i.MX 8M Mini BSP Manual PD23.1.0

PHYTEC Messtechnik GmbH

Dec 02, 2024

CONTENTS

i.MX 8M Mini BSP Manual	
Document Title	i.MX 8M Mini BSP Manual
Document Type	BSP Manual
Yocto Manual	Kirkstone
Release Date	2024/01/10
Is Branch of	i.MX 8M Mini BSP Manual

The table below shows the Compatible BSPs for this manual:

Compatible BSPs	BSP Release Type	BSP Release Date	BSP Status
BSP-Yocto-NXP-i.MX8MM-PD23.1.0	Major	2023/12/12	Released

This BSP manual guides you through the installation and creation steps for the Board Support Package (BSP) and describes how to handle the interfaces for the **phyCORE-i.MX8M Mini Kit**. Furthermore, this document describes how to create BSP images from the source code. This is useful for those who need to change the default image and need a way to implement these changes in a simple and reproducible way. Further, some sections of this manual require executing commands on a personal computer (host). Any and all of these commands are assumed to be executed on a Linux Operating System.

Note

This document contains code examples that describe the communication with the board over the serial shell. The code examples lines begin with “host:~$”, “target:~$” or “u-boot=>”. This describes where the commands are to be executed. Only after these keywords must the actual command be copied.

PHYTEC provides a variety of hardware and software documentation for all of our products. This includes any or all of the following:

- **QS Guide**: A short guide on how to set up and boot a phyCORE board along with brief informationon building a BSP, the device tree, and accessing peripherals.
- **Hardware Manual**: A detailed description of the System on Module and accompanying carrierboard.
- **Yocto Guide**: A comprehensive guide for the Yocto version the phyCORE uses. This guide contains an overview of Yocto; introducing, installing, and customizing the PHYTEC BSP; how to work with programs like Poky and Bitbake; and much more.
- **BSP Manual**: A manual specific to the BSP version of the phyCORE. Information such as how to build the BSP, booting, updating software, device tree, and accessing peripherals can be found here.
- **Development Environment Guide**: This guide shows how to work with the Virtual Machine (VM) Host PHYTEC has developed and prepared to run various Development Environments. There are detailed step-by-step instructions for Eclipse and Qt Creator, which are included in the VM. There are instructions for running demo projects for these programs on a phyCORE product as well. Information on how to build a Linux host PC yourself is also a part of this guide.
- **Pin Muxing Table**: phyCORE SOMs have an accompanying pin table (in Excel format). This table will show the complete default signal path, from the processor to the carrier board. The default device tree muxing option will also be included. This gives a developer all the information needed in one location to make muxing changes and design options when developing a specialized carrier board or adapting a PHYTEC phyCORE SOM to an application.

On top of these standard manuals and guides, PHYTEC will also provide Product Change Notifications, Application Notes, and Technical Notes. These will be done on a case-by-case basis. Most of the documentation can be found on the https://www.phytec.de/produkte/system-on-modules/phycore-imx-8m-mini/nano/#downloads of our product.

CHAPTER

ONE

SUPPORTED HARDWARE

The phyBOARD-Polis populated with either the i.MX 8M Mini SoC or i.MX 8M Nano SoC, is supported.

On our web page, you can see all supported Machines with the available Article Numbers for this release: BSP-Yocto-NXP-i.MX8MM-PD23.1.0 download. If you choose a specific **Machine Name** in the section **Supported Machines**, you can see which **Article Numbers** are available under this machine and also a short description of the hardware information. In case you only have the **Article Number** of your hardware, you can leave the **Machine Name** drop-down menu empty and only choose your **Article Number**. Now it should show you the necessary **Machine Name** for your specific hardware

1.1 phyBOARD-Polis Components

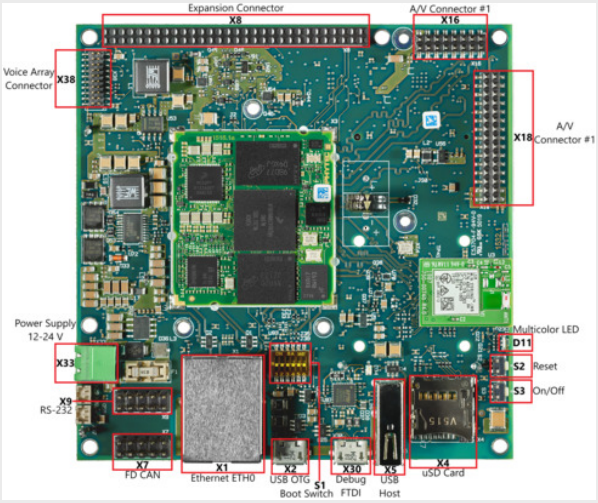


Fig. 1: phyBOARD-Polis Components (top)

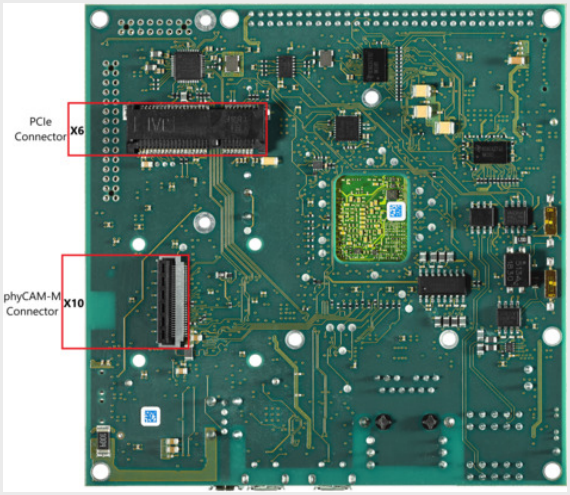


Fig. 2: phyBOARD-Polis Components (bottom)

CHAPTER

TWO

GETTING STARTED

The **phyCORE-i.MX8M Mini Kit** is shipped with a pre-flashed SD card. It contains the phytec-qt6demo-image and can be used directly as a boot source. The eMMC is programmed with only a U-Boot by default. You can get all sources from the PHYTEC download server. This chapter explains how to flash a BSP image to SD card and how to start the board.

There are several ways to flash an image to SD card or even eMMC. Most notably using simple, sequential writing with the Linux command line tool `dd`. An alternative way is to use PHYTEC's system initialization program called partup, which makes it especially easy to format more complex systems. You can get prebuilt Linux binaries of partup from its release page. Also read partup's README for installation instructions.

2.1 Get the Image

The image contains all necessary files and makes sure partitions and any raw data are correctly written. Both the partup package and the WIC image, which can be flashed using `dd`, can be downloaded from the PHYTEC download server.

Get either the partup package or the WIC image from the download server:

```
host:~$ wget https://download.phytec.de/Software/Linux/BSP-Yocto-i.MX8MM/BSP-Yocto-NXP-i.MX8MM-
↪PD23.1.0/images/ampliphy-vendor-xwayland/phyboard-polis-imx8mm-5/phytec-qt6demo-image-phyboard-
↪polis-imx8mm-5.partup
host:~$ wget https://download.phytec.de/Software/Linux/BSP-Yocto-i.MX8MM/BSP-Yocto-NXP-i.MX8MM-
↪PD23.1.0/images/ampliphy-vendor-xwayland/phyboard-polis-imx8mm-5/phytec-qt6demo-image-phyboard-
↪polis-imx8mm-5.wic
```

Note

For eMMC, more complex partitioning schemes or even just large images, we recommend using the partup package, as it is faster in writing than `dd` and allows for a more flexible configuration of the target flash device.

2.2 Write the Image to SD Card

Warning

To create your bootable SD card, you must have root privileges on your Linux host PC. Be very careful when specifying the destination device! All files on the selected device will be erased immediately without any further query!

Selecting the wrong device may result in **data loss** and e.g. could erase your currently running system on your host PC!

2.2.1 Finding the Correct Device

To create your bootable SD card, you must first find the correct device name of your SD card and possible partitions. If any partitions of the SD cards are mounted, unmount those before you start copying the image to the SD card.

1. In order to get the correct device name, remove your SD card and execute:

   ```
   host:~$ lsblk
   ```

2. Now insert your SD card and execute the command again:

   ```
   host:~$ lsblk
   ```

3. Compare the two outputs to find the new device names listed in the second output. These are the device names of the SD card (device and partitions if the SD card was formatted).

4. In order to verify the device names being found, execute the command `sudo dmesg`. Within the last lines of its output, you should also find the device names, e.g. `/dev/sde` or `/dev/mmcblk0` (depending on your system).

Alternatively, you may use a graphical program of your choice, like GNOME Disks or KDE Partition Manager, to find the correct device.

Now that you have the correct device name, e.g. `/dev/sde`, you can see the partitions which must be unmounted if the SD card is formatted. In this case, you will also find the device name with an appended number (e.g. `/dev/sde1`) in the output. These represent the partitions. Some Linux distributions automatically mount partitions when the device gets plugged in. Before writing, however, these need to be unmounted to avoid data corruption.

Unmount all those partitions, e.g.:

```
host:~$ sudo umount /dev/sde1
host:~$ sudo umount /dev/sde2
```

Now, the SD card is ready to be flashed with an image, using either `partup`, `dd` or `bmap-tools`.

2.2.2 Using partup

Writing to an SD card with partup is done in a single command:

```
host:~$ sudo partup install phytec-qt6demo-image-phyboard-polis-imx8mm-5.partup /dev/<your_
↪device>
```

Make sure to replace <your_device> with your actual device name found previously.

Further usage of partup is explained at its official documentation website.

Note

partup has the advantage of allowing to clear specific raw areas in the MMC user area, which is used in our provided partup packages to erase any existing U-Boot environments. This is a known issue *bmaptool* does not solve, as mentioned below.

Another key advantage of partup over other flashing tools is that it allows configuring MMC specific parts, like writing to eMMC boot partitions, without the need to call multiple other commands when writing.

2.2.3 Using bmap-tools

An alternative and also fast way to prepare an SD card is using bmap-tools. Yocto automatically creates a block map file (`<IMAGENAME>-<MACHINE>.wic.bmap`) for the WIC image that describes the image content and includes checksums for data integrity. *bmaptool* is packaged by various Linux distributions. For Debian-based systems install it by issuing:

```
host:~$ sudo apt install bmap-tools
```

Flash a WIC image to SD card by calling:

```
host:~$ bmaptool copy phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic /dev/<your_device>
```

Replace <your_device> with your actual SD card's device name found previously, and make sure to place the file `<IMAGENAME>-<MACHINE>.wic.bmap` alongside the regular WIC image file, so bmaptool knows which blocks to write and which to skip.

Warning

bmaptool only overwrites the areas of an SD card where image data is located. This means that a previously written U-Boot environment may still be available after writing the image.

2.2.4 Using `dd`

After having unmounted all SD card's partitions, you can create your bootable SD card.

Some PHYTEC BSPs produce uncompressed images (with filename-extension *.wic), and some others produce compressed images (with filename-extension *.wic.xz).

To flash an uncompressed images (*.wic) use command below:

```
host:~$ sudo dd if=phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic of=/dev/<your_device> bs=1M␣
↪conv=fsync status=progress
```

Or to flash a compressed images (*.wic.xz) use that command:

```
host:~$ xzcat phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic.xz | sudo dd of=/dev/<your_device>
↪ bs=1M conv=fsync status=progress
```

Again, make sure to replace <your_device> with your actual device name found previously.

The parameter `conv=fsync` forces a sync operation on the device before `dd` returns. This ensures that all blocks are written to the SD card and none are left in memory. The parameter `status=progress` will print out information on how much data is and still has to be copied until it is finished.

2.3 First Start-up

- To boot from an SD card, *bootmode switch (S1)* needs to be set to the following position:

Table 1: Bootmode Selection

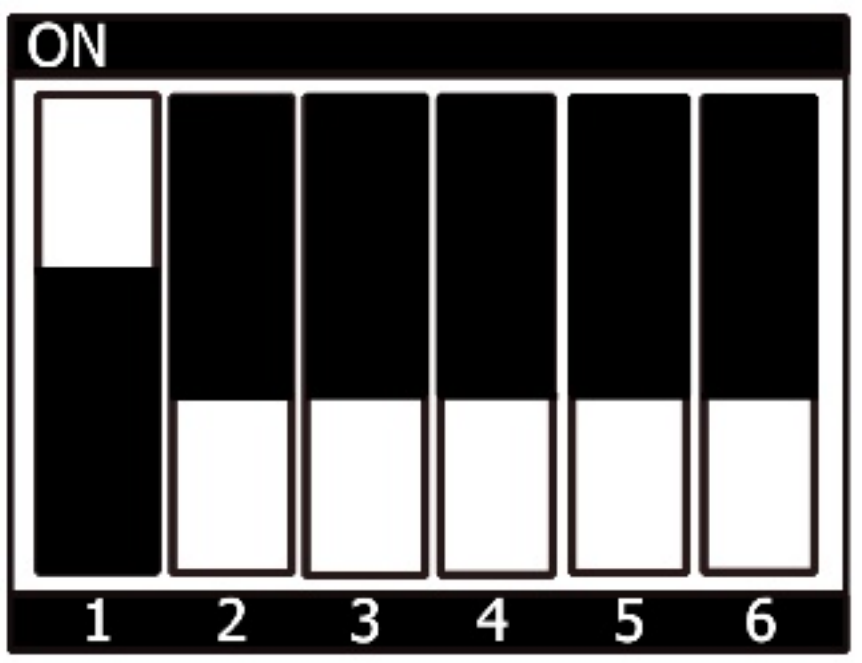


Fig. 1: Mini

- Insert the SD card
- Connect the target and the host with **mirco USB** on *(X30)* debug USB
- Power up the board

CHAPTER

THREE

BUILDING THE BSP

This section will guide you through the general build process of the i.MX 8M Mini BSP using Yocto and the phyLinux script. For more information about our meta-layer or Yocto in general visit: Yocto Reference Manual (kirkstone).

3.1 Basic Set-Up

If you have never created a Phytec BSP with Yocto on your computer, you should take a closer look at the chapter BSP Workspace Installation in the Yocto Reference Manual (kirkstone).

3.2 Get the BSP

There are two ways to get the BSP sources. You can download the complete BSP sources from our download page: BSP-Yocto-IMX8MM; or you can fetch and build it yourself with Yocto. This is particularly useful if you want to make customizations.

The phyLinux script is a basic management tool for PHYTEC Yocto BSP releases written in Python. It is mainly a helper to get started with the BSP structure.

- Create a fresh project folder, get phyLinux, and make the script executable:

```
host:~$ mkdir ~/yocto
host:~$ cd yocto/
host:~/yocto$ wget https://download.phytec.de/Software/Linux/Yocto/Tools/phyLinux
host:~/yocto$ chmod +x phyLinux
```

> **Warning**
>
> A clean folder is important because phyLinux will clean its working directory. Calling phyLinux from a directory that isn't empty will result in a warning.

- Run phyLinux:

```
host:~/yocto$ ./phyLinux init
```

> **Note**
>
> On the first initialization, the phyLinux script will ask you to install the Repo tool in your `/usr/local/bin` directory.

- During the execution of the init command, you need to choose your processor platform (SoC), PHYTEC's BSP release number, and the hardware you are working on.

> **Note**
>
> If you cannot identify your board with the information given in the selector, have a look at the invoice for the product. And have a look at our BSP.

- It is also possible to pass this information directly using command line parameters:

```
host:~/yocto$ DISTRO=ampliphy-vendor-xwayland MACHINE=phyboard-polis-imx8mm-5 ./phyLinux␣
↪init -p imx8mm -r BSP-Yocto-NXP-i.MX8MM-PD23.1.0
```

After the execution of the init command, phyLinux will print a few important notes. For example, it will print your git identify, SOC and BSP release which was selected as well as information for the next steps in the build process.

3.2.1 Starting the Build Process

- Set up the shell environment variables:

```
host:~/yocto$ source sources/poky/oe-init-build-env
```

> **Note**
>
> This needs to be done every time you open a new shell for starting builds.

- The current working directory of the shell should change to build/.
- Open the main configuration file and accept the GPU and VPU binary license agreements. Do this by uncommenting the corresponding line, as below.

```
host:~/yocto/build$ vim conf/local.conf
# Uncomment to accept NXP EULA
# EULA can be found under ../sources/meta-freescale/EULA
ACCEPT_FSL_EULA = "1"
```

- Build your image:

```
host:~/yocto/build$ bitbake phytec-qt6demo-image
```

> **Note**
>
> For the first build we suggest starting with our smaller non-graphical image phytec-headless-image to see if everything is working correctly.
>
> ```
> host:~/yocto/build$ bitbake phytec-headless-image
> ```
>
> The first compile process takes about 40 minutes on a modern Intel Core i7. All subsequent builds will use the filled caches and should take about 3 minutes.

