Yocto / i.MX 6

BSP Manual

Document No.: **L-814e_1**

Release No.: **i.MX 6 PD15.2.0**

Edition: September 2015

	EUROPE	NORTH AMERICA	FRANCE
Address:	PHYTEC Messtechnik GmbH Robert-Koch-Str. 39 D-55129 Mainz GERMANY	PHYTEC America LLC 203 Parfitt Way SW Bainbridge Island, WA 98110 USA	PHYTEC France 17, place Saint-Etienne F-72140 Sillé-le-Guillaume FRANCE
Sales:	+49 6131 9221-32 sales@phytec.de	+1 800 278-9913 sales@phytec.com	+33 2 43 29 22 33 info@phytec.fr
Technical Support:	+49 6131 9221-31 support@phytec.de	+1 206 780-9047 support@phytec.com	 support@phytec.fr
Fax:	+49 6131 9221-33	+1 206 780-9135	+33 2 43 29 22 34
Web Site:	http://www.phytec.de http://www.phytec.eu	http://www.phytec.com	http://www.phytec.fr

	INDIA	CHINA
Address:	PHYTEC Embedded Pvt. Ltd. #16/9C, 3rd Main, 3rd Floor, 8th Block, Opp. Police Station Koramangala, Bangalore-560095 INDIA	PHYTEC Information Technology (Shenzhen) Co. Ltd. Suite 2611, Floor 26, Anlian Plaza, 4018 Jin Tian Road Futian District, Shenzhen CHINA 518026
Sales:	+91-80-4086 7046/48 sales@phytec.in	+86-755-3395-5875 sales@phytec.cn
Technical Support:	+91-80-4086 7047 support@phytec.in	 support@phytec.cn
Fax:		+86-755-3395-5999
Web Site:	http://www.phytec.in	http://www.phytec.cn

1st Edition September 2015

Yocto/i.MX6 BSP Manual

Introduction Yocto

Please read the Yocto Reference Manual for a better understanding of Yocto and this BSP.

Introduction BSP

Supported hardware

Phytec's i.MX6 unified BSP contains support for the following boards and

modules. See BSP-Yocto_Machines PD15.1.1, or PD15.2.0.

Building the BSP

This section will guide you through the general build process of the unified i.MX6 BSP using the phyLinux script. But if you want to use our software without phyLinux and the repo tool managed environment, you can find all git repositories on

```
git://git.phytec.de
```

Used Barebox repository:

```
git://git.phytec.de/barebox
```

Our barebox version is based the barebox mainline and adds only a few patches which will be send upstream in future.

Used Linux kernel repository:

```
git://git.phytec.de/linux-mainline
```

Our i.MX6 Kernel is based on the linux stable kernel. The stable Kernel repository can be found at:

```
git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git
```

To find out which tag is used for a specific board, have a look at:

```
meta-phytec/meta-phyimx6/recipes-bsp/barebox/barebox_*.bb
meta-phytec/meta-phyimx6/recipes-kernel/linux/linux-mainline_*.bb
```

Get the BSP

- Create a fresh project directory, e.g.

```
mkdir ~/yocto
```

- Download and run the phyLinux script

```
cd ~/yocto
wget ftp://ftp.phytec.de/pub/Software/Linux/Yocto/Tools/phyLinux
chmod +x phyLinux
./phyLinux init
```

Basic set-up

There are a few important steps which have to be done, before the main build process.

- Setting up the host, see Yocto Reference Manual, Setting up the host
- Setting up the git configuration, see Yocto Reference Manual, Git Configuration

Finding the right software platform

The i.MX6 BSP is a unified BSP, which means it supports a set of different PHYTEC carrier boards with different SOMs. Sometimes it is not easy to find the right software for your PHYTEC board. So if you need to figure out the corresponding machine name of your board, have a look at the Yocto/SupportChart or at the files in:

```
meta-phytec/meta-phyimx6/conf/machine/*.conf
```

There you can find the platform name to the corresponding product ids. All those informations are also displayed by the phyLinux script.

Example, *phyflex-imx6-1.conf* machine configuration file:

```
#@TYPE: Machine
#@NAME: phyflex-imx6-1
#@DESCRIPTION: PFL-A-02-23237E0.A1/PBA-B-01 (i.MX6 Quad, 1GB RAM on two banks, 16MB NOR)
```

Machine *phyflex-imx6-1.conf* represents the PBA-B-01 (Kit) CB with PFL-A-02-23237E0.A1 SOM.

Selecting a software platform

Call

```
./phyLinux init
```

to select the correct SoC, BSP version and platform. It is also possible to pass those information directly over command line parameters:

```
./phyLinux init -p imx6 -r PD15.1-rc1 -m phyflex-imx6-1   # Just an example. You mostly want to select another relase and machine.
```

Please read the phyLinux init documentation for more informations.

Starting the build process

See Yocto Reference Manual, Start the build.

BSP images

All images generated by bitbake are deployed to yocto/build/deploy/images/<machine>

Example: Generated files for the i.MX6 SoC, *phyflex-imx6-1 machine*:

- **Barebox:** barebox.bin
- **Barebox configuration:** barebox-defconfig
- **Kernel:** zImage-phyflex-imx6-1.bin
- **Kernel device tree file:** zImage-imx6q-phytec-pbab01.dtb
- **Kernel configuration:** zImage-defconfig
- **Root Filesystem:** phytec-qt5demo-image-phyflex-imx6-1.tar.gz, phytec-qt5demo-image-phyflex-imx6-1.ubifs, phytec-qt5demo-image-phyflex-imx6-1.ext3
- **SD card image:** phytec-qt5demo-image-phyflex-imx6-1.sdcard

Booting the System

The default boot source for the i.MX6 modules like phyFLEX-i.MX6 and phyCARD-i.MX6 is the NAND flash. The easiest way to get started with your freshly created images, is writing them onto a SD card and select the boot jumpers accordingly. How set the correct boot jumpers for the available carrier boards check: -> i.MX6 bootjumper configurations

Booting from NAND

NAND is the default boot source. To update the Software of the NAND flash see 'Updating the Software'

Booting from SD card

Booting from SD card is useful in several situations, e.g. if the board does not start any more due to a damaged bootloader. The SD card has to be formatted in a special way, so that the ROM Loader of the i.MX6 is able to find the bootloader.

There are two ways to create a bootable SD card. First of all, the yocto build drops out a SD card image which can be easily copied to SD card. You can find images on our ftp Server [1] or use the ones created with a yocto build under build/deploy/images/<MACHINE>/<IMAGENAME>-<MACHINE>.sdcard . The image is copied to the SD card with dd:

```
sudo dd if=<IMAGENAME>-<MACHINE>.sdcard of=/dev/<your_device> conv=fsync
```

where <your_device> could be "sde" for example, depending on your system. Be very careful when selecting the correct drive! You have been warned. The parameter *conv=fsync* forces a sync operaton onto the device before dd returns. This ensures that all blocks are written to the sdcard and aren't still in memory.

The sdcard image contains already all BSP files in correct formatted partitions.

Creating a SD card manually

It is also possible to make these steps manually.

A new card must be setup with 2 partitions and 8MB of free space at the beginning of the card. Use the following procedure with fdisk under Linux:

- Create a new fat partition with partition id C. When creating the new partition you must leave 8MB of free space at the beginning of the card. When you go through the process of creating a new partition, fdisk lets you specify where the first sector starts. During this process fdisk will tell you where the first sector on the disk begins. If the first sector begins at 1000 for example, and each sector is 512 bytes in size, then 8MB / 512 bytes = 16384 sectors ==> your first sector should begin at 17384 to leave 8MB of free space. The size of this fat partition only needs to be big enough to hold the zImage which is only a few megabytes. To be safe, make this partition 64MB in size.
- Create a new Linux partition with partition id 83. Make sure you start this partition after the last sector of partition 1! By default fdisk will try to use the first partition available on the disk, which in this example is 10000. However, this is our reserved space! You can use the remaining portion of the card for this partition.

Write the new partition to the SD card and exit fdisk.

