos are located in */usr/bin/qt4-demos*. Please go into this directory with

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

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

and then start demos with commands like

```
spreadsheet/spreadsheet -qws

chip/chip -qws
```

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

5.13 OpenGL

Additionaly to Qt you have OpenGL support. Two demos of OpenGL can be started with

```
cd /opt/viv_samples/vdk

./tutorial3_es20 –f 2000

./tutorial7 –f 15000
```

5.14 Video player

For playing a demo judt enter

```
gplay filmdemo.avi
```

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 phyFLEX-i.MX6 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 / phyFLEX-i.MX6*

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 phyFLEX-i.MX6
Document Number:	L-777e_2 **March** 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