3.2.2 BSP Images

All images generated by Bitbake are deployed to `~/yocto/build/deploy*/images/<machine>`. The following list shows for example all files generated for the phyboard-polis-imx8mm-5 machine:

- **u-boot.bin**: Binary compiled U-boot bootloader (U-Boot). Not the final Bootloader image!
- **oftree**: Default kernel device tree
- **u-boot-spl.bin**: Secondary program loader (SPL)
- **bl31-imx8mm.bin**: ARM Trusted Firmware binary
- **lpddr4_pmu_train_2d_dmem_202006.bin, lpddr4_pmu_train_2d_imem_202006.bin**: DDR PHY firmware images
- **imx-boot**: Bootloader build by imx-mkimage which includes SPL, U-Boot, ARM Trusted Firmware and DDR firmware. This is the final bootloader image which is bootable.
- **Image**: Linux kernel image
- **Image.config**: Kernel configuration
- **imx8mm-phyboard-polis-rdk*.dtb**: Kernel device tree file
- **imx8mm-phy*.dtbo**: Kernel device tree overlay files
- **phytec-qt6demo-image*.tar.gz**: Root file system
- **phytec-qt6demo-image*.wic**: SD card image

CHAPTER

FOUR

INSTALLING THE OS

4.1 Bootmode Switch (S1)

The phyBOARD-Polis features a boot switch with six individually switchable ports to select the phyCORE-i.MX 8M Mini default bootsource.

4.1.1 Mini

Table 1: Bootmode Selection

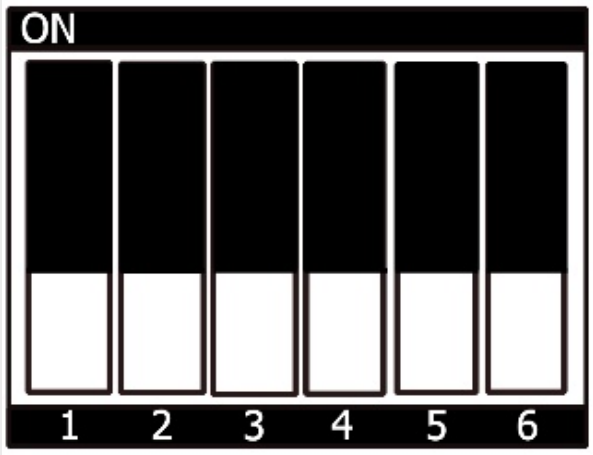


Fig. 1: eMMC (Default SoM boot)

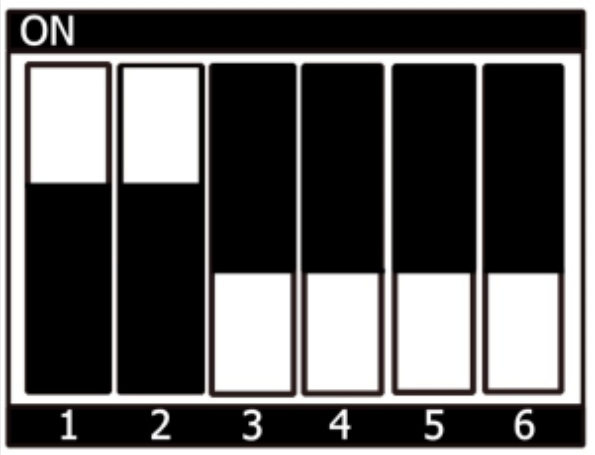


Fig. 2: SPI NOR

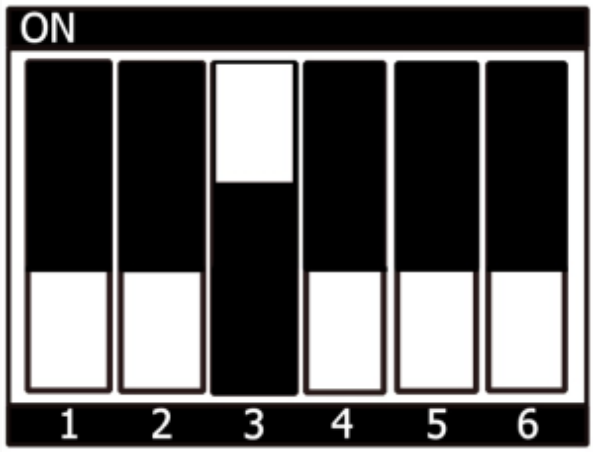


Fig. 3: USB Serial Download

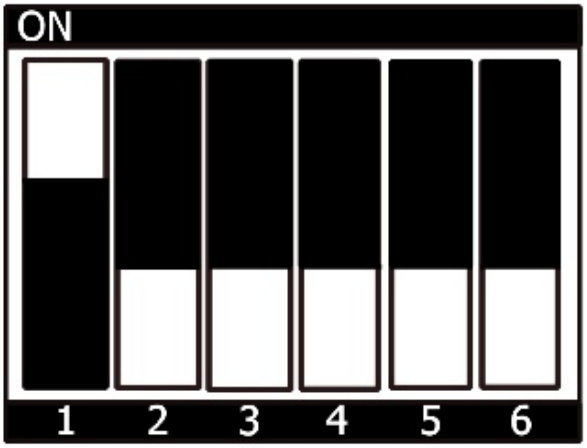


Fig. 4: SD Card

Table 2: Switch between UART1 RS485/RS232

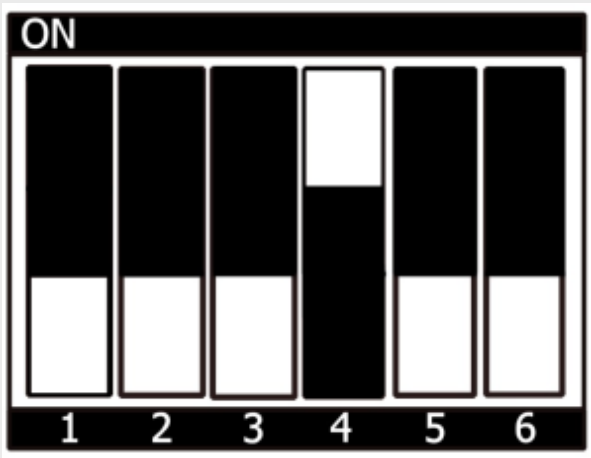


Fig. 5: UART1 RS485

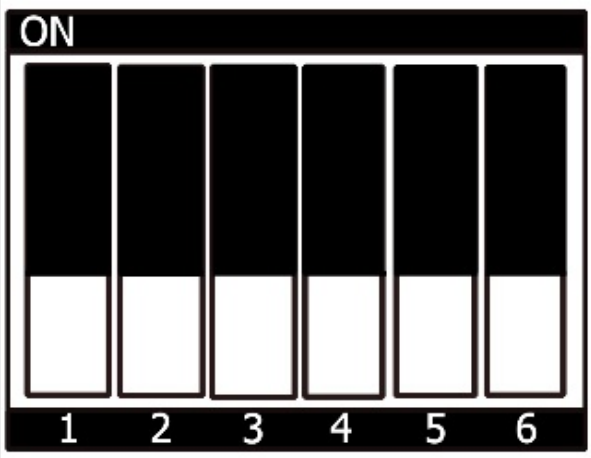


Fig. 6: UART1 RS232

4.2 Flash eMMC

For consistency, it is assumed that a TFTP server is configured; More importantly, all generated images, as listed above, are copied to the default /srv/tftp directory. If you do not have this set up, you need to adjust the paths that point to the images being used in the instructions. For instructions on how to set up the TFTP server and directory, see *Setup Network Host*.

To boot from eMMC, make sure that the BSP image is flashed correctly to the eMMC and the *bootmode switch (S1)* is set to **eMMC**.

Warning

When eMMC and SD card are flashed with the same (identical) image, the UUIDs of the boot partitions are also identical. If the SD card is connected when booting, this leads to non-deterministic behavior as Linux mounts the boot partition based on UUID.

```
target:~$ blkid
```

can be run to inspect whether the current setup is affected. If `mmcblk2p1` and `mmcblk1p1` have an identical UUID, the setup is affected.

4.2.1 Flash eMMC from Network

i.MX 8M Mini boards have an Ethernet connector and can be updated over a network. Be sure to set up the development host correctly. The IP needs to be set to 192.168.3.10, the netmask to 255.255.255.0, and a TFTP server needs to be available. From a high-level point of view, an eMMC device is like an SD card. Therefore, it is possible to flash the **WIC image** (`<name>.wic`) from the Yocto build system directly to the eMMC. The image contains the bootloader, kernel, device tree, device tree overlays, and root file system.

Flash eMMC from Network in U-Boot on Target

These steps will show how to update the eMMC via a network.

Tip

This step only works if the size of the image file is less than 1GB due to limited usage of RAM size in the Bootloader after enabling OPTEE.

Tip

A working network is necessary! *Setup Network Host*

Load your image via network to RAM:

- when using dhcp

```
u-boot=> dhcp phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic
BOOTP broadcast 1
DHCP client bound to address 192.168.3.1 (1 ms)
Using ethernet@30be0000 device
TFTP from server 192.168.3.10; our IP address is 192.168.3.1
Filename 'phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic'.
Load address: 0x40480000
Loading: #################################
         #################################
         #################################
         ...
         ...
         ...
         #################################
         #############
         11.2 MiB/s
done
Bytes transferred = 911842304 (36599c00 hex)
```

- when using a static ip address (serverip and ipaddr must be set additionally).

```
u-boot=> tftp ${loadaddr} phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic
Using ethernet@30be0000 device
TFTP from server 192.168.3.10; our IP address is 192.168.3.11
Filename 'phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic'.
Load address: 0x40480000
Loading: #################################
         #################################
         #################################
         ...
         ...
         ...
         #################################
         #############
         11.2 MiB/s
```

(continues on next page)

(continued from previous page)

```
done
Bytes transferred = 911842304 (36599c00 hex)
```

Write the image to the eMMC:

```
u-boot=> mmc dev 2
switch to partitions #0, OK
mmc2(part 0) is current device
u-boot=> setexpr nblk ${filesize} / 0x200
u-boot=> mmc write ${loadaddr} 0x0 ${nblk}

MMC write: dev # 2, block # 0, count 1780942 ... 1780942 blocks written: OK
```

Flash eMMC via Network in Linux on Target

You can update the eMMC from your target.

Tip

A working network is necessary! *Setup Network Host*

Take a compressed or uncompressed image with accompanying block map on the host and send it with ssh through the network to the eMMC of the target with a one-line command:

```
target:~$ scp <USER>@192.168.3.10:/srv/tftp/phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic.* /
↪tmp && bmaptool copy /tmp/phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic /dev/mmcblk2
```

Flash eMMC via Network in Linux on Host

It is also possible to install the OS at eMMC from your Linux host. As before, you need a complete image on your host.

Tip

A working network is necessary! *Setup Network Host*

Show your available image files on the host:

```
host:~$ ls /srv/tftp
phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic
phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic.bmap
```

Send the image with the `bmaptool` command combined with ssh through the network to the eMMC of your device:

```
host:~$ scp /srv/tftp/phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic root@192.168.3.11:/tmp &&␣
↪ssh root@192.168.3.11 "bmaptool copy /tmp/phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic /
↪dev/mmcblk2"
```

4.2.2 Flash eMMC U-Boot image via Network from running U-Boot

Update the standalone U-Boot image imx-boot is also possible from U-Boot. This can be used if the bootloader on eMMC is located in the eMMC user area.

Tip

A working network is necessary! *Setup Network Host*

Load image over tftp into RAM and then write it to eMMC:

```
u-boot=> tftp ${loadaddr} imx-boot
u-boot=> setexpr nblk ${filesize} / 0x200
u-boot=> mmc dev 2
u-boot=> mmc write ${loadaddr} 0x42 ${nblk}
```

Hint

The hexadecimal value represents the offset as a multiple of 512 byte blocks. See the *offset table* for the correct value of the corresponding SoC.

4.2.3 Flash eMMC from USB stick

Flash eMMC from USB stick in U-Boot on Target

Tip

This step only works if the size of the image file is less than 1GB due to limited usage of RAM size in the Bootloader after enabling OPTEE.

These steps will show how to update the eMMC via a USB device. Configure the *bootmode switch (S1)* to SD Card and insert an SD card. Power on the board and stop in U-Boot prompt. Insert a USB device with the copied WIC image to the USB slot.

Load your image from the USB device to RAM:

```
u-boot=> usb start
starting USB...
USB0:   USB EHCI 1.00
scanning bus 0 for devices... 2 USB Device(s) found
       scanning usb for storage devices... 1 Storage Device(s) found
u-boot=> fatload usb 0:1 ${loadaddr} *.wic
497444864 bytes read in 31577 ms (15 MiB/s)
```

Write the image to the eMMC:

```
u-boot=> mmc dev 2
switch to partitions #0, OK
mmc2(part 0) is current device
u-boot=> setexpr nblk ${filesize} / 0x200
u-boot=> mmc write ${loadaddr} 0x0 ${nblk}
```

(continues on next page)

(continued from previous page)

```
MMC write: dev # 2, block # 0, count 1024000 ... 1024000 blocks written: OK
u-boot=> boot
```

Flash eMMC from USB in Linux

These steps will show how to flash the eMMC on Linux with a USB stick. You only need a complete image saved on the USB stick and a bootable WIC image. (e.g. phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic). Set the *bootmode switch (S1)* to SD Card.

- Insert and mount the USB stick:

```
[   60.458908] usb-storage 1-1.1:1.0: USB Mass Storage device detected
[   60.467286] scsi host0: usb-storage 1-1.1:1.0
[   61.504607] scsi 0:0:0:0: Direct-Access                     8.07 PQ: 0 ANSI: 2
[   61.515283] sd 0:0:0:0: [sda] 3782656 512-byte logical blocks: (1.94 GB/1.80 GiB)
[   61.523285] sd 0:0:0:0: [sda] Write Protect is off
[   61.528509] sd 0:0:0:0: [sda] No Caching mode page found
[   61.533889] sd 0:0:0:0: [sda] Assuming drive cache: write through
[   61.665969]  sda: sda1
[   61.672284] sd 0:0:0:0: [sda] Attached SCSI removable disk
target:~$ mount /dev/sda1 /mnt
```

- Now show your saved image files on the USB Stick:

```
target:~$ ls /mnt
phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic
phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic.bmap
```

- Write the image to the phyCORE-i.MX 8M Mini eMMC (MMC device 2 without partition):

```
target:~$ bmaptool copy /mnt/phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic /dev/mmcblk2
```

- After a complete write, your board can boot from eMMC.

> **Tip**
>
> Before this will work, you need to configure the *bootmode switch (S1)* to **eMMC**.

4.2.4 Flash eMMC from SD Card

Even if there is no network available, you can update the eMMC. For that, you only need a ready-to-use image file (`*.wic`) located on the SD card. Because the image file is quite large, you have to create a third partition. To create a new partition or enlarge your SD card, see *Resizing ext4 Root Filesystem*.

Alternatively, flash a partup package to the SD card, as described in *Getting Started*. This will ensure the full space of the SD card is used.

Flash eMMC from SD card in U-Boot on Target

Tip

This step only works if the size of the image file is less than 1GB due to limited usage of RAM size in the Bootloader after enabling OPTEE. If the image file is too large use the *Updating eMMC from SD card in Linux on Target* subsection.

- Flash an SD card with a working image and create a third ext4 partition. Copy the WIC image (for example phytec-qt6demo-image.wic) to this partition.
- Configure the *bootmode switch (S1)* to SD Card and insert the SD Card.
- Power on the board and stop in U-Boot.
- Load the image:

```
u-boot=> ext4load mmc 1:3 ${loadaddr} phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic
reading
911842304 bytes read in 39253 ms (22.2 MiB/s)
```

- Switch the mmc dev to eMMC:

```
u-boot=> mmc list
FSL_SDHC: 1 (SD)
FSL_SDHC: 2 (eMMC)
u-boot=> mmc dev 2
switch to partitions #0, OK
mmc2(part 0) is current device
```

- Flash your WIC image (for example phytec-qt6demo-image.wic) from the SD card to eMMC. This will partition the card and copy imx-boot, Image, dtb, dtbo, and root file system to eMMC.

```
u-boot=> setexpr nblk ${filesize} / 0x200
u-boot=> mmc write ${loadaddr} 0x0 ${nblk}

MMC write: dev # 2, block # 0, count 1780942 ... 1780942 blocks written: OK
```

- Power off the board and change the *bootmode switch (S1)* to eMMC.

Flash eMMC from SD card in Linux on Target

You can also flash the eMMC on Linux. You only need a partup package or WIC image saved on the SD card.

- Show your saved partup package or WIC image files on the SD card:

```
target:~$ ls
phytec-qt6demo-image-phyboard-polis-imx8mm-5.partup
phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic
phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic.bmap
```

- Write the image to the phyCORE-i.MX 8M Mini eMMC (MMC device 2 **without** partition) using partup:

```
target:~$ partup install phytec-qt6demo-image-phyboard-polis-imx8mm-5.partup /dev/mmcblk2
```

Flashing the partup package has the advantage of using the full capacity of the eMMC device, adjusting partitions accordingly.