Example:

```
sudo fdisk -l /dev/sde

Platte /dev/sde: 2013 MByte, 2013265920 Byte
4 Köpfe, 32 Sektoren/Spur, 30720 Zylinder, zusammen 3932160 Sektoren
Einheiten = Sektoren von 1 × 512 = 512 Bytes
Sector size (logical/physical): 512 bytes / 512 bytes
```

```
I/O size (minimum/optimal): 512 bytes / 512 bytes
Festplattenidentifikation: 0x77e2079d

   Gerät  boot.     Anfang        Ende     Blöcke   Id  System
/dev/sde1            18432      149503       65536    c  W95 FAT32 (LBA)
/dev/sde2           149504     3932159     1891328   83  Linux
```

Make a filesystem on the partitions like so (note, you may need to remove and reinsert the card for Linux to recognize the new partitions created in the previous step):

```
sudo mkfs.vfat /dev/sde1              # replace 'sde' with your device
sudo mkfs.ext3 -L "rootfs" /dev/sde2  # replace 'sde' with your device
```

Write the bootloader in front of the first partition

```
sudo dd if=barebox.bin of=/dev/sde bs=512 skip=2 seek=2 conv=fsync  # replace 'sde' with your device
```

Afterwards the images need to be copied to the SD card. Make shure that they are named correct on the SD card. Copy the zImage and oftree file to the first vfat partition.

In case that you want to boot the whole Linux from SD card, unpack the files for the root filesystem to partition rootfs:

```
sudo mount /dev/sde2 /media      # replace 'sde' with your device
sudo tar xf <IMAGENAME>-<MACHINE>.tar.gz -C /media/
sudo umount /media
```

Booting from SPI NOR flash

Boards with phyFLEX-i.MX6 modules have SPI NOR flashes optionally mounted. To boot from them, select the correct boot mode on the baseboard. Please check the correct setting for your board here. The SPI Flash is usually quite small so that only the bootloader, kernel and, if needed, a devicetree fit on it. The rootfs is taken from NAND flash on default. How to flash the SPI NOR is described in 'Updating the Software' .

Booting from network

Booting from network means to load the kernel over TFTP and the rootfs over NFS. The bootloader itself has to be still loaded from any other available boot device.

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 build directory to /tftpboot now.

You 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.

Next to the TFTP server a NFS server needs to be installed, too. The NFS server is not restricted to a certain filesystem location, so all you have to do on most distributions is to modify the file /etc/exports and export your 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.

Preparations on the Embedded Board

To find out the ethernet settings in the bootloader of the target type:

```
ifup eth0
devinfo eth0
```

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

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

If you need to change something, type

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

Edit the settings, save them by leaving the editor with Ctrl-D, then type saveenv. If you don't want to save your changes, leave the editor with Ctrl-C.

Setting up paths for tftp and nfs, can be done in the file

```
edit /env/boot/net
```

Booting the Embedded Board

Booting from network can be then done with calling

```
boot net
```

or restart the board and stop autoboot by pressing m. You'll get a menu:

```
Main menu
1: Boot default
2: Detect bootsources
3: Settings
4: Save environment
5: Shell
6: Reset
```

Press 2 and then the enter key

```
boot
1: mmc
2: nand
3: spi
4: net
5: back
```

Press 4 and then the enter key in order to boot the board from network.

Custom boot setup

You may have custom boot requirements that are not covered by the four available boot files (nand, net, mmc, spi). If this is the case you can create your own custom boot entry specifying the kernel and root filesystem location. In Barebox, create your own boot entry, for example named custom:

```
edit /env/boot/custom
```

Use the following template to specify the location of the Linux kernel and root filesystem. Please note that the text in <> such as <kernel_loc_bootm.image>, <rootfs_loc_dyn.root>, and <nfs_root_path> are intended to be replaced with user specified values.

```
#!/bin/sh

global.bootm.image="<kernel_loc_bootm.image>"
global.bootm.oftree="<dts_loc_bootm.oftree>"

nfsroot="<nfs_root_path>"
bootargs-ip
/env/config-expansions

global.linux.bootargs.dyn.root="<rootfs_loc_dyn.root>"
```

- <kernel_loc_bootm.image> Specifies the location of the Linux kernel image

```
/dev/nand0.kernel.bb - To boot the Linux kernel from NAND
${path}/linuximage - To boot the Linux kernel via TFTP
/boot/linuximage - To boot the Linux kernel from SD/MMC card
```

- <dts_loc_bootm.oftree> Specifies the location of the device tree binary

```
/dev/nand0.oftree.bb - To boot the device tree binary from NAND
${path}/oftree - To boot the device tree binary via TFTP
/boot/oftree - To boot the device tree binary from SD/MMC card
```

- <rootfs_loc_dyn.root> Specifies the location of the root filesystem

```
root=ubi0:root ubi.mtd=root rootfstype=ubifs - To mount the root filesystem from NAND
root=/dev/nfs nfsroot=$nfsroot,vers=3,udp rw consoleblank=0 - To mount the root filesystem via NFS
root=/dev/mmcblk0p2 rootwait - To mount the root filesystem from SD/MMC card
```

- <nfs_root_path> Only required if mounting the root filesystem from NFS. Replace with the following:

```
nfsroot="/home/${global.user}/nfsroot/${global.hostname}"
```

Once complete with file modifications exit the editor using CTRL+D. Save the environment:

```
saveenv
```

To run your custom boot entry from the Barebox shell:

```
boot custom
```

If you want to configure the bootloader for always booting from custom, you need to create the */env/nv/boot.default* file. Here you can just insert *custom* and save it. Otherwise the bootsource and bootorder is defined in:

```
/env/init/bootsource
```

USB-OTG Boot (Serial Downloader)

The i.MX6 ROM code is capable of downloading a bootloader from the USB-OTG interface (*Serial Downloader* in the i.MX6 Manual). That's useful for a last resort recovery of a broken on device bootloader or for rapid barebox development.

First you have to compile the program *imx-usb-loader* in the barebox source directory. You can use any current mainline barebox version and you have to enable *System Type ---> i.MX specific settings ---> compile imx-usb-loader*.

```
make ARCH=arm imx_v7_defconfig
make ARCH=arm menuconfig    # activate config symbol CONFIG_ARCH_IMX_USBLOADER
make ARCH=arm CROSS_COMPILE=<prefix of your arm cross toolchain> scripts/imx/
```

Now the tool is in *scripts/imx/imx-usb-loader*.

To load the bootloader onto the module execute the sequence:

1. Connect your target to your development machine via USB (Use the correct usb-port for your board)
2. Set the correct bootjumpers, so the ROM code enters *Serial Downloader* mode (See below)
3. Replug the power to perform a hardreset (boot jumpers are not read on soft reset)
4. Execute *imx-usb-loader*. Example

```
sudo scripts/imx/imx-usb-loader
images/barebox-phytec-pbab01-1gib-1bank.img
```

After that a running barebox should welcome you on the serial console.

TODOs:

- Add jumper settings for phyCARD, phyFLEX and phyCORE-i.MX6
- add table of usb-otg ports for all boards

Updating the Software

In this chapter we explain how to update the Images over the Barebox bootloader into NAND and SPI NOR flash.

Updating from network

i.MX6 boards that have a ethernet connector can be updated over network. Be sure to set up the development host correct. The IP needs to be set to 192.168.3.10 and netmask 255.255.255.0 and a tftp Server needs to be available.

Boot the system over any available boot mode. Press any key to stop autoboot, then type:

```
ifup eth0
devinfo eth0
```

The ethernet interfaces should be configured like this:

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

If you need to change something, type

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

Edit the settings, save them by leaving the editor with Ctrl-D, then type

```
saveenv
```

and reboot the board. Otherwise leave the editor with Ctrl-C.

Update NAND Flash over network

To update the bootloader you may use the barebox_update command. This provides a handler which automatically erases and flashes two copies of barebox image in the NAND flash. This handler also creates a FCB table in the NAND. The FCB table is needed from the ROM to boot from NAND

```
barebox_update -t nand /mnt/tftp/barebox.bin

erase /dev/nand0.barebox-environment.bb       # optionally erase the environment and use the default barebox environment
```

If you have erased the environment, you have to reset your board. Otherwise the barebox still uses the old environment.