Note

Alternatively, `bmaptool` may be used instead:

```
target:~$ bmaptool copy phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic /dev/mmcblk2
```

Keep in mind that the root partition does not make use of the full space when flashing with `bmaptool`.

- After a complete write, your board can boot from eMMC.

Warning

Before this will work, you need to configure the *bootmode switch (S1)* to eMMC.

4.3 Flash SPI NOR Flash

The phyCORE-i.MX8MM modules are optionally equipped with SPI NOR Flash. To boot from SPI Flash, set *bootmode switch (S1)* to **SPI NOR**. The SPI Flash is usually quite small. The phyBOARD-Pollux-i.MX8MP kit only has 32MB SPI NOR flash populated. Only the bootloader and the environment can be stored. The kernel, device tree, and file system are taken from eMMC by default.

The SPI NOR flash partition table is defined in the U-Boot environment. It can be printed with:

```
u-boot=> printenv mtdparts
mtdparts=30bb0000.spi:3840k(u-boot),128k(env),128k(env:redund),-(none)
```

4.3.1 Flash SPI NOR Flash from Network

The SPI NOR can contain the bootloader and environment to boot from. The arm64 kernel can not decompress itself, the image size extends the SPI NOR flash populated on the phyCORE-i.MX 8M Mini.

Tip

A working network is necessary! *Setup Network Host*

Flash SPI NOR from Network in U-Boot on Target

Similar to updating the eMMC over a network, be sure to set up the development host correctly. The IP needs to be set to 192.168.3.10, the netmask to 255.255.255.0, and a TFTP server needs to be available. Before reading and writing is possible, the SPI NOR flash needs to be probed:

```
u-boot=> sf probe
SF: Detected mt25qu512a with page size 256 Bytes, erase size 64 KiB, total 64 MiB
```

- A specially formatted U-Boot image for the SPI NOR flash is used. Ensure you use the correct image file. Load the image over tftp, erase and write the bootloader to the flash:

```
u-boot=> tftp ${loadaddr} imx-boot-phyboard-polis-imx8mm-5-fspi.bin-flash_evk_flexspi
u-boot=> sf update ${loadaddr} 0 ${filesize}
device 0 offset 0x0, size 0x1c0b20
1641248 bytes written, 196608 bytes skipped in 4.768s, speed 394459 B/s
```

- Erase the environment partition as well. This way, the environment can be written after booting from SPI NOR flash:

```
u-boot=> sf erase 0x400000 0x100000
```

Flash SPI NOR from Network in kernel on Target

- Copy the image from the host to the target:

```
host:~$ scp imx-boot-phyboard-polis-imx8mm-5-fspi.bin-flash_evk_flexspi root@192.168.3.11:/
↪root
```

- Find the number of blocks to erase of the U-boot partition:

```
target:~$ mtdinfo /dev/mtd0
mtd0
Name:                           u-boot
Type:                           nor
Eraseblock size:                65536 bytes, 64.0 KiB
Amount of eraseblocks:          60 (3932160 bytes, 3.7 MiB)
Minimum input/output unit size: 1 byte
Sub-page size:                  1 byte
Character device major/minor:   90:0
Bad blocks are allowed:         false
Device is writable:             true
```

- Erase the U-Boot partition and flash it:

```
target:~$ flash_erase /dev/mtd0 0x0 60
target:~$ flashcp imx-boot-phyboard-polis-imx8mm-5-fspi.bin-flash_evk_flexspi /dev/mtd0
```

4.3.2 Flash SPI NOR Flash from SD Card

The bootloader on SPI NOR flash can be also flashed with SD Card.

Flash SPI NOR from SD Card in U-Boot on Target

- Copy the SPI NOR flash U-boot image imx-boot-phyboard-polis-imx8mm-5-fspi.bin-flash_evk_flexspi to the first partition on the SD Card.
- Before reading and writing are possible, the SPI-NOR flash needs to be probed:

```
u-boot=> sf probe
SF: Detected n25q256ax1 with page size 256 Bytes, erase size 64 KiB, total 32 MiB
```

- A specially formatted U-boot image for the SPI NOR flash is used. Ensure you use the correct image file. Load the image from the SD Card, erase and write the bootloader to the flash:

```
u-boot=> mmc dev 1
u-boot=> fatload mmc 1:1 ${loadaddr} imx-boot-phyboard-polis-imx8mm-5-fspi.bin-flash_evk_
↪flexspi
u-boot=> sf update ${loadaddr} 0 ${filesize}
```

- Erase the environment partition as well. This way, the environment can be written after booting from SPI NOR flash:

```
u-boot=> sf erase 0x400000 0x100000
```

Flash SPI NOR from SD Card in kernel on Target

- Copy the SPI NOR flash U-boot image imx-boot-phyboard-polis-imx8mm-5-fspi.bin-flash_evk_flexspi to the first partition on the SD Card.
- Mount the SD Card:

```
target:~$ mount /dev/mmcblk1p1 /mnt
```

- Find the number of blocks to erase of the U-Boot partition:

```
target:~$ mtdinfo /dev/mtd0
mtd0
Name:                           u-boot
Type:                           nor
Eraseblock size:                65536 bytes, 64.0 KiB
Amount of eraseblocks:          60 (3932160 bytes, 3.7 MiB)
Minimum input/output unit size: 1 byte
Sub-page size:                  1 byte
Character device major/minor:   90:0
Bad blocks are allowed:         false
Device is writable:             true
```

- Erase the u-boot partition and flash it:

```
target:~$ flash_erase /dev/mtd0 0x0 60
target:~$ flashcp /mnt/imx-boot-phyboard-polis-imx8mm-5-fspi.bin-flash_evk_flexspi /dev/mtd0
```

4.4 RAUC

The RAUC (Robust Auto-Update Controller) mechanism support has been added to meta-ampliphy. It controls the procedure of updating a device with new firmware. This includes updating the Linux kernel, Device Tree, and root filesystem. PHYTEC has written an online manual on how we have intergraded RAUC into our BSPs: L-1006e.A5 RAUC Update & Device Management Manual.

CHAPTER

FIVE

DEVELOPMENT

5.1 Host Network Preparation

For various tasks involving a network in the Bootloader, some host services are required to be set up. On the development host, a TFTP, NFS and DHCP server must be installed and configured. The following tools will be needed to boot via Ethernet:

```
host:~$ sudo apt install tftpd-hpa nfs-kernel-server kea
```

5.1.1 TFTP Server Setup

- First, create a directory to store the TFTP files:

  ```
  host:~$ sudo mkdir /srv/tftp
  ```

- Then copy your BSP image files to this directory and make sure other users have read access to all the files in the tftp directory, otherwise they are not accessible from the target.

  ```
  host:~$ sudo chmod -R o+r /srv/tftp
  ```

- You also need to configure a static IP address for the appropriate interface. The default IP address of the PHYTEC evaluation boards is 192.168.3.11. Setting a host address 192.168.3.10 with netmask 255.255.255.0 is a good choice.

  ```
  host:~$ ip addr show <network-interface>
  ```

  Replace <network-interface> with the network interface you configured and want to connect the board to. You can show all network interfaces by not specifying a network interface.

- The message you receive should contain this:

  ```
  inet 192.168.3.10/24 brd 192.168.3.255
  ```

- Create or edit the `/etc/default/tftpd-hpa` file:

  ```
  # /etc/default/tftpd-hpa

  TFTP_USERNAME="tftp"
  TFTP_DIRECTORY="/srv/tftp"
  TFTP_ADDRESS=":69"
  TFTP_OPTIONS="-s -c"
  ```

- Set TFTP_DIRECTORY to your TFTP server root directory

- Set TFTP_ADDRESS to the host address the server is listening to (set to 0.0.0.0:69 to listen to all local IPs)
- Set TFTP_OPTIONS, the following command shows the available options:

```
host:~$ man tftpd
```

- Restart the services to pick up the configuration changes:

```
host:~$ sudo service tftpd-hpa restart
```

Now connect the ethernet port of the board to your host system. 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.

NFS Server Setup

- Create an nfs directory:

```
host:~$ sudo mkdir /srv/nfs
```

- The NFS server is not restricted to a certain file system location, so all we have to do on most distributions is modify the file `/etc/exports` and export our root file system to the embedded network. In this example file, the whole directory is exported and the "lab network" address of the development host is 192.168.3.10. The IP address has to be adapted to the local needs:

```
/srv/nfs 192.168.3.0/255.255.255.0(rw,no_root_squash,sync,no_subtree_check)
```

- Now the NFS-Server has to read the `/etc/exportfs` file again:

```
host:~$ sudo exportfs -ra
```

DHCP Server setup

- Create or edit the `/etc/kea/kea-dhcp4.conf` file; Using the internal subnet sample. Replace <network-interface> with the name for the physical network interface:

```
{
  "Dhcp4": {
    "interfaces-config": {
      "interfaces": [ "<network-interface>/192.168.3.10" ]
    },
    "lease-database": {
      "type": "memfile",
      "persist": true,
      "name": "/tmp/dhcp4.leases"
    },
    "valid-lifetime": 28800,
    "subnet4": [{
        "id": 1,
        "next-server": "192.168.3.10",
        "subnet": "192.168.3.0/24",
        "pools": [
          { "pool": "192.168.3.1 - 192.168.3.255" }
        ]
```

(continues on next page)

(continued from previous page)

```
    }]
  }
}
```

Warning

Be careful when creating subnets as this may interfere with the company network policy. To be on the safe side, use a different network and specify that via the `interfaces` configuration option.

- Now the DHCP-Server has to read the `/etc/kea/kea-dhcp4.conf` file again:

```
host:~$ sudo systemctl restart kea-dhcp4-server
```

When you boot/restart your host PC and don't have the network interface, as specified in the kea-dhcp4 config, already active the kea-dhcp4-server will fail to start. Make sure to start/restart the systemd service when you connect the interface.

5.2 Booting the Kernel from a Network

Booting from a network means loading the kernel and device tree over TFTP and the root file system over NFS. The bootloader itself must already be loaded from another available boot device.

5.2.1 Place Images on Host for Netboot

- Copy the kernel image to your tftp directory:

```
host:~$ cp Image /srv/tftp
```

- Copy the devicetree to your tftp directory:

```
host:~$ cp oftree /srv/tftp
```

- Copy all the overlays you want to use into your tftp directory:

```
host:~$ cp *.dtbo /srv/tftp
```

- Make sure other users have read access to all the files in the tftp directory, otherwise they are not accessible from the target:

```
host:~$ sudo chmod -R o+r /srv/tftp
```

- Extract the rootfs to your nfs directory:

```
host:~$ sudo tar -xvzf phytec-qt6demo-image-phyboard-polis-imx8mm-5.tar.gz -C /srv/nfs
```

Note

Make sure you extract with sudo to preserve the correct ownership.

5.2.2 Set the bootenv.txt for Netboot

Create a bootenv.txt file in your tftp directory and write the following variables into it.

```
bootfile=Image
fdt_file=oftree
nfsroot=/srv/nfs
overlays=<overlayfilenames>
```

<overlayfilenames> has to be replaced with the devicetree overlay filenames that you want to use. Separate the filenames by spaces. For example:

```
overlays=example-overlay1.dtbo example-overlay2.dtbo
```

Tip

All supported devicetree overlays are in the *device tree* chapter.

5.2.3 Network Settings on Target

To customize the targets ethernet configuration, please follow the description here: *Network Environment Customization*

5.2.4 Booting from an Embedded Board

Boot the board into the U-boot prompt and press any key to hold.

- To boot from a network, call:

```
u-boot=> run netboot
```

5.3 Working with UUU-Tool

The Universal Update Utility Tool (UUU-Tool) from NXP is a software to execute on the host to load and run the bootloader on the board through SDP (Serial Download Protocol). For detailed information visit https://github.com/nxp-imx/mfgtools or download the Official UUU-tool documentation.

5.3.1 Host preparations for UUU-Tool Usage

- Follow the instructions from https://github.com/nxp-imx/mfgtools#linux.
- If you built UUU from source, add it to `PATH`:

  This BASH command adds UUU only temporarily to `PATH`. To add it permanently, add this line to `~/.bashrc`.

```
export PATH=~/mfgtools/uuu/:"$PATH"
```

- Set udev rules (documented in `uuu -udev`):

```
host:~$ sudo sh -c "uuu -udev >> /etc/udev/rules.d/70-uuu.rules"
host:~$ sudo udevadm control --reload
```

5.3.2 Get Images

Download imx-boot from our server or get it from your Yocto build directory at build/deploy/images/phyboard-polis-imx8mm-5/. For flashing a wic image to eMMC, you will also need phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic.

5.3.3 Prepare Target

Set the *bootmode switch (S1)* to **USB Serial Download**. Also, connect USB port *X2* to your host.

5.3.4 Starting bootloader via UUU-Tool

Execute and power up the board:

```
host:~$ sudo uuu -b spl imx-boot
```

You can see the bootlog on the console via *(X30)*, as usual.

Note

The default boot command when booting with UUU-Tool is set to fastboot. If you want to change this, please adjust the environment variable bootcmd_mfg in U-boot prompt with setenv bootcmd_mfg. Please note, when booting with UUU-tool the default environment is loaded. Saveenv has no effect. If you want to change the boot command permanently for UUU-boot, you need to change this in U-Boot code.

5.3.5 Flashing U-boot Image to eMMC via UUU-Tool

Warning

UUU flashes U-boot into eMMC BOOT (hardware) boot partitions, and it sets the BOOT_PARTITION_ENABLE in the eMMC! This is a problem since we want the bootloader to reside in the eMMC USER partition. Flashing next U-Boot version .wic image and not disabling BOOT_PARTITION_ENABLE bit will result in device always using U-boot saved in BOOT partitions. To fix this in U-Boot:

```
u-boot=> mmc partconf 2 0 0 0
u-boot=> mmc partconf 2
EXT_CSD[179], PARTITION_CONFIG:
BOOT_ACK: 0x0
BOOT_PARTITION_ENABLE: 0x0
PARTITION_ACCESS: 0x0
```

or check Disable booting from eMMC boot partitions from Linux.

This way the bootloader is still flashed to eMMC BOOT partitions but it is not used!

When using **partup** tool and `.partup` package for eMMC flashing this is done by default, which makes partup again superior flash option.

Execute and power up the board:

```
host:~$ sudo uuu -b emmc imx-boot
```

5.3.6 Flashing wic Image to eMMC via UUU-Tool

Execute and power up the board:

```
host:~$ sudo uuu -b emmc_all imx-boot phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic
```

5.4 Standalone Build preparation

In this section, we describe how to build the U-Boot and the Linux kernel without using the Yocto Project. This procedure makes the most sense for development. The U-Boot source code, the Linux kernel, and all other git repositories are available on our Git server at git://git.phytec.de.

Note

Should your company firewall/gateway inhibit the git protocol, you may use HTTP or HTTPS instead (e.g. git clone git://git.phytec.de/u-boot-imx)

5.4.1 Git Repositories

- Used U-Boot repository:

  ```
  git://git.phytec.de/u-boot-imx
  ```

- Our U-Boot is based on the u-boot-imx and adds board-specific patches.
- Used Linux kernel repository:

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

- Our i.MX 8M Mini kernel is based on the linux-imx kernel.

To find out which u-boot and kernel tags to use for a specific board, have a look at your BSP source folder:

```
meta-phytec/dynamic-layers/freescale-layer/recipes-kernel/linux/linux-imx_*.bb
meta-phytec/recipes-bsp/u-boot/u-boot-imx_*.bb
```

5.4.2 Get the SDK

You can download the SDK here, or build it yourself with Yocto:

- Move to the Yocto build directory:

  ```
  host:~$ source sources/poky/oe-init-build-env
  host:~$ bitbake -c populate_sdk phytec-qt6demo-image # or another image
  ```

After a successful build the SDK installer is deployed to `build/deploy*/sdk`.

5.4.3 Install the SDK

- Set correct permissions and install the SDK:

  ```
  host:~$ chmod +x phytec-ampliphy-vendor-xwayland-glibc-x86_64-phytec-qt6demo-image-
  ↪cortexa53-crypto-toolchain-4.0.13.sh
  host:~$ ./phytec-ampliphy-vendor-xwayland-glibc-x86_64-phytec-qt6demo-image-cortexa53-
  ```

(continues on next page)

(continued from previous page)

```
↪crypto-toolchain-4.0.13.sh
==========================================================================================================================
Enter target directory for SDK (default: /opt/ampliphy-vendor-xwayland/4.0.13):
You are about to install the SDK to "/opt/ampliphy-vendor-xwayland/4.0.13". Proceed [Y/n]? Y
Extracting SDK...done
Setting it up...done
SDK has been successfully set up and is ready to be used.
```

5.4.4 Using the SDK

Activate the toolchain for your shell by sourcing the *environment-setup* file in the toolchain directory:

```
host:~$ source /opt/ampliphy-vendor-xwayland/4.0.13/environment-setup-cortexa53-crypto-phytec-
↪linux
```

5.4.5 Installing Required Tools

Building Linux and U-Boot out-of-tree requires some additional host tool dependencies to be installed. For Ubuntu you can install them with:

```
host:~$ sudo apt install bison flex libssl-dev
```

5.5 U-Boot standalone build

Note

The regulator for the SD card reset pin has been disabled to ensure compatibility with 1532.1 revision baseboards. If you have a revision 1532.2(a) or higher baseboard, you may enable the device tree nodes for highest performance. In the imx8mm-phyboard-polis-rdk-u-boot.dtsi file, remove the following lines:

```
/delete-node/ &reg_usdhc2_vmmc;
/delete-property/ vmmc-supply;
```

5.5.1 Get the source code

- Get the U-Boot sources:

  ```
  host:~$ git clone git://git.phytec.de/u-boot-imx
  ```

- To get the correct *U-Boot* **tag** you need to take a look at our release notes, which can be found here: release notes
- The **tag** used in this release is called v2022.04_2.2.2-phy5
- Check out the needed *U-Boot* **tag**:

  ```
  host:~$ cd ~/u-boot-imx/
  host:~/u-boot-imx$ git fetch --all --tags
  host:~/u-boot-imx$ git checkout tags/v2022.04_2.2.2-phy5
  ```

- Set up a build environment:

```
host:~/u-boot-imx$ source /opt/ampliphy-vendor-xwayland/4.0.13/environment-setup-cortexa53-
↪crypto-phytec-linux
```

5.5.2 Get the needed binaries

To build the bootloader, you need to **copy** these **files** to your u-boot-imx **build directory** and rename them to fit with *mkimage* script:

- **ARM Trusted firmware binary** (*mkimage tool* compatible format **bl31.bin**): bl31-imx8mm.bin
- **OPTEE image** (optional): tee.bin
- **DDR firmware files** (*mkimage tool* compatible format **lpddr4_[i,d]mem_*d_*.bin**): lpddr4_dmem_1d_*.bin, lpddr4_dmem_2d_*.bin, lpddr4_imem_1d_*.bin, lpddr4_imem_2d_*.bin

If you already built our BSP with Yocto, you can get the bl31-imx8mm.bin, tee.bin and lpddr4_*.bin from the directory mentioned here: *BSP Images*

Or you can download the files here: https://download.phytec.de/Software/Linux/BSP-Yocto-i.MX8MM/BSP-Yocto-NXP-i.MX8MM-PD23.1.0/images/ampliphy-vendor-xwayland/phyboard-polis-imx8mm-5/imx-boot-tools/

Warning

Make sure you rename the files you need so that they are compatible with the *mkimage tool*.

5.5.3 Build the bootloader

- build flash.bin (imx-boot):

```
host:~/u-boot-imx$ make phycore-imx8mm_defconfig
host:~/u-boot-imx$ make flash.bin
```

5.5.4 Flash the bootloader to a block device

The flash.bin can be found at u-boot-imx/ directory and now can be flashed. A chip-specific offset is needed:

SoC	Offset User Area	Offset Boot Partition	eMMC Device
i.MX 8M Mini	33 kiB	33 kiB	/dev/mmcblk2

E.g. flash SD card:

```
host:~/u-boot-imx$ sudo dd if=flash.bin of=/dev/sd[x] bs=1024 seek=33 conv=sync
```

Hint

The specific offset values are also declared in the Yocto variables “BOOTLOADER_SEEK” and “BOOTLOADER_SEEK_EMMC”

5.5.5 Build U-Boot With a Fixed RAM Size

If you cannot boot your system anymore because the hardware introspection in the EEPROM is damaged or deleted, you can create a flash.bin with a fixed ram size. You should still contact support and flash the correct EEPROM data, as this could lead to unexpected behavior.

Follow the steps to get the U-boot sources and check the correct branch in the **Build U-Boot** section.

Edit the file configs/phycore-imx8mm_defconfig:

```
CONFIG_TARGET_PHYCORE_IMX8MM=y
CONFIG_PHYCORE_IMX8MM_RAM_SIZE_FIX=y
# CONFIG_PHYCORE_IMX8MM_RAM_SIZE_1GB=y
# CONFIG_PHYCORE_IMX8MM_RAM_SIZE_2GB=y
# CONFIG_PHYCORE_IMX8MM_RAM_SIZE_4GB=y
```

Choose the correct RAM size as populated on the board and uncomment the line for this ram size. After saving the changes, follow the remaining steps from *Build U-Boot*.

5.6 Kernel standalone build

5.6.1 Setup sources

- The used linux-imx branch can be found in the release notes
- The tag needed for this release is called v5.15.71_2.2.2-phy3
- Check out the needed linux-imx tag:

```
host:~$ git clone git://git.phytec.de/linux-imx
host:~$ cd ~/linux-imx/
host:~/linux-imx$ git fetch --all --tags
host:~/linux-imx$ git checkout tags/v5.15.71_2.2.2-phy3
```

- For committing changes, it is highly recommended to switch to a new branch:

```
host:~/linux-imx$ git switch --create <new-branch>
```

- Set up a build environment:

```
host:~/linux-imx$ source /opt/ampliphy-vendor-xwayland/4.0.13/environment-setup-cortexa53-
↪crypto-phytec-linux
```

5.6.2 Build the kernel

- Build the linux kernel:

```
host:~/linux-imx$ make imx_v8_defconfig imx8_phytec_distro.config imx8_phytec_platform.
↪config
host:~/linux-imx$ make -j$(nproc)
```

- Install kernel modules to e.g. NFS directory:

```
host:~/linux-imx$ make INSTALL_MOD_PATH=/home/<user>/<rootfspath> modules_install
```

- The Image can be found at ~/linux-imx/arch/arm64/boot/Image

- The dtb can be found at ~/linux-imx/arch/arm64/boot/dts/freescale/imx8mm-phyboard-polis-rdk.dtb
- For (re-)building only Devicetrees and -overlays, it is sufficient to run

```
host:~/linux-imx$ make dtbs
```

Note

If you are facing the following build issue:

```
scripts/dtc/yamltree.c:9:10: fatal error: yaml.h: No such file or directory
```

Make sure you installed the package *"libyaml-dev"* on your host system:

```
host:~$ sudo apt install libyaml-dev
```

5.6.3 Copy Kernel to SD Card

When one-time boot via netboot is not sufficient, the kernel along with its modules and the corresponding device tree blob may be copied directly to a mounted SD card.

```
host:~/linux-imx$ cp arch/arm64/boot/Image /path/to/sdcard/boot/
host:~/linux-imx$ cp arch/arm64/boot/dts/freescale/imx8mm-phyboard-polis-rdk.dtb /path/to/sdcard/
↪boot/oftree
host:~/linux-imx$ make INSTALL_MOD_PATH=/path/to/sdcard/root/ modules_install
```

5.7 Accessing the Development states

5.7.1 Development state of current release

These release manifests exist to give you access to the development states of the *Yocto* BSP. They will not be displayed in the phyLinux selection menu but need to be selected manually. This can be done using the following command line:

```
host:~$ ./phyLinux init -p imx8mm -r BSP-Yocto-NXP-i.MX8MM-PD23.1.y
```

This will initialize a BSP that will track the latest development state of the current release (BSP-Yocto-NXP-i.MX8MM-PD23.1.0). From now on *repo sync* in this folder will pull all the latest changes from our Git repositories:

```
host:~$ repo sync
```

5.7.2 Development state of upcoming release

Also development states of upcoming releases can be accessed this way. For this execute the following command and look for a release with a higher PDXX.Y number than the latest one (BSP-Yocto-NXP-i.MX8MM-PD23.1.0) and `.y` at the end:

```
host:~$ ./phyLinux init -p imx8mm
```

5.8 Accessing the Latest Upstream Support

We have a vanilla manifest that makes use of the Yocto master branches (not an NXP release), Linux, and U-Boot. This can be used to test the latest upstream kernel/U-Boot.

Note

The master manifest reflects the latest state of development. This tends to be broken from time to time. We try to fix the master on a regular basis.

```
host:~$ ./phyLinux init -p imx8mm -r BSP-Yocto-Ampliphy-i.MX8MM-master
```

5.9 Format SD-Card

Most images are larger than the default root partition. To flash any storage device with SD Card, the rootfs needs to be expanded or a separate partition needs to be created. There are some different ways to format the SD Card. The easiest way to do this is to use the UI program Gparted.

5.9.1 Gparted

- Get GParted:

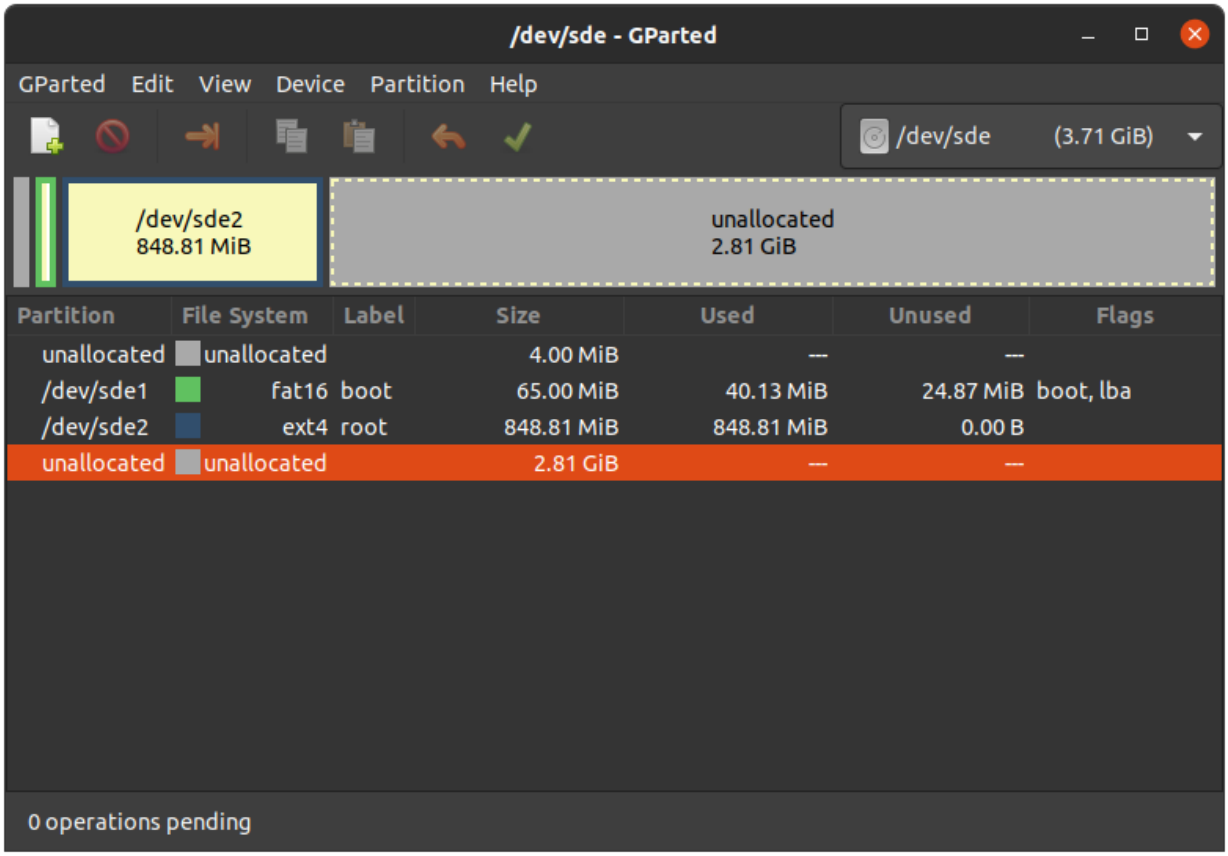
  ```
  host:~$ sudo apt install gparted
  ```

- Insert the SD Card into your host and get the device name:

  ```
  host:~$ dmesg | tail
  ...
  [30436.175412] sd 4:0:0:0: [sdb] 62453760 512-byte logical blocks: (32.0 GB/29.8 GiB)
  [30436.179846]  sdb: sdb1 sdb2
  ...
  ```

- Unmount all SD Card partitions.
- Launch GParted:

  ```
  host:~$ sudo gparted
  ```