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

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

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

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

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

Update SPI-NOR Flash over network

To update the second stage bootloader in SPI over network do the following:

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

erase /dev/m25p0.barebox-environment        # optionally erase the environment and use the barebox default environment
```

If you have erased the environment, you have to reset your board. Otherwise the barebox still uses the old environment.

The kernel and oftree are then updated with the regular erase and cp commands:

```
erase /dev/m25p0.kernel
cp /mnt/tftp/zImage /dev/m25p0.kernel
erase /dev/m25p0.oftree
cp /mnt/tftp/oftree /dev/m25p0.oftree
```

The root file system is to big to fit into the SPI NOR flash. So the default set up when booting from SPI is to take the rootfs from NAND flash.

Updating from SD card

To update a i.MX6 board from SD card some requisites need to be taken.

The kernel and device tree is already in the first partition of the SD card, because they are needed for the SD card boot. If you want to update the barebox in the NAND or NOR, too, you have to copy the *barebox.bin* onto the SD-Card on your host PC.

Note: The actual bootloader on the SD card is not in a partition. It's located before the first partition after the partition table.

Note: You cannot update the rootfs, because the first partition is too small for it.

Update NAND flash over SD-Card

To update the images the same commands as updating from tftp are basically used with adapted path parameters:

```
barebox_update -t nand /mnt/mmc/barebox.bin
erase /dev/nand0.barebox-environment.bb       # optionally erase the environment and use the default barebox environment
erase /dev/nand0.kernel.bb
cp /mnt/mmc/zImage /dev/nand0.kernel.bb
erase /dev/nand0.oftree.bb
cp /mnt/mmc/oftree /dev/nand0.oftree.bb
```

Then switch to bootjumpers to NAND boot and reset your board.

Update SPI flash over SD Card

To update the images the same commands as updating from tftp are basically used with adapted path parameters:

```
cp /mnt/mmc/barebox.bin /dev/m25p0.barebox
erase /dev/m25p0.barebox-environment        # optionally erase the environment and use the barebox default environment
erase /dev/m25p0.kernel
cp /mnt/mmc/zImage /dev/m25p0.kernel
erase /dev/m25p0.oftree
cp /mnt/mmc/oftree /dev/m25p0.oftree
```

Then switch to bootjumpers to NOR boot and reset your board.

Device Tree (DT) introduction

Copied from the linux kernel, see

```
linux/Documentation/devicetree/usage-model.txt
```

The "Open Firmware Device Tree", or simply Device Tree (DT), is a data structure and language for describing hardware. More specifically, it is a description of hardware that is readable by an operating system so that the operating system doesn't need to hard code details of the machine.

Structurally, the DT is a tree, or acyclic graph with named nodes, and nodes may have an arbitrary number of named properties encapsulating arbitrary data. A mechanism also exists to create arbitrary links from one node to another outside of the natural tree structure.

Conceptually, a common set of usage conventions, called 'bindings', is defined for how data should appear in the tree to describe typical hardware characteristics including data busses, interrupt lines, GPIO connections, and peripheral devices.

The kernel is a really good source for a DT introduction. An overview of the device tree data format can be found at http://devicetree.org/Device_Tree_Usage .

PHYTEC i.MX6 BSP device tree concept

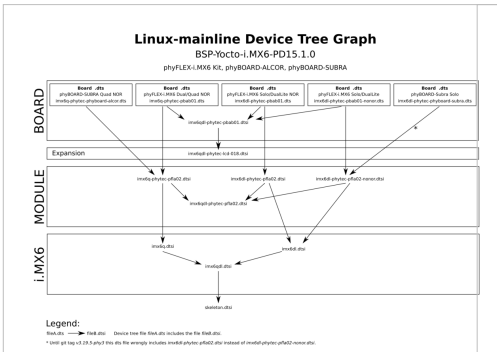
To cope with the variations of modules and baseboards the device tree for the Phytec i.MX6 Platform is structured in certain way.

Device Tree Structure

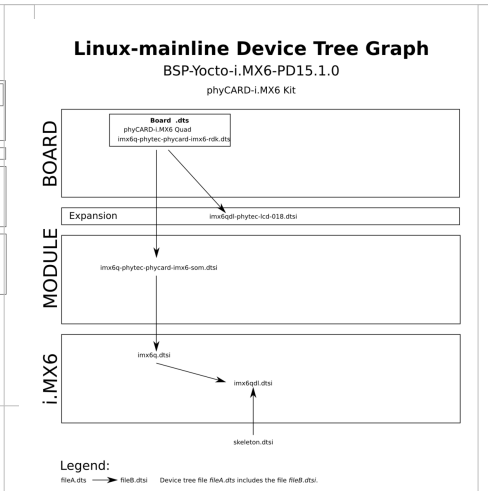
The device tree files are divided roughly into three layers: the SoC layer, the module layer and the baseboard layer. This resembles the physical properties of the hardware. For example, the same phyFLEX-i.MX6 module can be used on the phyFLEX Carrier Board, the phyBOARD-ALCOR and the phyBOARD-SUBRA. In each layer there are multiple device tree include files, which are include by another layer.

Furthermore common functionality within a layer is factored out into an extra device tree include file. So, for example, the module settings, which are the same for Solo/DualLite and Dual/Quad, are placed into *imx6qdl-phytec-pfla02.dtsi*. The file is included by the specific Solo/DualLite module file *imx6dl-phytec-pfla02.dtsi* and by the specific Dual/Quad module file *imx6q-phytec-pfla02.dtsi*.

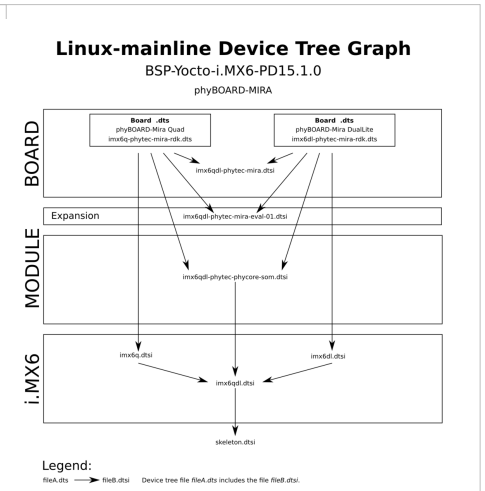
An overview of the device tree hierarchy for all Phytec i.MX6 platforms is presented in the table below.



Device Tree BSP-Yocto-i.MX6-PD15.1.0 phyFLEX



Device Tree BSP-Yocto-i.MX6-PD15.1.0 phyCARD



Device Tree BSP-Yocto-i.MX6-PD15.1.0 phyCORE

There are some special device tree files which don't fit into the layer scheme. The *imx6qdl-phytec-lcd-018.dtsi* is an example. It doesn't belong to a baseboard nor to a module. It contains the configuration of a display and a backlight which can be connected to the phyFLEX or phyCARD Carrier Board.

Bootloader's DT modifications

The bootloader loads the device tree blob for the kernel from the boot device. That's maybe a partition on NOR or NAND or from the SD card. It doesn't pass the blob unmodified to the kernel, but tweaks the device tree in some parts.

For example it will modify the memory node, which contains the total amount of available memory of the system. Thus we don't have to create a extra device tree for each memory size. So hence the memory size is set to zero in the device tree file, it works as expected at runtime.

Snippet from *imx6qdl-phytec-pfla02.dtsi*:

```
        memory {
                reg = <0x0 0x0>; /* will be filled by bootloader */
        };
```

Changing NOR and NAND partitions

The bootloader itself contains also a device tree in the bootloader image which differs from the one used by the kernel. Some changes require to changed both device trees: kernel and barebox device tree.

For example changing the NAND partition layout: For the phyFLEX-i.MX6 you have to make your changes in barebox's device tree *arch/arm/dts/imx6qdl-phytec-pfla02.dtsi* in node *gpmi* and in linux's device tree *arch/arm/boot/dts/imx6qdl-phytec-pfla02.dtsi* also in node *gpmi*.

Note: The unified BSP contains device tree sources for the barebox bootloader and for the linux kernel. The bootloader device tree holds only the absolutely necessary hardware description for the basic board bring-up. Make sure you are working with the right device tree.

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.

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 device trees (DT).

```
linux/arch/arm/boot/dts/*.dts*
```

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.

The hole board specific hardware is described in DTs and is not part of the kernel image itself. The hardware description is in an own separate binary, called device tree blob (DTB).

The following sections provide an overview of the supported hardware components and their operating system drivers on the i.MX6 Platform.

i.MX6 pin muxing

The i.MX6 has more internal functional blocks and signals than pads. Therefore only a subset of the available functionality can be exposed and connected to external devices. Selecting which signals of an internal block is connected to the pad is called *muxing*.

The muxing is done in the kernel device tree. This information is passed to the IOMUX Controller of the i.MX6 to setup the selected mux options.

You should consult the Freescale Reference Manual of the i.MX6 for more information about the specific pins and the muxing capabilites:

- Solo and DualLite: i.MX6 DualLite Documentation [2] -> Reference Manual *IMX6SDLRM.pdf*
- Quad: i.MX6 Quad Documentation [3] -> Reference Manual *IMX6DQRM.pdf*

Here is an example of the pin muxing of the uart4 device in *imx6qdl-phytec-pfla02.dtsi*.