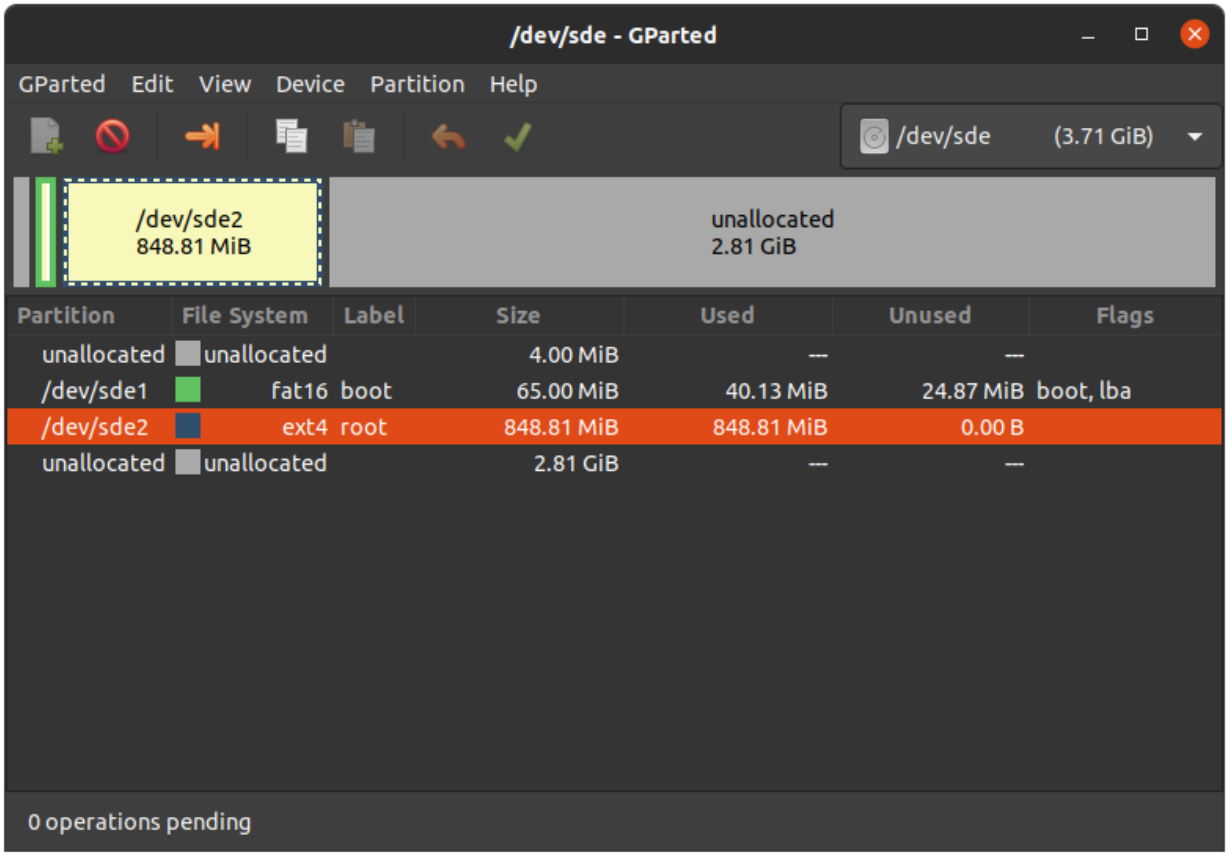


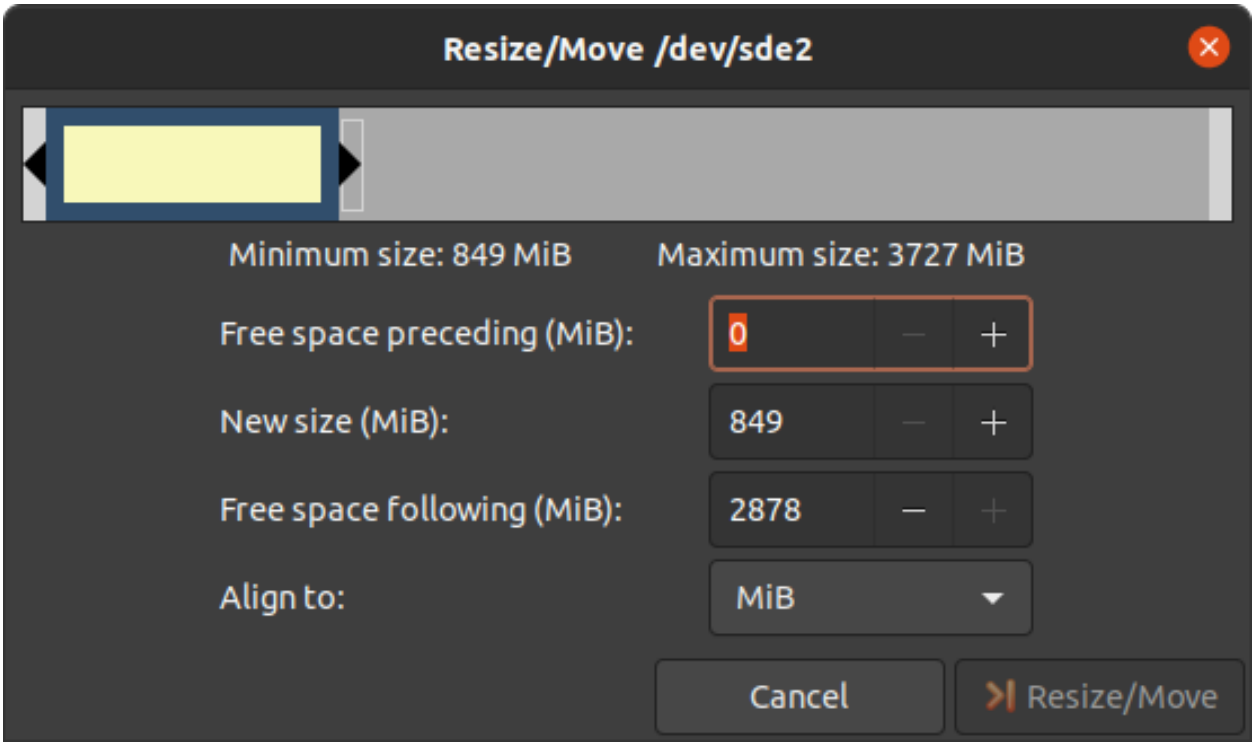
Expand rootfs

Warning

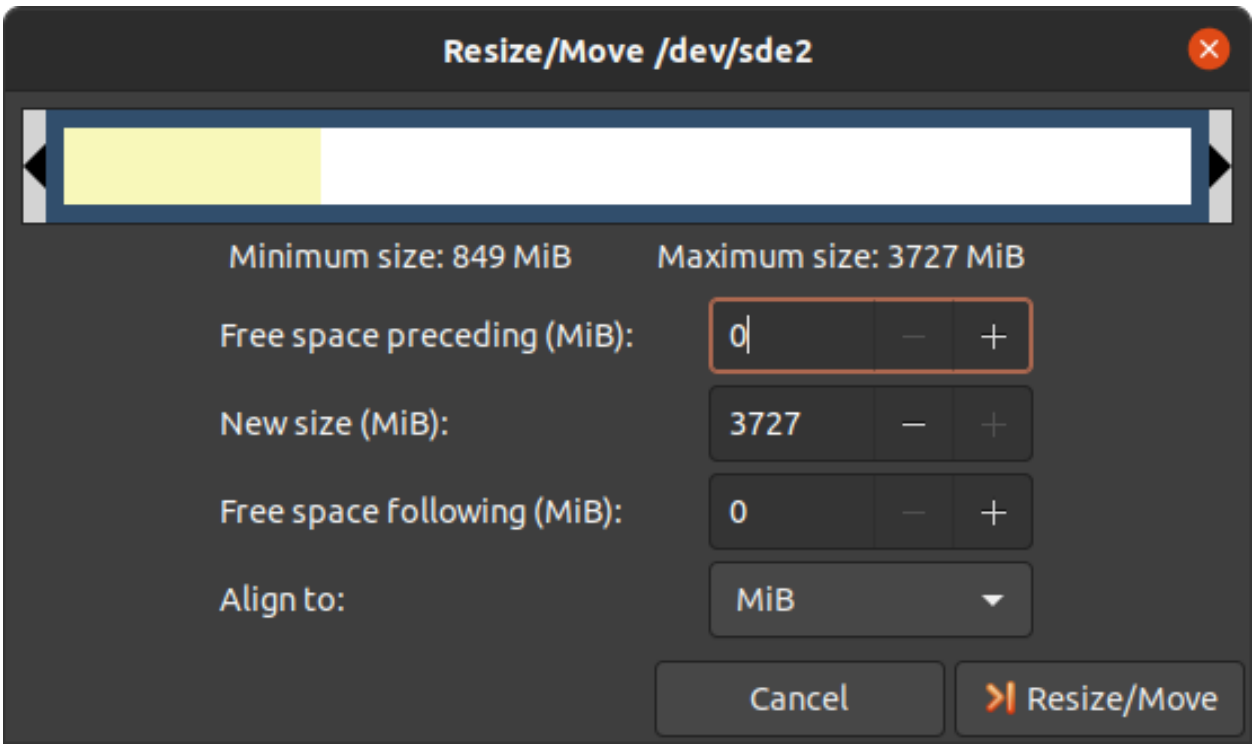
Running gparted on host systems which are using resize2fs version 1.46.6 and older (e.g. Ubuntu 22.04) are not able to expand the ext4 partition created with Yocto Mickledore and newer. This is due to a new default option in resize2fs which causes a incompatibility. See release notes.

- Choose your SD Card device at the drop-down menu on the top right
- Choose the ext4 root partition and click on resize:

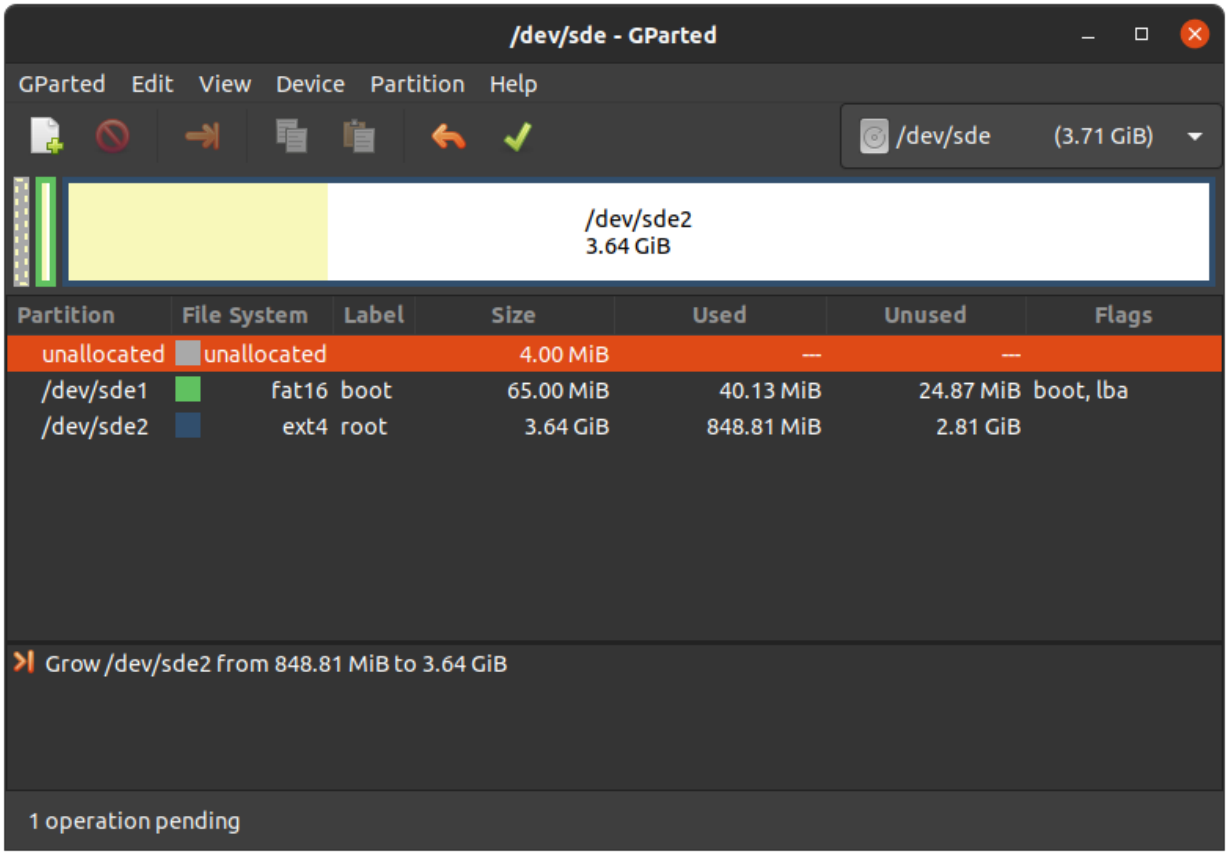




- Drag the slider as far as you like or enter the size manually.



- Confirm your entry by clicking on the “Change size” button.



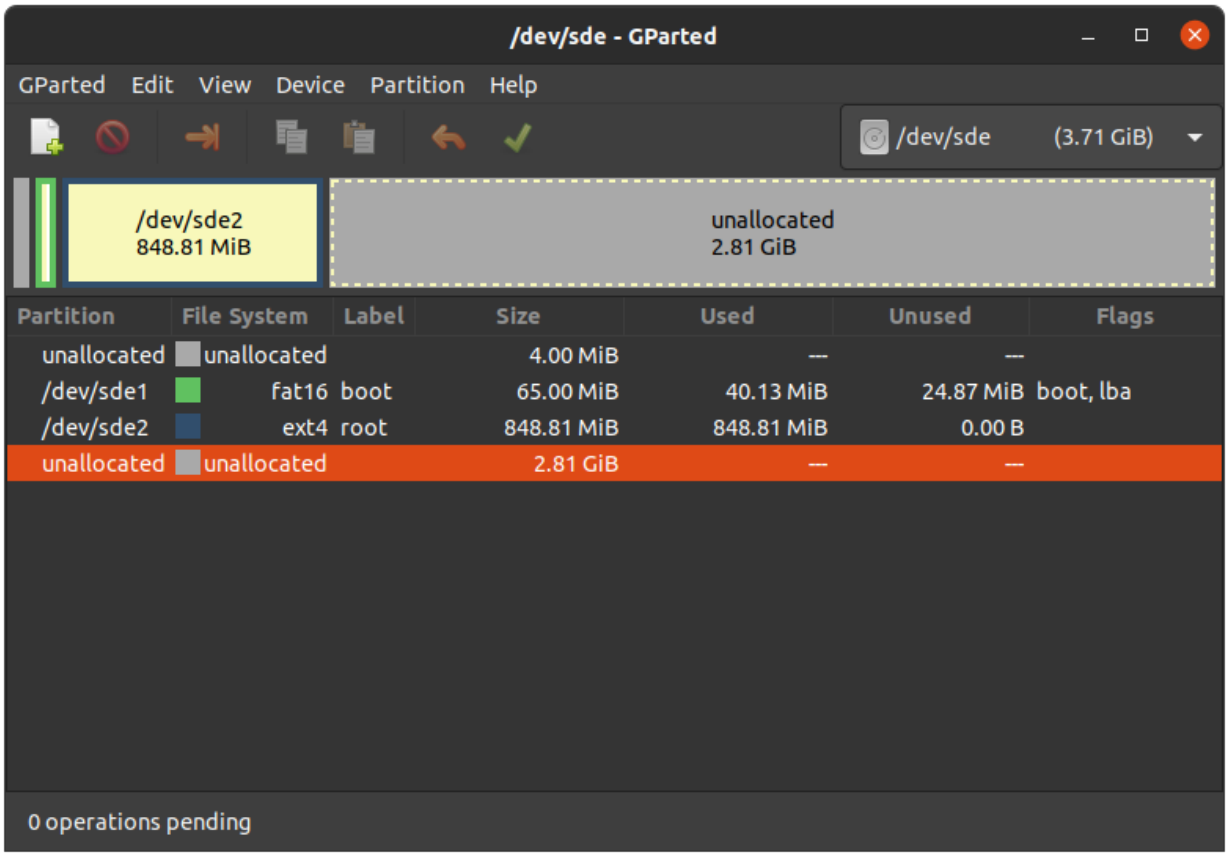
- To apply your changes, press the green tick.

- Now you can mount the root partition and copy e.g. the phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic image to it. Then unmount it again:

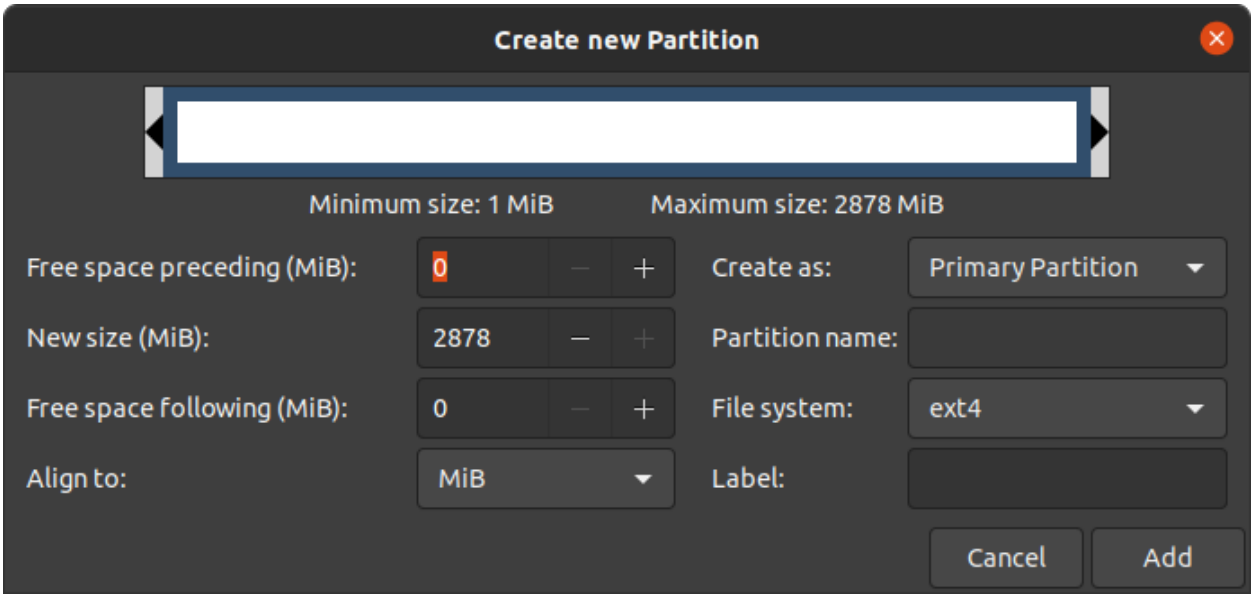
```
host:~$ sudo cp phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic /mnt/ ; sync
host:~$ umount /mnt
```

Create the Third Partition

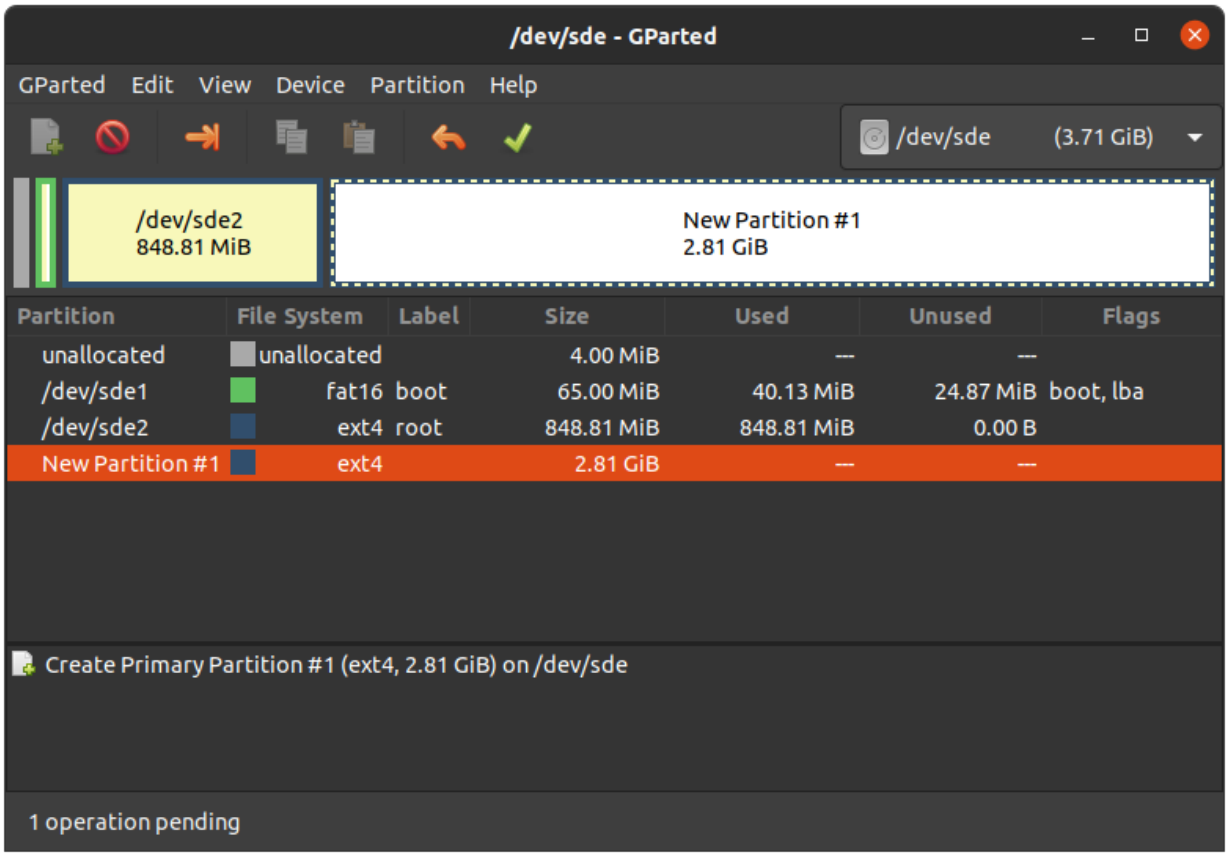
- Choose your SD Card device at the drop-down menu on the top right