```
pinctrl_uart4: uart4grp {
        fsl,pins = <
                MX6QDL_PAD_KEY_COL0__UART4_TX_DATA       0x1b0b1
                MX6QDL_PAD_KEY_ROW0__UART4_RX_DATA       0x1b0b1
        >;
};
```

The macro on the left is defined in arch/arm/boot/dts/imx6q-pinfunc.h [4] or arch/arm/boot/dts/imx6dl-pinfunc.h [5]. The first part of the string is the pad (*PAD_KEY_COL0*) and the second part of the string (*UART4_TX_DATA*) is one of the available mux options for this pad.

The pad setting value (hex value on the right) is explained in Documentation/devicetree/bindings/pinctrl/fsl,imx6q-pinctrl.txt [6]. Here the pad setting value *0x1b0b1* means: PAD_CTL_HYS, PAD_CTL_SRE_SLOW, PAD_CTL_DSE_40ohm, PAD_CTL_SPEED_MED, PAD_CTL_PUS_100K_UP, PAD_CTL_PUE and PAD_CTL_PKE.

Serial TTYs

The i.MX6 SoC supports up to 5 so called UART units. PHYTEC boards implement different amounts of UARTs. The PHYTEC i.MX6 Board use different standard console outputs

board	standard console output
phyFLEX-i.MX6	ttymxc3
phyCARD-i.MX6	ttymxc2
phyBOARD-ALCOR	ttymxc3
phyBOARD-SUBRA	ttymxc3

If you are at the command line prompt of Linux userspace, then you can easily access other UART devices:

```
echo "test" > /dev/ttymxc2
```

Be sure that the baudrate is set correctly on host and target side. You can use the command *stty* on the target for that. Here is an example to copy all serial settings from *ttymxc3* (your working serial console) to *ttymxc2*.

```
# get current parameters
$ stty -F /dev/ttymxc3 -g
5500:5:1cb2:a3b:3:1c:7f:15:4:0:1:0:11:13:1a:0:12:f:17:16:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0
# copy output from stty command above to argument of next command
```

```
$ stty -F /dev/ttymxc2 5500:5:1cb2:a3b:3:1c:7f:15:4:0:1:0:11:13:1a:0:12:f:17:16:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0
```

Here is an example from *imx6qdl-phytec-pfla02.dtsi*

```
&iomuxc {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_hog>;

        imx6q-phytec-pfla02 {
                pinctrl_uart3: uart3grp {
                        fsl,pins = <
                                MX6QDL_PAD_EIM_D24__UART3_TX_DATA
       0x1b0b1
                                MX6QDL_PAD_EIM_D25__UART3_RX_DATA
       0x1b0b1
                                MX6QDL_PAD_EIM_D30__UART3_RTS_B
         0x1b0b1
                                MX6QDL_PAD_EIM_D31__UART3_CTS_B
         0x1b0b1
                        >;
                };

                pinctrl_uart4: uart4grp {
                        fsl,pins = <
                                MX6QDL_PAD_KEY_COL0__UART4_TX_DATA
      0x1b0b1
                                MX6QDL_PAD_KEY_ROW0__UART4_RX_DATA
      0x1b0b1
                        >;
                };
        };
};

&uart3 {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_uart3>;
        fsl,uart-has-rtscts;
        status = "disabled";
};

&uart4 {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_uart4>;
        status = "disabled";
};
```

Network

Different modules and different boards support different ethernet features (e.g.: 1 x 10/100Mbit or 1 x 1Gbit).

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

The network configuration is handled by the systemd-networkd daemon. The relevant configuration files can be found on the target in */lib/systemd/network/*. And also in the BSP: *meta-yogurt/recipes-core/systemd/systemd/*

IP addresses can be configured within **.network* files. The default IP address and netmask of eth0 is

```
eth0: 192.168.3.11/24
```

Example: phyFLEX-i.MX6 *imx6qdl-phytec-pfla02.dtsi*

```
&fec {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_enet>;
        phy-handle = <&ethphy>;
        phy-mode = "rgmii";
        phy-reset-gpios = <&gpio3 23 GPIO_ACTIVE_LOW>;
        phy-supply = <&vdd_eth_io_reg>;
        status = "disabled";

        mdio {
                #address-cells = <1>;
                #size-cells = <0>;

                ethphy: ethernet-phy@3 {
                        reg = <3>;
                        txc-skew-ps = <1680>;
                        rxc-skew-ps = <1860>;
                };
        };
};

&iomuxc {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_hog>;

        imx6q-phytec-pfla02 {
                pinctrl_enet: enetgrp {
                        fsl,pins = <
                                MX6QDL_PAD_ENET_MDIO__ENET_MDIO
         0x1b0b0
                                MX6QDL_PAD_ENET_MDC__ENET_MDC
           0x1b0b0
                                MX6QDL_PAD_RGMII_TXC__RGMII_TXC
         0x1b0b0
                                MX6QDL_PAD_RGMII_TD0__RGMII_TD0
         0x1b0b0
                                MX6QDL_PAD_RGMII_TD1__RGMII_TD1
```

```
        0x1b0b0
                                MX6QDL_PAD_RGMII_TD2__RGMII_TD2
        0x1b0b0
                                MX6QDL_PAD_RGMII_TD3__RGMII_TD3
        0x1b0b0
                                MX6QDL_PAD_RGMII_TX_CTL__RGMII_TX_CTL
   0x1b0b0
                                MX6QDL_PAD_ENET_REF_CLK__ENET_TX_CLK
    0x1b0b0
                                MX6QDL_PAD_RGMII_RXC__RGMII_RXC
        0x1b0b0
                                MX6QDL_PAD_RGMII_RD0__RGMII_RD0
        0x1b0b0
                                MX6QDL_PAD_RGMII_RD1__RGMII_RD1
        0x1b0b0
                                MX6QDL_PAD_RGMII_RD2__RGMII_RD2
        0x1b0b0
                                MX6QDL_PAD_RGMII_RD3__RGMII_RD3
        0x1b0b0
                                MX6QDL_PAD_RGMII_RX_CTL__RGMII_RX_CTL
   0x1b0b0
                                MX6QDL_PAD_ENET_TX_EN__ENET_TX_EN
       0x1b0b0
                        >;
                };
         };
};
```

CAN Bus

The phyFLEX-i.MX6 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.

The information for can0 looks like

```
# ifconfig can0
can0      Link encap:UNSPEC  HWaddr
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
```

```
          UP RUNNING NOARP  MTU:16  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:10
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:71
```

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

Configuration happens within the systemd configuration file */lib/systemd/system/can0.service*.

For a persistent change of the default bitrates change configuration in *meta-yogurt/recipes-core/systemd/systemd/can0.service* instead and rebuild the rootfs.

```
[Unit]
Description=can0 interface setup

[Service]
Type=simple
RemainAfterExit=yes
ExecStart=/sbin/ip link set can0 up type can bitrate 500000
ExecStop=/sbin/ip link set can0 down

[Install]
WantedBy=basic.target
```

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

```
cansend can0 123#45.67
candump can0
```

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

The corresponding kernel part can be found within the board specific DT, e.g. *arch/arm/boot/dts/imx6qdl-phytec-pfla02.dtsi*

```
&can1 {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_flexcan1>;
        status = "disabled";
```

```
};

&iomuxc {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_hog>;

        imx6q-phytec-pfla02 {
                pinctrl_flexcan1: flexcan1grp {
                        fsl,pins = <
                                MX6QDL_PAD_KEY_ROW2__FLEXCAN1_RX
        0x1b0b0
                                MX6QDL_PAD_KEY_COL2__FLEXCAN1_TX
        0x1b0b0
                        >;
                };
        };
};
```

MMC/SD Card

All i.MX6 kits support a slot for Secure Digital Cards and Multi Media Cards to be used as general purpose block devices. The phyFLEX-i.MX6 supports two slots for Secure Digital Cards and Multi Media Cards. These devices can be used in the same way as any other block device.

These kind of devices are hot pluggable, so you must pay attention not to unplug the device while it's still mounted. This may result in data loss. After inserting an MMC/SD card, the kernel will generate new device nodes in /dev/. The full device can be reached via its /dev/mmcblk0 device node, MMC/SD card partitions will occur in the following way:

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

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

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

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

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

DT configuration for the MMC interface:

```
&iomuxc {
        imx6q-phytec-pfla02 {

                pinctrl_usdhc2: usdhc2grp {
                        fsl,pins = <
                                MX6QDL_PAD_SD2_CMD__SD2_CMD
             0x170f9
                                MX6QDL_PAD_SD2_CLK__SD2_CLK
             0x100f9
                                MX6QDL_PAD_SD2_DAT0__SD2_DATA0
```

```
                0x170f9
                                MX6QDL_PAD_SD2_DAT1__SD2_DATA1
                0x170f9
                                MX6QDL_PAD_SD2_DAT2__SD2_DATA2
                0x170f9
                                MX6QDL_PAD_SD2_DAT3__SD2_DATA3
                0x170f9
                        >;
                };

                pinctrl_usdhc3: usdhc3grp {
                        fsl,pins = <
                                MX6QDL_PAD_SD3_CMD__SD3_CMD
                   0x17059
                                MX6QDL_PAD_SD3_CLK__SD3_CLK
                   0x10059
                                MX6QDL_PAD_SD3_DAT0__SD3_DATA0
                0x17059
                                MX6QDL_PAD_SD3_DAT1__SD3_DATA1
                0x17059
                                MX6QDL_PAD_SD3_DAT2__SD3_DATA2
                0x17059
                                MX6QDL_PAD_SD3_DAT3__SD3_DATA3
                0x17059
                        >;
                };

                pinctrl_usdhc3_100mhz: usdhc3grp100mhz {
                        fsl,pins = <
                                MX6QDL_PAD_SD3_CMD__SD3_CMD
                   0x170b9
                                MX6QDL_PAD_SD3_CLK__SD3_CLK
                   0x100b9
                                MX6QDL_PAD_SD3_DAT0__SD3_DATA0
                0x170b9
                                MX6QDL_PAD_SD3_DAT1__SD3_DATA1
                0x170b9
                                MX6QDL_PAD_SD3_DAT2__SD3_DATA2
                0x170b9
                                MX6QDL_PAD_SD3_DAT3__SD3_DATA3
                0x170b9
                        >;
                };

                pinctrl_usdhc3_200mhz: usdhc3grp200mhz {
                        fsl,pins = <
                                MX6QDL_PAD_SD3_CMD__SD3_CMD
```

```
            0x170f9
                                MX6QDL_PAD_SD3_CLK__SD3_CLK
            0x100f9
                                MX6QDL_PAD_SD3_DAT0__SD3_DATA0
          0x170f9
                                MX6QDL_PAD_SD3_DAT1__SD3_DATA1
          0x170f9
                                MX6QDL_PAD_SD3_DAT2__SD3_DATA2
          0x170f9
                                MX6QDL_PAD_SD3_DAT3__SD3_DATA3
          0x170f9
                        >;
                };

                pinctrl_usdhc3_cdwp: usdhc3cdwp {
                        fsl,pins = <
                                MX6QDL_PAD_ENET_RXD0__GPIO1_IO27
0x80000000
                                MX6QDL_PAD_ENET_TXD1__GPIO1_IO29
0x80000000
                        >;
                };
        };
};

&usdhc2 {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_usdhc2>;
        cd-gpios = <&gpio1 4 0>;
        wp-gpios = <&gpio1 2 0>;
        status = "disabled";
};

&usdhc3 {
        pinctrl-names = "default", "state_100mhz", "state_200mhz";
        pinctrl-0 = <&pinctrl_usdhc3
                     &pinctrl_usdhc3_cdwp>;
        pinctrl-1 = <&pinctrl_usdhc3_100mhz
                        &pinctrl_usdhc3_cdwp>;
        pinctrl-2 = <&pinctrl_usdhc3_200mhz
                        &pinctrl_usdhc3_cdwp>;
        cd-gpios = <&gpio1 27 0>;
        wp-gpios = <&gpio1 29 0>;
        no-1-8-v;

        vmmc-supply = <&vdd_sd0_reg>;
        status = "disabled";
```

```
};
```

NAND Flash

PHYTEC modules integrating NAND memory, which is used as media for storing linux, dtb and its root filesystem, including applications and their data files. This type of media will be managed by the UBIFS filesystem. This filesystem uses compression and decompression on the fly, so there is a chance to bring more data into this device.

From Linux userspace the NAND flash partitions starting with */dev/mtdblock0*. Only the /dev/mtdblock4 on the PHYTEC modules 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.

Note: The partition's position and size is defined in the device tree of the kernel **and** the bootloader. If you want to change something, you have to modify both device trees, otherwise the definitions are out of sync. E.g.: An ubifs filesystem, which was flashed in the barebox, cannot be mounted in the kernel. Please keep this in mind.

Adding new partitions can be done with creating a new partition node in the dts. The *label* defines the name of the partition and the *reg* value defines offset and size of a partition. Keep in mind to update all following partitions when adding one or updating the size of a partition.

The partitions are defined in the DT, e.g. *imx6qdl-phytec-pfla02.dtsi*:

```
&gpmi {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_gpmi_nand>;
        nand-on-flash-bbt;
        status = "okay";

        #address-cells = <1>;
        #size-cells = <1>;

        partition@0 {
                label = "barebox";
                reg = <0x0 0x400000>;
        };

        partition@1 {
                label = "barebox-environment";
                reg = <0x400000 0x100000>;
        };

        partition@2 {
                label = "oftree";
                reg = <0x500000 0x100000>;
        };

        partition@3 {
                label = "kernel";
                reg = <0x600000 0x800000>;
        };

        partition@4 {
```

```
                label = "root";
                reg = <0xe00000 0x0>;
        };
};
```

GPIO

PHYTEC boards have often a set of different user I/Os. Those pins are muxed as GPIOs.

The linux kernel uses a single integer to enumerate all available GPIOs in the system. Therefore the tuple of numbers in i.MX6 GPIO name *GPIO<X>_IO<Y>* (example: *GPIO3_IO23*, X=3, Y=23) is convert using the following forumla

```
linux gpio number = (<X> - 1) * 32 + <Y>
```

Before using a GPIO from userspace you have to export it in */sys/class/gpio/export*. To this you write the *linux gpio number* to the file. Example:

```
echo 79 > /sys/class/gpio/export
```

Then you can change into the directory of the GPIO (e.g. */sys/class/gpio/gpio79/*) and controll the GPIO via the files *direction* and *value*. The following command sequence shows howto set the GPIO to low and high and to read the external value from it.

```
cd /sys/class/gpio/gpio79/
echo out > direction
echo 255 > value
echo 0 > value
echo in > direction
cat < value
```

Pinmuxing of some GPIO pins in the device tree *imx6qdl-phytec-pbab01.dtsi*:

```
                pinctrl_hog: hoggrp {
                        fsl,pins = <
                                MX6QDL_PAD_EIM_D23__GPIO3_IO23
0x80000000
                                MX6QDL_PAD_DISP0_DAT3__GPIO4_IO24
0x80000000 /* SPI NOR chipselect */
                                MX6QDL_PAD_SD4_DAT1__GPIO2_IO09
0x80000000 /* PMIC interrupt */
                                MX6QDL_PAD_ENET_TXD0__GPIO1_IO30
0x80000000 /* Green LED */
                                MX6QDL_PAD_EIM_EB3__GPIO2_IO31
0x80000000 /* Red LED */
                                MX6QDL_PAD_GPIO_8__GPIO1_IO08
0x80000000
[...]
```

GPI0 LEDS

I/O pins like LEDs can be accessed from userspace. They appear in */sys/class/leds/*. The maximum brightness of the LED is defined in max_brightness file. The brightness file will set the brightness of the LED (taking a value 0 to max_brightness). Most LEDs don't have hardware brightness support so will just be turned on for non-zero brightness settings.

Here is a simple example for the phyFLEX-i.MX6 Carrier Board:

```
root@phyflex-imx6-2:/sys/class/leds# ls
green:user4            phyflex:green          red:user1
mmc0::                 phyflex:red            yellow:user2
mmc1::                 phyflex:user_led_gpio  yellow:user3
root@phyflex-imx6-2:/sys/class/leds# echo 255 > phyflex\:user_led_gpio/brightness
root@phyflex-imx6-2:/sys/class/leds# echo 0 > phyflex\:user_led_gpio/brightness
```

User I/O configuration in device tree *imx6qdl-phytec-pbab01.dtsi*:

```
&gpio_leds {
        user_led_gpio {
                label = "phyflex:user_led_gpio";
                gpios = <&gpio2 24 GPIO_ACTIVE_HIGH>;
                linux,default-trigger = "gpio";
        };
};
```

SPI Master

Some PHYTEC boards have integrated NOR flashes which are connected through the i.MX6 ECSPI. The NOR flashes can be used for booting. See Booting from SPI NOR flash.

From Linux userspace the NOR flash partitions starting with *dev/mtdblock5* (see *mtdinfo -v*). There are currently four partitions: barebox, barebox-environment, oftree and kernel. Please note that there isn't a rootfs partition on the NOR.

The partitions are also defined in the DT, e.g. *imx6qdl-phytec-pfla02.dtsi*. If the partition layout needs to be changed, this has to be done in kernel the barebox device tree.