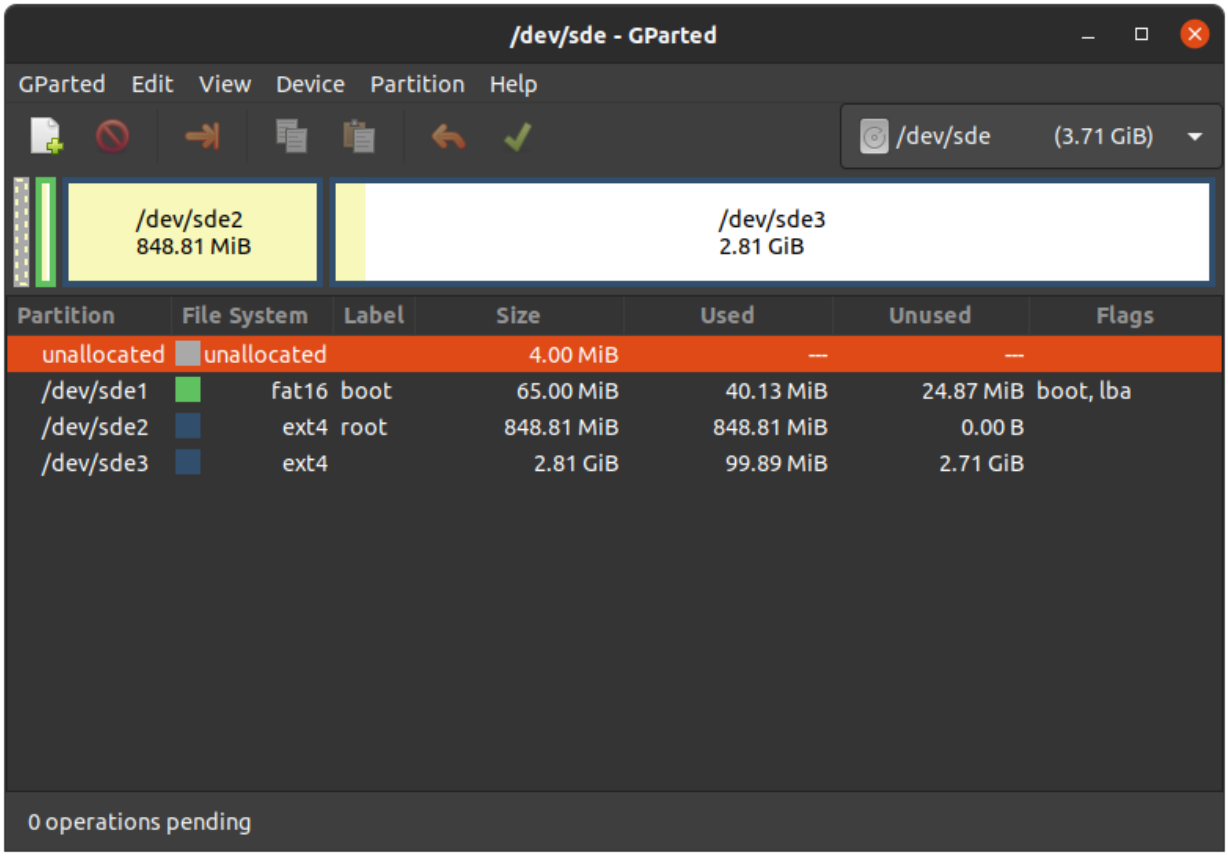
- Choose the bigger unallocated area and press “New”:



- Click “Add”



- Confirm your changes by pressing the green tick.



- Now you can mount the new partition and copy e.g. phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic image to it. Then unmount it again:

```
host:~$ sudo mount /dev/sde3 /mnt
host:~$ sudo cp phytec-qt6demo-image-phyboard-polis-imx8mm-5.wic /mnt/ ; sync
host:~$ umount /mnt
```

CHAPTER

SIX

DEVICE TREE (DT)

6.1 Introduction

The following text briefly describes the Device Tree and can be found in the Linux kernel Documentation (https://docs.kernel.org/devicetree/usage-model.html)

"The "Open Firmware Device Tree", or simply Devicetree (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."

The kernel documentation is a really good source for a DT introduction. An overview of the device tree data format can be found on the device tree usage page at devicetree.org.

6.2 PHYTEC i.MX 8M Mini BSP Device Tree Concept

The following sections explain some rules PHYTEC has defined on how to set up device trees for our i.MX 8M Mini SoC-based boards.

6.2.1 Device Tree Structure

- *Module.dtsi* - Module includes all devices mounted on the SoM, such as PMIC and RAM.
- *Board.dts* - include the module dtsi file. Devices that come from the i.MX 8M Mini SoC but are just routed down to the carrier board and used there are included in this dts.
- *Overlay.dtso* - enable/disable features depending on optional hardware that may be mounted or missing on SoM or baseboard (e.g SPI flash or PEB-AV-10)

From the root directory of the Linux Kernel our devicetree files for i.MX 8 platforms can be found in `arch/arm64/boot/dts/freescale/`.

6.2.2 Device Tree Overlay

Device Tree overlays are device tree fragments that can be merged into a device tree during boot time. These are for example hardware descriptions of an expansion board. They are instead of being added to the device tree as an extra include, now applied as an overlay. They also may only contain setting the status of a node depending on if it is mounted or not. The device tree overlays are placed next to the other device tree files in our Linux kernel repository in the subfolder `arch/arm64/boot/dts/freescale/overlays`.

Available overlays for phyboard-polis-imx8mm-5.conf are:

```
imx8mm-phyboard-polis-peb-eval-01.dtbo
imx8mm-phyboard-polis-peb-av-010.dtbo
```

(continues on next page)

(continued from previous page)

```
imx8mm-phycore-rpmsg.dtbo
imx8mm-phycore-no-eth.dtbo
imx8mm-phycore-no-spiflash.dtbo
imx8mm-vm016.dtbo
imx8mm-vm016-fpdlink.dtbo
imx8mm-vm017.dtbo
imx8mm-vm017-fpdlink.dtbo
imx8mm-dual-vm017-fpdlink.dtbo
```

The usage of overlays can be configured during runtime in Linux or U-Boot. Overlays are applied during the boot process in the bootloader after the boot command is called and before the kernel is loaded. The next sections explain the configuration in more detail.

Set `${overlays}` variable

The `${overlays}` U-Boot environment variable contains a space-separated list of overlays that will be applied during boot. Depending on the boot source the overlays have to either be placed in the boot partition of eMMC/SD-Card or are loaded over tftp. Overlays set in the $KERNEL_DEVICETREE Yocto machine variable will be automatically added to the boot partition of the final WIC image.

The `${overlays}` variable can be either set directly in the U-Boot environment or can be part of the external `bootenv.txt` environment file. By default, the `${overlays}` variable comes from the external `bootenv.txt` environment file which is located in the boot partition. You can read and write the file on booted target from linux:

```
target:~$ cat /boot/bootenv.txt
overlays=imx8mm-phyboard-polis-rdk-peb-eval-01.dtbo imx8mm-phyboard-polis-peb-av-010.dtbo
```

Changes will take effect after the next reboot. If no `bootenv.txt` file is available the overlays variable can be set directly in the U-Boot environment.

```
u-boot=> setenv overlays imx8mm-phyboard-polis-peb-av-010.dtbo
u-boot=> printenv overlays
overlays=imx8mm-phyboard-polis-peb-av-010.dtbo
u-boot=> boot
```

If a user defined `${overlays}` variable should be directly loaded from U-Boot environment but there is still an external `bootenv.txt` available, the `${no_bootenv}` variable needs to be set as a flag:

```
u-boot=> setenv no_bootenv 1
u-boot=> setenv overlays imx8mm-phyboard-polis-peb-av-010.dtbo
u-boot=> boot
```

More information about the external environment can be found in U-boot External Environment subsection of the *device tree overlay section*.

We use the `${overlays}` variable for overlays describing expansion boards and cameras that can not be detected during run time. To prevent applying overlays listed in the `${overlays}` variable during boot, the `${no_overlays}` variable can be set to *1* in the bootloader environment.

```
u-boot=> setenv no_overlays 1
u-boot=> boot
```

Extension Command

The U-Boot extension command makes it possible to easily apply overlays that have been detected and added to a list by the board code callback extension_board_scan(). Overlays applied this way disable components that are not populated on the SoM. The detection is done with the EEPROM data (EEPROM SoM Detection) found on the SoM i2c EEPROM.

It depends on the SoM variant if any device tree overlays will be applied. To check if an overlay will be applied on the running SoM in U-Boot, run:

```
u-boot=> extension scan
Found 1 extension board(s).
u-boot=> extension list
Extension 0: phyCORE-i.MX8MM no SPI flash
        Manufacturer:           PHYTEC
        Version:
        Devicetree overlay:     imx8mm-phycore-no-spiflash.dtbo
        Other information:      SPI flash not populated on SoM
```

If the EEPROM data is not available, no device tree overlays are applied.

To prevent running the extension command during boot the `${no_extensions}` variable can be set to *1* in the bootloader environment:

```
u-boot=> setenv no_extensions 1
u-boot=> boot
```

6.2.3 U-boot External Environment

During the start of the Linux Kernel the external environment `bootenv.txt` text file will be loaded from the boot partition of an MMC device or via TFTP. The main intention of this file is to store the `${overlays}` variable. This makes it easy to pre-define the overlays in Yocto depending on the used machine. The content from the file is defined in the Yocto recipe bootenv found in meta-phytec: https://git.phytec.de/meta-phytec/tree/recipes-bsp/bootenv?h=kirkstone

Other variables can be set in this file, too. They will overwrite the existing settings in the environment. But only variables evaluated after issuing the boot command can be overwritten, such as `${nfsroot}` or `${mmcargs}`. Changing other variables in that file will not have an effect. See the usage during netboot as an example.

If the external environment can not be loaded the boot process will be anyway continued with the values of the existing environment settings.

6.2.4 Change U-boot Environment from Linux on Target

Libubootenv is a tool included in our images to modify the U-Boot environment of Linux on the target machine.

Print the U-Boot environment using the following command:

```
target:~$ fw_printenv
```

Modify a U-Boot environment variable using the following command:

```
target:~$ fw_setenv <variable> <value>
```

Caution

Libubootenv takes the environment selected in a configuration file. The environment to use is inserted there, and by default it is configured to use the eMMC environment (known as the default used environment).

If the eMMC is not flashed or the eMMC environment is deleted, libubootenv will not work. You should modify the `/etc/fw_env.config` file to match the environment source that you want to use.

CHAPTER

SEVEN

ACCESSING PERIPHERALS

To find out which boards and modules are supported by the release of PHYTEC's phyCORE-i.MX 8M Mini BSP described herein, visit our BSP web page and click the corresponding BSP release in the download section. Here you can find all hardware supported in the columns "Hardware Article Number" and the correct machine name in the corresponding cell under "Machine Name".

To achieve maximum software reuse, the Linux kernel offers a sophisticated infrastructure that layers software components into board-specific parts. The BSP tries to modularize the kit features as much as possible. When a customized baseboard or even a customer-specific module is developed, most of the software support can be re-used without error-prone copy-and-paste. The kernel code corresponding to the boards can be found in device trees (DT) in the kernel repository under `arch/arm64/boot/dts/freescale/*.dts`.

In fact, software reuse is one of the most important features of the Linux kernel, especially of the ARM implementation which always has to fight with an insane number of possibilities of the System-on-Chip CPUs. The whole board-specific hardware is described in DTs and is not part of the kernel image itself. The hardware description is in its own separate binary, called the Device Tree Blob (DTB) (section *device tree*).

Please read section PHYTEC i.MX 8M Mini BSP Device Tree Concept to get an understanding of our i.MX 8 BSP device tree model.

The following sections provide an overview of the supported hardware components and their operating system drivers on the i.MX 8 platform. Further changes can be ported upon customer request.

7.1 i.MX 8M Mini Pin Muxing

The i.MX 8M Mini SoC contains many peripheral interfaces. In order to reduce package size and lower overall system cost while maintaining maximum functionality, many of the i.MX 8M Mini terminals can multiplex up to eight signal functions. Although there are many combinations of pin multiplexing that are possible, only a certain number of sets, called IO sets, are valid due to timing limitations. These valid IO sets were carefully chosen to provide many possible application scenarios for the user.

Please refer to our Hardware Manual or the NXP i.MX 8M Mini Reference Manual for more information about the specific pins and the muxing capabilities.

The IO set configuration, also called muxing, is done in the Device Tree. The driver pinctrl-single reads the DT's node fsl,pins, and does the appropriate pin muxing.

The following is an example of the pin muxing of the UART1 device in imx8mm-phyboard-polis-rdk.dts:

```
pinctrl_uart1: uart1grp {
        fsl,pins = <
                MX8MM_IOMUXC_SAI2_RXFS_UART1_DCE_TX     0x00
                MX8MM_IOMUXC_SAI2_RXC_UART1_DCE_RX      0x00
                MX8MM_IOMUXC_SAI2_RXD0_UART1_DCE_RTS_B  0x00
```

(continues on next page)

(continued from previous page)

```
                MX8MM_IOMUXC_SAI2_TXFS_UART1_DCE_CTS_B  0x00
        >;
};
```

The first part of the string MX8MM_IOMUXC_SAI2_RXFS_UART1_DCE_TX names the pad (in this example SAI2_RXFS). The second part of the string (UART1_DCE_RX) is the desired muxing option for this pad. The pad setting value (hex value on the right) defines different modes of the pad, for example, if internal pull resistors are activated or not. In this case, the internal resistors are disabled.

7.2 RS232/RS485

The i.MX 8M Mini SoC provides up to 4 UART units. PHYTEC boards support different numbers of these UART units. UART1 can also be used as RS-485. For this, *bootmode switch (S1)* needs to be set correctly:

Table 1: Switch between UART1 RS485/RS232

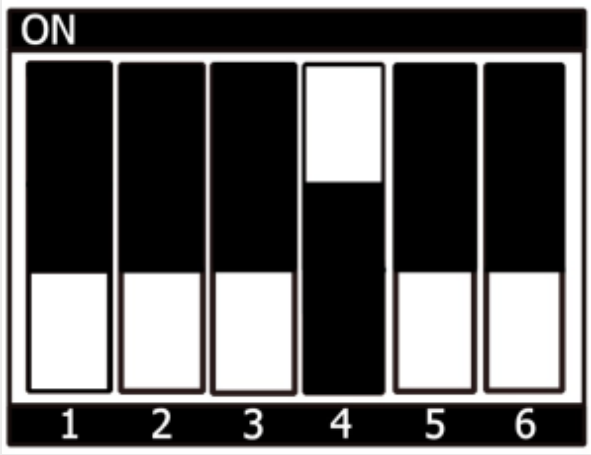


Fig. 1: **UART1 RS485**

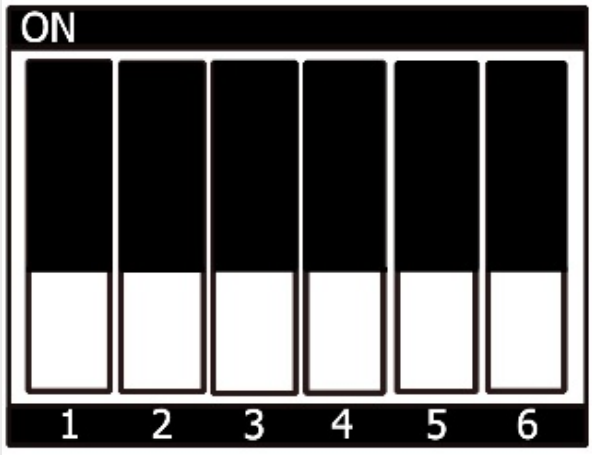


Fig. 2: **UART1 RS232**

7.2.1 RS232

- Display the current settings of a terminal in a human-readable format:

```
target:~$ stty -a
```

- Configuration of the UART interface can be done with stty. For example:

```
target:~$ stty -F /dev/ttymxc0 115200 crtscts raw -echo
```

- With a simple echo and cat, basic communication can be tested. Example:

```
target:~$ echo 123 > /dev/ttymxc0
```

```
host:~$ cat /dev/ttyUSB2
```

The host should print out “123”.