```
&ecspi3 {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_ecspi3>;
        status = "okay";
        fsl,spi-num-chipselects = <1>;
        cs-gpios = <&gpio4 24 0>;

        flash@0 {
                compatible = "m25p80";
                spi-max-frequency = <20000000>;
                reg = <0>;

                #address-cells = <1>;
                #size-cells = <1>;
```

```
                partition@0 {
                        label = "barebox";
                        reg = <0x0 0x100000>;
                };

                partition@1 {
                        label = "barebox-environment";
                        reg = <0x100000 0x20000>;
                };

                partition@2 {
                        label = "oftree";
                        reg = <0x120000 0x20000>;
                };

                partition@3 {
                        label = "kernel";
                        reg = <0x140000 0x0>;
                };
        };
};
```

I²C Bus

The i.MX6 offers I2C busses called I2C0, I2C1, and I2C2. PHYTEC boards implement plenty of different I2C devices to different i.MX6 I2C modules. This chapter will describe the basic device usage and its DT representation of a few integrated I2C devices on our RDK boards.

General I2C bus configuration (e.g. *imx6q-phytec-phycard-imx6-som.dtsi*):

```
&i2c1 {
        status = "okay";
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_i2c1>;
        /* ... */
};

&iomuxc {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_hog>;

        imx6q-phytec-pcaaxl3 {
                pinctrl_i2c1: i2c1grp {
                        fsl,pins = <
                                MX6QDL_PAD_EIM_D21__I2C1_SCL
            0x4001b8b1
                                MX6QDL_PAD_EIM_D28__I2C1_SDA
            0x4001b8b1
                        >;
```

```
                };
        };
};
```

EEPROM

It is possible to read and write directly to the device:

```
/sys/class/i2c-dev/i2c-0/device/0-0050/eeprom
```

To read and print the first 1024 bytes in the eeprom as hex, execute

```
 dd if=/sys/class/i2c-dev/i2c-0/device/0-0050/eeprom bs=1 count=1024  | od -x
```

For example to fill the whole eeprom with zeros execute

```
dd if=/dev/zero of=/sys/class/i2c-dev/i2c-0/device/0-0050/eeprom
```

This operation takes some time, because the eeprom is relatively slow.

Device tree representation of phyFLEX-i.MX6 file *imx6qdl-phytec-pfla02.dtsi*:

```
        eeprom@50 {
                compatible = "atmel,24c32";
                reg = <0x50>;
        };
```

RTC

RTCs can be accessed via /dev/rtc*. Because PHYTEC boards have often more than one RTC, there are also more RTC device files.

You can find out the name of the RTC device by reading its sysfs entry:

```
root@phyflex-imx6-2:~# cat /sys/class/rtc/rtc*/name
rtc-pcf8563
da9063-rtc
20cc034.snvs-rtc-lp
```

Note that this will list all RTCs and also the non I2C RTCs. Linux assigns RTC device IDs based on device tree /aliases entries if present.

imx6q-phytec-phycard-imx6-som.dtsi

```
        aliases {
                ipu0 = &ipu1;
                ipu1 = &ipu2;
                rtc1 = &pmic;
                rtc2 = &snvs_rtc;
        };
```

imx6q-phytec-phycard-imx6-rdk.dts:

```
        aliases {
                rtc0 = &i2c_rtc;
        };
```

So rtc0 should be always an I2C RTC.

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

DT representation for I2C RTCs: *imx6q-phytec-phycard-imx6-rdk.dts:*

```
        i2c_rtc: rtc@51 {
                compatible = "nxp,rtc8564";
                reg = <0x51>;
        };
```

Capacitive Touchscreen

The capacitive touchscreen is a part of the display module. A simple test of this feature can be run with

```
ts_calibrate
```

to calibrate the touch and with

```
ts_test
```

to start simple application using this feature.

imx6q-phytec-phycard-imx6-rdk.dts:

```
        polytouch: edt-ft5x06@38 {
                compatible = "edt,edt-ft5406", "edt,edt-ft5x06";
                reg = <0x38>;
                pinctrl-names = "default";
                pinctrl-0 = <&pinctrl_edt_ft5x06>;
                interrupt-parent = <&gpio1>;
                interrupts = <6 0>;
        };
```

LED dimmer

All LED dimmer controlled LEDs can be accessed through the sysfs. Here is a simple example:

List all LEDs:

```
root@phyflex-imx6-2:~# ls /sys/class/leds/
green:user4             phyflex:green           red:user1
mmc0::                  phyflex:red             yellow:user2
mmc1::                  phyflex:user_led_gpio   yellow:user3
```

LED red:user1 on:

```
root@phyflex-imx6-1:~# echo 255 > /sys/class/leds/red\:user1/brightness
```

LED red:user1 off:

```
root@phyflex-imx6-1:~# echo 0 > /sys/class/leds/red\:user1/brightness
```

Currently, only the GPIO functionality is used with the PCA9533. So you can only switch on/off the LED. We have not tested the dimmer switch functionality.

DT representation: *imx6qdl-phytec-pbab01.dtsi:*

```
        leddim: leddimmer@62 {
                compatible = "nxp,pca9533";
```

```
                gpio-controller;
                #gpio-cells = <2>;
                nxp,typecodes = <0xFF>;
                nxp,statecodes = <0x55>;
                reg = <0x62>;
        };
```

As you can see, the LED dimmer is a gpio-controller node. The PCA9533 controlled LEDs can be used within a gpio-leds driver.

Please read the related binding documentation for more information about gpio-controller and gpio-leds.

```
&user_leds_pca9533 {
        led1 {
                label = "red:user1";
                gpios = <&leddim 0 0>;
                linux,default-trigger = "none";
                default-state = "on";
        };

        led2 {
                label = "yellow:user2";
                gpios = <&leddim 1 0>;
                linux,default-trigger = "none";
                default-state = "on";
        };

        led3 {
                label = "yellow:user3";
                gpios = <&leddim 2 0>;
                linux,default-trigger = "none";
                default-state = "on";
        };

        led4 {
                label = "green:user4";
                gpios = <&leddim 3 0>;
                linux,default-trigger = "none";
                default-state = "on";
        };
};
```

USB Host Controller

The USB controller of the i.MX6 SoC provides a low-cost connectivity solution for numerous consumer portable devices by providing a mechanism for data transfer between USB devices with a line/bus speed up to 480 Mbps. The USB subsystem has four independent USB Controller cores, but only two supported the inter-chip UTMI interface and are connected to the two USB 2.0 PHY macrocells. The first USB controller is an OTG Controler (On-The-Go) capable of acting as an USB peripheral or USB host and the second Controller is host only.

The USB PHY's interface can be routed to normal USB 1.0/1.1/2.0 devices or plugs and connectors.

The unified BSP 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 USB 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.

Kernel device tree *imx6qdl-phytec-pfla02.dtsi*:

```
[...]
                pinctrl_usbh1: usbh1grp {
                        fsl,pins = <
                                MX6QDL_PAD_GPIO_0__GPIO1_IO00
           0x80000000
                                MX6QDL_PAD_GPIO_3__USB_H1_OC
            0x1b0b0
                        >;
                };

                pinctrl_usbotg: usbotggrp {
                        fsl,pins = <
                                MX6QDL_PAD_GPIO_1__USB_OTG_ID
           0x17059
                                MX6QDL_PAD_KEY_COL4__USB_OTG_OC
         0x1b0b0
                                MX6QDL_PAD_KEY_ROW4__GPIO4_IO15
         0x80000000
                        >;
                };
[...]
&usbh1 {
        vbus-supply = <&reg_usb_h1_vbus>;
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_usbh1>;
        status = "disabled";
};

&usbotg {
        vbus-supply = <&reg_usb_otg_vbus>;
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_usbotg>;
        disable-over-current;
        status = "disabled";
};
```

Kernel device tree *imx6qdl-phytec-pbab01.dtsi*:

```
 &usbh1 {
         status = "okay";
 };
```

```
&usbotg {
        status = "okay";
        dr_mode = "peripheral";
};
```

USB OTG

Some PHYTEC boards have support for USB OTG. USB OTG ports automatically act as USB device or USB host. The mode depends on the USB hardware which is attached to the USB OTG Port. If, for example, an USB mass storage device is attached to the USB OTG port, the device will show up as */dev/sda* on the device.

In order to connect the board as an USB device to an USB host port (for example a PC), you need to load an appropriate USB gadget, which is a kernel module, with *modprobe*. The BSP includes several USB gadgets: *g_ether*, *g_mass_storage* and *g_ether*.

1. To start the ethernet gadget execute

```
modprobe g_ether
```

   on the device. Then you have an additional ethernet interface like *usb0*.
2. To use the mass storage gadget execute

```
dd if=/dev/zero of=/tmp/file.img bs=1M count=64 
modprobe g_mass_storage file=/tmp/file.img
```

   on the device. The host can partition, format and mount the gadget mass storage the same way as any other USB mass storage.
3. The *g_serial* gadget is used the same way as *g_ether*, but it creates an additional serial interface like */dev/ttyGS0*.

SATA

Some boards like the phyFLEX-i.MX6 Kit features a SATA Connecter (Serial ATA) which is used to attach harddisks or optical drives. After connecting a harddisk it shows up as a standard block device (here */dev/sda*) like in any other linux systems.

```
root@phyflex-imx6-2:~# ls -l /dev/sda*
brw-rw----    1 root     disk        8,    0 May 27 07:56 /dev/sda
brw-rw----    1 root     disk        8,    1 May 27 07:56 /dev/sda1
```

The disk can be partitioned and formatted like any other block device.

To access the partitions you can also use the persistent block device naming in */dev/disk/*. See ArchLinux Wiki-Persistent block device naming [7].

PCIe

Some Phytec boards feature a PCIe- or Mini-PCIe-Slot. PCIe autodetects new devices on the bus. After connecting the device and booting up the system you can use the command *lspci* to see all recognized PCIe devices.

```
root@phyflex-imx6-2:/# lspci -v

00:00.0 PCI bridge: Device 16c3:abcd (rev 01) (prog-if 00 [Normal decode])

        Flags: bus master, fast devsel, latency 0

        Memory at 01000000 (32-bit, non-prefetchable) [size=1M]

        Bus: primary=00, secondary=01, subordinate=01, sec-latency=0

        I/O behind bridge: 00001000-00001fff
```

```
        Prefetchable memory behind bridge: 01100000-011fffff
        [virtual] Expansion ROM at 01200000 [disabled] [size=64K]
        Capabilities: [40] Power Management version 3
        Capabilities: [50] MSI: Enable- Count=1/1 Maskable+ 64bit+
        Capabilities: [70] Express Root Port (Slot-), MSI 00
        Capabilities: [100] Advanced Error Reporting
        Capabilities: [140] Virtual Channel
        Kernel driver in use: pcieport

01:00.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller (rev 06)
        Subsystem: Realtek Semiconductor Co., Ltd. Device 0123
        Flags: bus master, fast devsel, latency 0, IRQ 308
        I/O ports at 1000 [size=256]
        Memory at 01104000 (64-bit, prefetchable) [size=4K]
        Memory at 01100000 (64-bit, prefetchable) [size=16K]
        Capabilities: [40] Power Management version 3
        Capabilities: [50] MSI: Enable- Count=1/1 Maskable- 64bit+
        Capabilities: [70] Express Endpoint, MSI 01
        Capabilities: [b0] MSI-X: Enable- Count=4 Masked-
        Capabilities: [d0] Vital Product Data
        Capabilities: [100] Advanced Error Reporting
        Capabilities: [140] Virtual Channel
        Capabilities: [160] Device Serial Number ef-ba-00-00-68-4c-e0-00
        Kernel driver in use: r8169
        Kernel modules: r8169
```

Here the PCIe device is the *RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller*.

For PCIe devices you have to enable the correct driver in the kernel configuration. Here the ethernet card is manufactured by *Realtek Semiconductor Co.*. The corresponding Kconfig option for the driver, which you have to enable, is named *CONFIG_R8169* and it's describe as

```
Realtek 8169 gigabit ethernet support
```

For some devices like this ethernet card additional binary firmware blobs are needed. For example trying to bring up the network interface prints the following output onto the serial console (or see with *dmesg*):

```
root@phyflex-imx6-2:/# ip link set up enp1s0
[  620.116794] r8169 0000:01:00.0: Direct firmware load for rtl_nic/rtl8168e-3.fw failed with error -2
[  620.125943] r8169 0000:01:00.0 enp1s0: unable to load firmware patch rtl_nic/rtl8168e-3.fw (-2)
[  620.163733] r8169 0000:01:00.0 enp1s0: link down
[  620.168703] r8169 0000:01:00.0 enp1s0: link down
[  620.178436] IPv6: ADDRCONF(NETDEV_UP): enp1s0: link is not ready
```

These firmware blobs have to be placed in */lib/firmware/* before the device can be used.

Note: Interestingly the ethernet PCIe card works without the firmware, too. It has maybe an internal default firmware. That mustn't be the case for all PCIe devices.

Sound

On Phytec i.MX6 boards different audio chips are used. This section is written for the phyFLEX-i.MX6 baseboard, but it should be valid for other boards and audio chips, too.

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

Audio Sources and Sinks

The baseboard has four jacks for Microphone, Headset Speaker, Line In and Line Out.

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.

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 they will scroll if your cursor reaches the right or left edge of it. In case you get trouble in the display during scrolling, please use ssh instead of microcom.

Screenshot of Alsamixer

alsamixer can be left by pressing the ESC key. The settings are saved automatically on shutdown and restored on boot by the systemd service *alsa-restore*. If you want to save the mixer settings now, you can execute *aslactl store* by hand. The settings are saved in */var/lib/alsa/asound.state*.

Playback

To playback simple audio streams, you can use *aplay*. An example:

```
aplay /usr/share/sounds/alsa/Front_Center.wav
```

Other file formats like .ogg, .mp3 or .flac are currently not supported by default in the BSP image, but they can be playbacked by installing further software on the image.

Capture

arecord is a command line tool for capturing audio streams. Default input source is Line In. You can use alsamixer for selecting a different audio to capture. In particular switch on Right PGA Mixer Mic3R and Left PGA Mixer Mic3L for capturing from Mic. Please note that it's a known error that you need to choose Playback screen (F3) instead of Capture screen (F4) for accessing these two controls. The following example will capture the current stereo input source with sample rate and will create an audio file in WAV format (signed 16 bit per channel, 32 bit per sample):

```
arecord -t wav -c 2 -r 48000 -f S16_LE test.wav
```

Capturing can be stopped again using Strg-C.

Framebuffer

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

The BSP is already prepared for use with the ETM0700G0DH6 (800x480) display, e.g. for the phyFLEX-i.MX6 Kit. Other displays will be added in later BSP versions.

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. Some documentation is in the kernel Documentation/fb/modedb.txt [8]. Refer to the manual of your display driver for more details, too.

To modify the kernel arguments temporally in the barebox execute

```
global.linux.bootargs.base="$global.linux.bootargs.base video=HDMI-A-1:1024x768-24"
```

This sets the HDMI interface to 1024x768 pixels and bits per pixel to 24bpp. To get the name of the video interface use on the target

```
$ ls /sys/class/drm/ -1
card0
card0-HDMI-A-1
card0-LVDS-1
controlD64
version
```

The suffixes of "card0-*" are the identifiers.

For phyFLEX-i.MX6 the display DT representation can be found in *imx6qdl-phytec-lcd-018.dtsi*.

Backlight Control

If the Phytec board has attached display, you can control the backlight of it with the Linux kernel sysfs interface. See the folder */sys/class/backlight* for all available backlight devices in the system.

Mostly there should be only one folder for the single display named *backlight*. You can control the backlight by reading and writing to the appropriate files.

```
target$ cd /sys/class/backlight/backlight
target$ cat max_brightness   # Query max brightness level. Valid brightness values are 0 to <max_brightness>
7
target$ cat brightness       # Query the current brightness level in the driver
2
target$ echo 0 > brightness  # Set to zero brightness. Backlight is off
target$ echo 6 > brightness  # Set to second highest brightness level.
```

For the documentation of all files see Kernel Documentation/ABI/stable/sysfs-class-backlight [9].

Note: phyFLEX-i.MX6 Carrier Board (Kit)
If dimming by writing to the file *brightness* doesn't work, check the DIP switch settings on the bottom side of the display board. The correct setting of *DIP Switch S1* is 1=OFF, 2=OFF, **3=ON** and 4=OFF.

CPU core frequency scaling

The CPU in the i.MX6 SoC is able to scale the clock frequency and the voltage. This is used to save power when the full performance of the cpu is not needed. Scaling the frequency and the voltage is referred as 'Dynamic Voltage and Frequency Scaling' (DVFS).

The Linux Kernel provides a DVFS framework that allows each CPU core to have a min/max frequency and a governor that governs it.

Several different frequencies are supported. This depends on the i.MX6 variant, which is worked with. You may type:

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

and you'll get them all listed. In case you have an i.MX6 CPU with a maximum of nearly 1 GHz these are:

```
396000 792000 996000
```

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

You can decrease the maximum frequency (e.g. to 792000)

```
echo 792000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
```

or increase the minium frequency (e.g. to 792000)

```
echo 792000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq
```

Asking for the current frequency will be done with:

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

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

- *performance* Always selects the highest possible CPU core frequency.
- *powersave* Always selects the lowest possible CPU core frequency.
- *ondemand* Switches between possible CPU core frequencies in reference to the current system load. When the system load increases above a specific limit it increases the CPU core frequency immediately. This is the default governor when the system starts up.
- *userspace* Allows the user or userspace program running as root to set a specific frequency (e.g. to 792000):

```
echo 792000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
```

Show available governors

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

In order to ask for the current governor, type

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

and you'll normally get:

```
ondemand
```

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

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

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

```
linux/Documentation/cpu-freq/governors.txt.
```

Thermal Management

This sections is only valid for version BSP-Yocto-i.MX6-PD15.1.0.

The linux kernel has an integrated thermal management [10]. It is capable of monitoring SoC temperatures, reducing the CPU frequency, driving fans, advising other drivers to reduce the power consumption of devices and – in the worst case – shutting down the system gracefully.
This sections describes how the thermal management kernel api is used for the i.MX6 SoC platform:

There are two trip points registered by the *imx_thermal* kernel driver:

1. *trip_point_0*: 85°C type: *passive*
2. *trip_point_1*: 105°C type: *critical*

(See the kernel SysFS folder */sys/devices/virtual/thermal/thermal_zone*.)

These trip points are used by the kernel thermal management to trigger events and change the cooling behaviour.

There are different policy available in the kernel: *Step Wise*, *Fair Share*, *Bang Bang* and *User space*. Thermal policies are also named *thermal governors*.

The default policy, that is used in the BSP, is *step_wise*. When the SoC temperature in *temp* is above the *trip_point_0* (greater than 85°C) and the temperature of the SoC is rising, the cpu frequency is throttled. When SoC temperature is falling again, the throttling is released.

When the SoC temperature reaches 105°C, the thermal management of the kernel shuts down the systems and poweroffs. On ther serial console you can see:

```
kernel[194]: [  895.524255] thermal thermal_zone0: critical temperature reached(105 C),shutting down


[  OK  ] Stopped target Sound Card.
[  OK  ] Stopped target System Time Synchronized.
[  OK  ] Stopped target Network.
         Stopping Autostart Qt 5 Demo...
[...]
```

On-Chip OTP Controller (OCOTP) - Fuses

The i.MX6 contains one-time-programmable fuses to store information like the MAC Adresse, Boot configuration and other permanent settings. See the i.MX6 References Manual *Chapter 47 On-Chip OTP Controller (OCOTP_CTRL)* for general details about OCOTP_CTRL chip.

Short list of useful eFUSE registiers in the OCOTP_CTRL (base address 0x21BC000, from the i.MX6 Solo/DualLite Reference Manual).

- *OCOTP_CFG4*: addr 0x05, memory offset 0x450 (contains BOOT_CFG[1,2,3,4]. Transfered to register SRC_SBMR1 *0x20D8004*)
- *OCOTP_CFG5*: addr 0x06, memory offset 0x460 (contains BT_FUSE_SEL at the fifths bit. Some bits are transfered to register SRC_SBMR2 *0x20D801C*)
- *OCOTP_MAC0*: addr 0x22, memory offset 0x620 (Bits 0-31 of MAC_ADDR)
- *OCOTP_MAC1*: addr 0x23, memory offset 0x630 (Bits 32-47 of MAC_ADDR)

The list is not exhaustive. See the tables in *Chapter 5 Fusemap* of the i.MX6 reference manual for a detailed mapping between the fuses in OCOTP and the boot/mac/... configuration.

Read fuses values in barebox

You can read the fuse values using memory mapped shadow registers. To calculate the memory address take the *fuse address* (0x01-0x2f) and put it into the formula

```
memory address = 0x21BC400 + <fuse address> * 0x10
```

In the barebox use *md* to read the value. Examples:

```
md 0x21BC450+4    # OCOTP_CFG4: addr 0x05
md 0x21BC460+4    # OCOTP_CFG5: addr 0x06
md 0x21BC620+4    # OCOTP_MAC0: addr 0x22
md 0x21BC630+4    # OCOTP_MAC1: addr 0x23
```

Burn boot configuration fuses

CAUTION: Fuses can only be written once! You can brick your board easily, e.g. by burning a wrong boot configuration. I cannot be undone.

Execute the following commands on the barebox prompt to burn the BOOT_CFG? values and enable boot from fuses. Replace the *BOOT_CFG1-4* the appropriate value for your boot device, eg. eMMC, SD, SPI, NAND, ...

```
# enable write to fuses

ocotp0.permanent_write_enable=1

ocotp0.sense_enable=1

# check values

devinfo ocotp0

# burn fuses

mw -l -d /dev/imx-ocotp 0x14 0x58003283     # BOOT_CFG1-4 (Here: NAND 2Gb 64 pages per block) Replace with actual boot config of your board.

mw -l -d /dev/imx-ocotp 0x18 0x00000010     # Set bit BT_FUSE_SEL
```

Burn MAC Address

To burn the mac address you don't have to blow the two fuse registers by hand. The barebox ocotp drivers has a special command for that. Replace the dummy MAC address in the following commands with a valid MAC Addresse from your pool of addresses.

```
# enable write to fuses
ocotp0.permanent_write_enable=1
ocotp0.sense_enable=1
# check values
devinfo ocotp0
# burn MAC address
```

ocotp0.mac_addr=11:22:33:44:55:66

References

[1] ftp://ftp.phytec.de/pub/Software/Linux/Yocto/

[2] http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=i.MX6DL&tab=Documentation_Tab&link_28|Freescale

[3] http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=i.MX6Q&tab=Documentation_Tab&pspll_35|Freescale

[4] https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/arch/arm/boot/dts/imx6q-pinfunc.h

[5] https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/arch/arm/boot/dts/imx6dl-pinfunc.h

[6] https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/Documentation/devicetree/bindings/pinctrl/fsl,imx6q-pinctrl.txt

[7] https://wiki.archlinux.org/index.php/Persistent_block_device_naming|

[8] https://www.kernel.org/doc/Documentation/fb/modedb.txt

[9] https://www.kernel.org/doc/Documentation/ABI/stable/sysfs-class-backlight

[10] https://www.kernel.org/doc/Documentation/thermal/sysfs-api.txt

Document: Yocto/i.MX 6 BSP Manual
Document number: L-814e_1, September 2015

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
Postfach 100403
D-55135 Mainz, Germany
Fax : +49 (6131) 9221-33

Published by

PHYTEC

Ordering No. L-814e_1
Printed in Germany