7.2.2 RS485

Hint

Remember to use bus termination resistors of 120 Ohm at each end of the bus, when using longer cables.

For easy testing, look at the linux-serial-test. This tool is called the IOCTL for RS485 and sends a constant stream of data.

```
target:~$ linux-serial-test -p /dev/ttymxc0 -b 115200 --rs485 0
```

More information about the linux-serial-test tool and its parameters can be found here: linux-serial-test

The linux-serial-test will automatically set ioctls, but they can also be set manually with rs485conf.

You can show the current config with:

```
target:~$ rs485conf /dev/ttymxc1
```

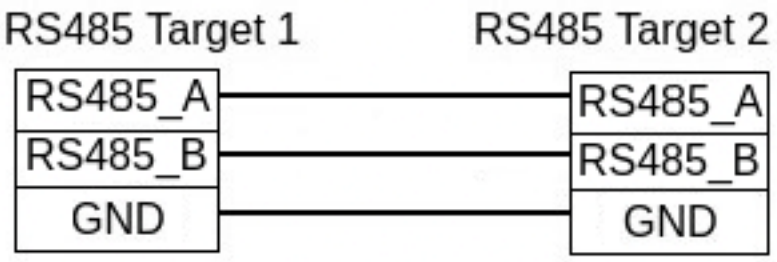
You can show all options with:

```
target:~$ rs485conf /dev/ttymxc1 -h
```

Documentation for calling the IOCTL within c-code is described in the Linux kernel documentation: https://www.kernel.org/doc/Documentation/serial/serial-rs485.txt

RS485 half-duplex

For half-duplex mode your connection setup should look like this:



Which function is on which pin is described in the hardware manual.

For half-duplex mode you can set the ioctls manually like this:

```
target:~$ rs485conf /dev/ttymxc1 -e 1 -r 0
target:~$ rs485conf /dev/ttymxc1
= Current configuration:
RS485 enabled:                true
RTS on send:                  high
RTS after send:               low
RTS delay before send:        0
```

(continues on next page)

(continued from previous page)

```
RTS delay after send:         0
Receive during sending data:  false
Bus termination enabled:  false
```

Then you can test if sending and receiving works like this:

```
target1:~$ cat /dev/ttymxc1
target2:~$ echo test > /dev/ttymxc1
```

You should see "test" printed out on target1. You can also switch the roles and send on target2 and receive on target1.

Alternatively you can also test with the linux-serial-test tool:

```
target1:~$ linux-serial-test -s -e -f -p /dev/ttymxc1 -b 115200 --rs485 0 -t -i 8
...
/dev/ttymxc1: count for this session: rx=57330, tx=0, rx err=0
target2:~$ linux-serial-test -s -e -f -p /dev/ttymxc1 -b 115200 --rs485 0 -r -o 5
...
/dev/ttymxc1: count for this session: rx=0, tx=57330, rx err=0
```

In this example target1 will be the receiver and target2 will be the transmitter. You should also be able to switch the roles. Remember to first start the receiver and then the transmitter immediately after. The receiver will receive for 8 sec and the transmitter will send for 5 sec. The receiver needs to receive for a bit longer than the transmitter sends. At the end the program will print the final "count for this session". There you can check, that all transmitted frames were received.

All the tests are target to target, but can also be done with host to target with a USB to rs485 converter. You may need to adjust the interfaces then.

The device tree representation for RS232 and RS485: https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/imx8mm-phyboard-polis-rdk.dts?h=v5.15.71_2.2.2-phy3#n291

7.3 Network

phyBOARD-Polis-i.MX 8M Mini provides one Gigabit Ethernet interface.

All interfaces offer a standard Linux network port that can be programmed using the BSD socket interface. The whole network configuration is handled by the systemd-networkd daemon. The relevant configuration files can be found on the target in `/lib/systemd/network/` as well as the BSP in `meta-ampliphy/recipes-core/systemd/systemd-conf`.

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

```
eth0: 192.168.3.11/24
```

The DT Ethernet setup might be split into two files depending on your hardware configuration: the module DT and the board-specific DT. The device tree set up for the FEC ethernet IP core where the ethernet PHY is populated on the SoM can be found here: https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/imx8mm-phycore-som.dtsi?h=v5.15.71_2.2.2-phy3#n59.

7.3.1 Network Environment Customization

U-boot network-environment

- To find the Ethernet settings in the target bootloader:

```
u-boot=> printenv ipaddr serverip netmask
```

- With your development host set to IP 192.168.3.10 and netmask 255.255.255.0, the target should return:

```
u-boot=> printenv ipaddr serverip netmask
ipaddr=192.168.3.11
serverip=192.168.3.10
netmask=255.225.255.0
```

- If you need to make any changes:

```
u-boot=> setenv <parameter> <value>
```

  <parameter> should be one of ipaddr, netmask, gatewayip or serverip. <value> will be the actual value of the chosen parameter.

- The changes you made are temporary for now. To save these:

```
u-boot=> saveenv
```

Here you can also change the IP address to DHCP instead of using a static one.

- Configure:

```
u-boot=> setenv ip dhcp
```

- Set up paths for TFTP and NFS. A modification could look like this:

```
u-boot=> setenv nfsroot /home/user/nfssrc
```

Please note that these modifications will only affect the bootloader settings.

Tip

Variables like nfsroot and netargs can be overwritten by the U-boot External Environment. For netboot, the external environment will be loaded from tftp. For example, to locally set the nfsroot variable in a `bootenv.txt` file, locate the tftproot directory:

```
nfsroot=/home/user/nfssrc
```

There is no need to store the info on the target. Please note that this does not work for variables like ipaddr or serveraddr as they need to be already set when the external environment is being loaded.

Kernel network-environment

- Find the ethernet settings for eth0 in the target kernel:

```
target:~$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 50:2D:F4:19:D6:33
```

(continues on next page)

(continued from previous page)

```
          UP BROADCAST MULTICAST  MTU:1500  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:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

- Temporary adaption of the eth0 configuration:

```
target:~$ ifconfig eth0 192.168.3.11 netmask 255.255.255.0 up
```

7.3.2 WLAN

For WLAN and Bluetooth support, we use the Sterling-LWB module from LSR. This module supports 2,4 GHz bandwidth and can be run in several modes, like client mode, Access Point (AP) mode using WEP, WPA, WPA2 encryption, and more. More information about the module can be found at https://connectivity-staging.s3.us-east-2.amazonaws.com/2019-09/CS-DS-SterlingLWB%20v7_2.pdf

Connecting to a WLAN Network

First set the correct regulatory domain for your country:

```
target:~$ iw reg set DE
target:~$ iw reg get
```

You will see:

```
country DE: DFS-ETSI
   (2400 - 2483 @ 40), (N/A, 20), (N/A)
   (5150 - 5250 @ 80), (N/A, 20), (N/A), NO-OUTDOOR
   (5250 - 5350 @ 80), (N/A, 20), (0 ms), NO-OUTDOOR, DFS
   (5470 - 5725 @ 160), (N/A, 26), (0 ms), DFS
   (57000 - 66000 @ 2160), (N/A, 40), (N/A)
```

Set up the wireless interface:

```
target:~$ ip link
target:~$ ip link set up dev wlan0
```

Now you can scan for available networks:

```
target:~$ iw wlan0 scan | grep SSID
```

You can use a cross-platform supplicant with support for WEP, WPA, and WPA2 called wpa_supplicant for an encrypted connection.

To do so, add the network-credentials to the file `/etc/wpa_supplicant.conf`:

```
country=DE
network={
    ssid="<SSID>"
    proto=WPA2
    psk="<KEY>"
}
```

Now a connection can be established:

```
target:~$ wpa_supplicant -D nl80211 -c /etc/wpa_supplicant.conf -i wlan0 -B
```

This should result in the following output:

```
...
ENT-CONNECTED - Connection to 88:33:14:5d:db:b1 completed [id=0 id_str=]
```

To finish the configuration you can configure DHCP to receive an IP address (supported by most WLAN access points). For other possible IP configurations, see section *Changing the Network Configuration* in the Yocto Reference Manual (kirkstone).

First, create the directory:

```
target:~$ mkdir -p /etc/systemd/network/
```

Then add the following configuration snippet in `/etc/systemd/network/10-wlan0.network`:

```
# file /etc/systemd/network/10-wlan0.network
[Match]
Name=wlan0

[Network]
DHCP=yes
```

Now, restart the network daemon so that the configuration takes effect:

```
target:~$ systemctl restart systemd-networkd
```

7.3.3 Bluetooth

Bluetooth is connected to UART2 interface. More information about the module can be found at https://connectivity-staging.s3.us-east-2.amazonaws.com/2019-09/CS-DS-SterlingLWB%20v7_2.pdf. The Bluetooth device needs to be set up manually:

```
target:~$ hciconfig hci0 up

target:~$ hciconfig -a

hci0:   Type: Primary  Bus: UART
        BD Address: 00:25:CA:2F:39:96  ACL MTU: 1021:8  SCO MTU: 64:1
        UP RUNNING PSCAN
        RX bytes:1392 acl:0 sco:0 events:76 errors:0
        TX bytes:1198 acl:0 sco:0 commands:76 errors:0
        ...
```

Now you can scan your environment for visible Bluetooth devices. Bluetooth is not visible during a default startup.

```
target:~$ hcitool scan
Scanning ...
       XX:XX:XX:XX:XX:XX       <SSID>
```

Visibility

To activate visibility:

```
target:~$ hciconfig hci0 piscan
```

To disable visibility:

```
target:~$ hciconfig hci0 noscan
```

Connect

```
target:~$ bluetoothctl
[bluetooth]# discoverable on
Changing discoverable on succeeded
[bluetooth]# pairable on
Changing pairable on succeeded
[bluetooth]# agent on
Agent registered
[bluetooth]# default-agent
Default agent request successful
[bluetooth]# scan on
[NEW] Device XX:XX:XX:XX:XX:XX <name>
[bluetooth]# connect XX:XX:XX:XX:XX:XX
```

Note

If the connection fails with the error message: "Failed to connect: org.bluez.Error.Failed" try restarting PulseAudio with:

```
target:~$ pulseaudio --start
```

7.4 SD/MMC Card

The i.MX 8M Mini supports a slot for Secure Digital Cards and MultiMedia Cards to be used as general-purpose block devices. These devices can be used in the same way as any other block device.

Warning

These kinds of devices are hot-pluggable. Nevertheless, you must ensure not to unplug the device while it is still mounted. This may result in data loss!

After inserting an SD/MMC card, the kernel will generate new device nodes in /dev. The full device can be reached via its /dev/mmcblk1 device node. SD/MMC card partitions will show up as:

```
/dev/mmcblk1p<Y>
```

<Y> counts as the partition number starting from 1 to the max count of partitions on this device. The partitions can be formatted with any kind of file system and also handled in a standard manner, e.g. the mount and umount command work as expected.

Tip

These partition device nodes will only be available if the card contains a valid partition table (”hard disk” like handling). If no partition table is present, the whole device can be used as a file system (”floppy” like handling). In this case, /dev/mmcblk1 must be used for formatting and mounting. The cards are always mounted as being writable.

DT configuration for the MMC (SD card slot) interface can be found here: https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/imx8mm-phyboard-polis-rdk.dts?h=v5.15.71_2.2.2-phy3#n383

DT configuration for the eMMC interface can be found here: https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/imx8mm-phycore-som.dtsi?h=v5.15.71_2.2.2-phy3#n335

7.5 eMMC Devices

PHYTEC modules like phyCORE-i.MX 8M Mini is populated with an eMMC memory chip as the main storage. eMMC devices contain raw Multi-Level Cells (MLC) or Triple-Level Cells (TLC) combined with a memory controller that handles ECC and wear leveling. They are connected via an SD/MMC interface to the i.MX 8M Mini and are represented as block devices in the Linux kernel like SD cards, flash drives, or hard disks.

The electric and protocol specifications are provided by JEDEC (https://www.jedec.org/standards-documents/technology-focus-areas/flash-memory-ssds-ufs-emmc/e-mmc). The eMMC manufacturer’s datasheet is relatively short and meant to be read together with the supported version of the JEDEC eMMC standard.

PHYTEC currently utilizes the eMMC chips with JEDEC Version 5.0 and 5.1

7.5.1 Extended CSD Register

eMMC devices have an extensive amount of extra information and settings that are available via the Extended CSD registers. For a detailed list of the registers, see manufacturer datasheets and the JEDEC standard.

In the Linux user space, you can query the registers:

```
target:~$ mmc extcsd read /dev/mmcblk2
```

You will see:

```
=============================================
   Extended CSD rev 1.7 (MMC 5.0)
=============================================

 Card Supported Command sets [S_CMD_SET: 0x01]
 [...]
```

7.5.2 Enabling Background Operations (BKOPS)

In contrast to raw NAND Flash, an eMMC device contains a Flash Transfer Layer (FTL) that handles the wear leveling, block management, and ECC of the raw MLC or TLC. This requires some maintenance tasks (for example erasing unused blocks) that are performed regularly. These tasks are called **Background Operations (BKOPS)**.

By default (depending on the chip), the background operations may or may not be executed periodically, impacting the worst-case read and write latency.

The JEDEC Standard has specified a method since version v4.41 that the host can issue BKOPS manually. See the JEDEC Standard chapter Background Operations and the description of registers BKOPS_EN (Reg: 163) and BKOPS_START (Reg: 164) in the eMMC datasheet for more details.

Meaning of Register BKOPS_EN (Reg: 163) Bit MANUAL_EN (Bit 0):

- Value 0: The host does not support the manual trigger of BKOPS. Device write performance suffers.
- Value 1: The host does support the manual trigger of BKOPS. It will issue BKOPS from time to time when it does not need the device.

The mechanism to issue background operations has been implemented in the Linux kernel since v3.7. You only have to enable BKOPS_EN on the eMMC device (see below for details).

The JEDEC standard v5.1 introduces a new automatic BKOPS feature. It frees the host to trigger the background operations regularly because the device starts BKOPS itself when it is idle (see the description of bit AUTO_EN in register BKOPS_EN (Reg: 163)).

- To check whether *BKOPS_EN* is set, execute:

  ```
  target:~$ mmc extcsd read /dev/mmcblk2 | grep BKOPS_EN
  ```

  The output will be, for example:

  ```
  Enable background operations handshake [BKOPS_EN]: 0x01
  #OR
  Enable background operations handshake [BKOPS_EN]: 0x00
  ```

  Where value 0x00 means BKOPS_EN is disabled and device write performance suffers. Where value 0x01 means BKOPS_EN is enabled and the host will issue background operations from time to time.

- Enabling can be done with this command:

  ```
  target:~$ target:~$ mmc --help

  [...]
  mmc bkops_en <auto|manual> <device>
      Enable the eMMC BKOPS feature on <device>.
      The auto (AUTO_EN) setting is only supported on eMMC 5.0 or newer.
      Setting auto won't have any effect if manual is set.
      NOTE!  Setting manual (MANUAL_EN) is one-time programmable (unreversible) change.
  ```

- To set the BKOPS_EN bit, execute:

  ```
  target:~$ mmc bkops_en manual /dev/mmcblk2
  ```

- To ensure that the new setting is taken over and the kernel triggers BKOPS by itself, shut down the system:

  ```
  target:~$ poweroff
  ```

Tip

The BKOPS_EN bit is one-time programmable only. It cannot be reversed.

7.5.3 Reliable Write

There are two different Reliable Write options:

1. Reliable Write option for a whole eMMC device/partition.
2. Reliable Write for single write transactions.

Tip

Do not confuse eMMC partitions with partitions of a DOS, MBR, or GPT partition table (see the previous section).

The first Reliable Write option is mostly already enabled on the eMMCs mounted on the phyCORE-i.MX 8M Mini SoMs. To check this on the running target:

```
target:~$ mmc extcsd read /dev/mmcblk2 | grep -A 5 WR_REL_SET
Write reliability setting register [WR_REL_SET]: 0x1f
 user area: the device protects existing data if a power failure occurs during a write o
peration
 partition 1: the device protects existing data if a power failure occurs during a write
 operation
 partition 2: the device protects existing data if a power failure occurs during a write
 operation
 partition 3: the device protects existing data if a power failure occurs during a write
 operation
 partition 4: the device protects existing data if a power failure occurs during a write
 operation
--
 Device supports writing EXT_CSD_WR_REL_SET
 Device supports the enhanced def. of reliable write
```

Otherwise, it can be enabled with the mmc tool:

```
target:~$ mmc --help

[...]
mmc write_reliability set <-y|-n|-c> <partition> <device>
    Enable write reliability per partition for the <device>.
    Dry-run only unless -y or -c is passed.
    Use -c if more partitioning settings are still to come.
    NOTE!  This is a one-time programmable (unreversible) change.
```

The second Reliable Write option is the configuration bit Reliable Write Request parameter (bit 31) in command CMD23. It has been used in the kernel since v3.0 by file systems, e.g. ext4 for the journal and user space applications such as fdisk for the partition table. In the Linux kernel source code, it is handled via the flag REQ_META.

Conclusion: ext4 file system with mount option data=journal should be safe against power cuts. The file system check can recover the file system after a power failure, but data that was written just before the power cut may be lost. In any case, a consistent state of the file system can be recovered. To ensure data consistency for the files of an application, the system functions fdatasync or fsync should be used in the application.

7.5.4 Resizing ext4 Root Filesystem

When flashing the sdcard image to eMMC the ext4 root partition is not extended to the end of the eMMC. parted can be used to expand the root partition. The example works for any block device such as eMMC, SD card, or hard disk.

- Get the current device size:

```
target:~$ parted /dev/mmcblk2 print
```

- The output looks like this:

```
Model: MMC Q2J55L (sd/mmc)
Disk /dev/mmcblk2: 7617MB
Sect[ 1799.850385]  mmcblk2: p1 p2
or size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type     File system  Flags
 1      4194kB  72.4MB  68.2MB  primary  fat16        boot, lba
 2      72.4MB  537MB   465MB   primary  ext4
```

- Use parted to resize the root partition to the max size of the device:

```
target:~$ parted /dev/mmcblk2 resizepart 2 100%
Information: You may need to update /etc/fstab.

target:~$ parted /dev/mmcblk2 print
Model: MMC Q2J55L (sd/mmc)
Disk /dev/mmcblk2: 7617MB
Sector size (logical/physical): 512[ 1974.191657]  mmcblk2: p1 p2
B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type     File system  Flags
 1      4194kB  72.4MB  68.2MB  primary  fat16        boot, lba
 2      72.4MB  7617MB  7545MB  primary  ext4
```

- Resize the filesystem to a new partition size:

```
target:~$ resize2fs /dev/mmcblk2p2
resize2fs 1.46.1 (9-Feb-2021)
Filesystem at /dev/mmcblk2p2 is mounted on /; on-line resizing required
[ 131.609512] EXT4-fs (mmcblk2p2): resizing filesystem
from 454136 to 7367680 blocks
old_desc_blocks = 4, new_desc_blocks = 57
[ 131.970278] EXT4-fs (mmcblk2p2): resized filesystem to 7367680
The filesystem on /dev/mmcblk2p2 is now 7367680 (1k) blocks long
```

Increasing the filesystem size can be done while it is mounted. But you can also boot the board from an SD card and then resize the file system on the eMMC partition while it is not mounted.

7.5.5 Enable pseudo-SLC Mode

eMMC devices use MLC or TLC (https://en.wikipedia.org/wiki/Multi-level_cell) to store the data. Compared with SLC used in NAND Flash, MLC or TLC have lower reliability and a higher error rate at lower costs.

If you prefer reliability over storage capacity, you can enable the pseudo-SLC mode or SLC mode. The method used here employs the enhanced attribute, described in the JEDEC standard, which can be set for continuous regions of the device. The JEDEC standard does not specify the implementation details and the guarantees of the enhanced attribute. This is left to the chipmaker. For the Micron chips, the enhanced attribute increases the reliability but also halves the capacity.

Warning

When enabling the enhanced attribute on the device, all data will be lost.

The following sequence shows how to enable the enhanced attribute.

- First obtain the current size of the eMMC device with:

```
target:~$ parted -m /dev/mmcblk2 unit B print
```

  You will receive:

```
BYT;
/dev/mmcblk2:63652757504B:sd/mmc:512:512:unknown:MMC S0J58X:;
```

  As you can see this device has 63652757504 Byte = 60704 MiB.

- To get the maximum size of the device after pseudo-SLC is enabled use:

```
target:~$ mmc extcsd read /dev/mmcblk2 | grep ENH_SIZE_MULT -A 1
```

  which shows, for example:

```
Max Enhanced Area Size [MAX_ENH_SIZE_MULT]: 0x000764
i.e. 3719168 KiB
--
Enhanced User Data Area Size [ENH_SIZE_MULT]: 0x000000
i.e. 0 KiB
```

  Here the maximum size is 3719168 KiB = 3632 MiB.

- Now, you can set enhanced attribute for the whole device, e.g. 3719168 KiB, by typing:

```
target:~$ mmc enh_area set -y 0 3719168 /dev/mmcblk2
```

  You will get:

```
Done setting ENH_USR area on /dev/mmcblk2
setting OTP PARTITION_SETTING_COMPLETED!
Setting OTP PARTITION_SETTING_COMPLETED on /dev/mmcblk2 SUCCESS
Device power cycle needed for settings to take effect.
Confirm that PARTITION_SETTING_COMPLETED bit is set using 'extcsd read' after power cycle
```

- To ensure that the new setting has taken over, shut down the system:

```
target:~$ poweroff
```

and perform a power cycle. It is recommended that you verify the settings now.

- First, check the value of ENH_SIZE_MULT which must be 3719168 KiB:

```
target:~$ mmc extcsd read /dev/mmcblk2 | grep ENH_SIZE_MULT  -A 1
```

You should receive:

```
Max Enhanced Area Size [MAX_ENH_SIZE_MULT]: 0x000764
i.e. 3719168 KiB
--
Enhanced User Data Area Size [ENH_SIZE_MULT]: 0x000764
i.e. 3719168 KiB
```

- Finally, check the size of the device:

```
target:~$ parted -m /dev/mmcblk2 unit B print
BYT;
/dev/mmcblk2:31742492672B:sd/mmc:512:512:unknown:MMC S0J58X:;
```

7.5.6 Erasing the Device

It is possible to erase the eMMC device directly rather than overwriting it with zeros. The eMMC block management algorithm will erase the underlying MLC or TLC or mark these blocks as discard. The data on the device is lost and will be read back as zeros.

- After booting from SD Card execute:

```
target:~$ blkdiscard -f --secure /dev/mmcblk2
```

The option –secure ensures that the command waits until the eMMC device has erased all blocks. The -f (force) option disables all checking before erasing and it is needed when the eMMC device contains existing partitions with data.

Tip

```
target:~$ dd if=/dev/zero of=/dev/mmcblk2 conv=fsync
```

also destroys all information on the device, but this command is bad for wear leveling and takes much longer!

7.5.7 eMMC Boot Partitions

An eMMC device contains four different hardware partitions: user, boot1, boot2, and rpmb.

The user partition is called the User Data Area in the JEDEC standard and is the main storage partition. The partitions boot1 and boot2 can be used to host the bootloader and are more reliable. Which partition the i.MX 8M Mini uses to load the bootloader is controlled by the boot configuration of the eMMC device. The partition rpmb is a small partition and can only be accessed via a trusted mechanism.

Furthermore, the user partition can be divided into four user-defined General Purpose Area Partitions. An explanation of this feature exceeds the scope of this document. For further information, see the JEDEC Standard Chapter 7.2 Partition Management.

Tip

Do not confuse eMMC partitions with partitions of a DOS, MBR, or GPT partition table.

The current PHYTEC BSP does not use the extra partitioning feature of eMMC devices. The U-Boot is flashed at the beginning of the user partition. The U-Boot environment is placed at a fixed location after the U-Boot. An MBR partition table is used to create two partitions, a FAT32 boot, and ext4 rootfs partition. They are located right after the U-Boot and the U-Boot environment. The FAT32 boot partition contains the kernel and device tree.

With eMMC flash storage it is possible to use the dedicated boot partitions for redundantly storing the bootloader. The U-Boot environment still resides in the user area before the first partition. The user area also still contains the bootloader which the image first shipped during its initialization process. Below is an example, to flash the bootloader to one of the two boot partitions and switch the boot device via userspace commands.

7.5.8 Via userspace Commands

On the host, run:

```
host:~$ scp imx-boot root@192.168.3.11:/tmp/
```

The partitions boot1 and boot2 are read-only by default. To write to them from user space, you have to disable force_ro in the sysfs.

To manually write the bootloader to the eMMC boot partitions, first disable the write protection:

```
target:~$ echo 0 > /sys/block/mmcblk2boot0/force_ro
target:~$ echo 0 > /sys/block/mmcblk2boot1/force_ro
```

Write the bootloader to the eMMC boot partitions:

```
target:~$ dd if=/tmp/imx-boot of=/dev/mmcblk2boot0
target:~$ dd if=/tmp/imx-boot of=/dev/mmcblk2boot1
```

The following table is for the offset of the i.MX 8M Mini SoC:

SoC	Offset User Area	Offset Boot Partition	eMMC Device	Bootloader Filename
i.MX 8M Mini	33 kiB	0 kiB	/dev/mmcblk2	imx-boot

After that set the boot partition from user space using the mmc tool:

(for 'boot0') :

```
target:~$ mmc bootpart enable 1 0 /dev/mmcblk2
```

(for 'boot1') :

```
target:~$ mmc bootpart enable 2 0 /dev/mmcblk2
```

To disable booting from the eMMC boot partitions simply enter the following command:

```
target:~$ mmc bootpart enable 0 0 /dev/mmcblk2
```

To choose back to the user area u-boot environment:

```
target:~$ mmc bootpart enable 7 0 /dev/mmcblk2
```

7.5.9 Resizing ext4 Root Filesystem

fdisk can be used to expand the root filesystem. The example works for any block device such as eMMC, SD Card, or hard disk.

- Get the current device size:

```
target:~$ fdisk -l /dev/mmcblk2
```

- The output looks like:

```
Disk /dev/mmcblk2: 7264 MB, 7616856064 bytes, 14876672 sectors 116224 cylinders, 4 heads,␣
↪32 sectors/track
Units: sectors of 1 * 512 = 512 bytes

Device       Boot StartCHS        EndCHS      StartLBA     EndLBA    Sectors   Size   Id␣
↪Type
/dev/mmcblk2p1 *    128,0,1       1023,3,32    16384       140779     124396   60.7M   c␣
↪Win95 FAT32 (LBA)
/dev/mmcblk2p2      1023,3,32     1023,3,32    141312      2192013    2050702  1001M  83␣
↪Linux
```

- Use fdisk to delete and create a partition with a max size of the device:

```
target:~$ fdisk /dev/mmcblk2

The number of cylinders for this disk is set to 116224.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)

Command (m for help): p
Disk /dev/mmcblk2: 7264 MB, 7616856064 bytes, 14876672 sectors
116224 cylinders, 4 heads, 32 sectors/track
Units: sectors of 1 * 512 = 512 bytes

Device       Boot   StartCHS        EndCHS      StartLBA     EndLBA    Sectors   Size   Id␣
↪Type
/dev/mmcblk2p1 *     128,0,1     1023,3,32         16384     140779     124396  60,7M    c␣
↪Win95 FAT32 (LBA)
/dev/mmcblk2p2     1023,3,32     1023,3,32        141312    2192013    2050702  1001M   83␣
↪Linux

Command (m for help): d
Partition number (1-4): 2

Command (m for help): p
Disk /dev/mmcblk2: 7264 MB, 7616856064 bytes, 14876672 sectors
```

(continues on next page)

(continued from previous page)

```
116224 cylinders, 4 heads, 32 sectors/track
Units: sectors of 1 * 512 = 512 bytes

Device       Boot StartCHS    EndCHS          StartLBA     EndLBA    Sectors  Size   Id Type
/dev/mmcblk2p1 *  128,0,1     1023,3,32          16384     140779     124396  60.7M   c␣
↪Win95 FAT32 (LBA)

Command (m for help): n
Partition type
   p   primary partition (1-4)
   e   extended
p
Partition number (1-4): 2
First sector (32-14876671, default 32): 141456
Last sector or +size{,K,M,G,T} (141456-14876671, default 14876671):
Using default value 14876671

Command (m for help): p
Disk /dev/mmcblk2: 7264 MB, 7616856064 bytes, 14876672 sectors
116224 cylinders, 4 heads, 32 sectors/track
Units: sectors of 1 * 512 = 512 bytes

Device       Boot StartCHS      EndCHS        StartLBA     EndLBA    Sectors  Size   Id Type
/dev/mmcblk2p1 *   128,0,1      1023,3,32        16384     140779     124396  60.7M   c␣
↪Win95 FAT32 (LBA)
/dev/mmcblk2p2   1023,3,32      1023,3,32       141456   14876671   14735216  7194M  83␣
↪Linux
```

Increasing the file system size can be done while it is mounted. An online resizing operation is performed. But you can also boot the board from an SD card and then resize the file system on the eMMC partition while it is not mounted. Furthermore, the board has to be rebooted so that the new partition table will be read.

7.6 SPI Master

The i.MX 8M Mini controller has a FlexSPI and an ECSPI IP core included. The FlexSPI host controller supports two SPI channels with up to 4 devices. Each channel supports Single/Dual/Quad/Octal mode data transfer (1/2/4/8 bidirectional data lines). The ECSPI controller supports 3 SPI interfaces with one dedicated chip selected for each interface. As chip selects should be realized with GPIOs, more than one device on each channel is possible.

7.6.1 SPI NOR Flash

phyCORE-i.MX 8M Mini is equipped with a QSPI NOR Flash which connects to the i.MX 8M Mini's FlexSPI interface. The QSPI NOR Flash is suitable for booting. Please see different sections for flashing and bootmode setup. Due to limited space on the SPI NOR Flash, only the bootloader is stored inside. By default, the kernel, device tree, and rootfs are taken from eMMC.

The Bootloader does detect with the help of the EEPROM Introspection data if an SPI flash is mounted or not. If no SPI flash is mounted a device tree overlay is applied with the expansion command to disable the SPI flash device tree node during boot. If no introspection data is available the SPI NOR flash node is always enabled. Find more information in the device tree overlay section.

The bootloader also passes the SPI MTD partition table to Linux by fixing up the device tree based on the mtdparts boot parameter. The default partition layout in the BSP is set to:

```
mtdparts=30bb0000.spi:3840k(u-boot),128k(env),128k(env_redund),-(none)
```

This is a bootloader environment variable that is defined here and can be changed during runtime. From Linux userspace, the NOR Flash partitions are accessible via /dev/mtd<N> devices where <N> is the MTD device number associated with the NOR flash partition to access. To find the correct MTD device number for a partition, run on the target:

```
root@phyboard-polis-imx8mm-5:~$ mtdinfo --all
Count of MTD devices:           4
Present MTD devices:            mtd0, mtd1, mtd2, mtd3
Sysfs interface supported:      yes

mtd0
Name:                           u-boot
Type:                           nor
Eraseblock size:                65536 bytes, 64.0 KiB
Amount of eraseblocks:          60 (3932160 bytes, 3.7 MiB)
Minimum input/output unit size: 1 byte
Sub-page size:                  1 byte
Character device major/minor:   90:0
Bad blocks are allowed:         false
Device is writable:             true

mtd1
Name:                           env
Type:                           nor
Eraseblock size:                65536 bytes, 64.0 KiB
Amount of eraseblocks:          2 (131072 bytes, 128.0 KiB)
Minimum input/output unit size: 1 byte
Sub-page size:                  1 byte
Character device major/minor:   90:2
Bad blocks are allowed:         false
Device is writable:             true

mtd2
Name:                           env_redund
Type:                           nor
Eraseblock size:                65536 bytes, 64.0 KiB
Amount of eraseblocks:          2 (131072 bytes, 128.0 KiB)
Minimum input/output unit size: 1 byte
Sub-page size:                  1 byte
Character device major/minor:   90:4
Bad blocks are allowed:         false
Device is writable:             true

mtd3
Name:                           none
Type:                           nor
Eraseblock size:                65536 bytes, 64.0 KiB
Amount of eraseblocks:          448 (29360128 bytes, 28.0 MiB)
Minimum input/output unit size: 1 byte
```

(continues on next page)

(continued from previous page)

```
Sub-page size:                  1 byte
Character device major/minor:   90:6
Bad blocks are allowed:         false
Device is writable:             true
```

It lists all MTD devices and the corresponding partition names. The flash node is defined inside of the SPI master node in the module DTS. The SPI node contains all devices connected to this SPI bus which is in this case only the SPI NOR Flash.

The definition of the SPI master node in the device tree can be found here:

https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/imx8mm-phycore-som.dtsi?h=v5.15.71_2.2.2-phy3#n87

7.7 GPIOs

The phyBOARD-Polis has a set of pins especially dedicated to user I/Os. Those pins are connected directly to i.MX 8M Mini pins and are muxed as GPIOs. They are directly usable in Linux userspace. The processor has organized its GPIOs into five banks of 32 GPIOs each (GPIO1 – GPIO5) GPIOs. gpiochip0, gpiochip32, gpiochip64, gpiochip96, and gpiochip128 are the sysfs representation of these internal i.MX 8M Mini GPIO banks GPIO1 – GPIO5.

The GPIOs are identified as GPIO<X>_<Y> (e.g. GPIO5_07). <X> identifies the GPIO bank and counts from 1 to 5, while <Y> stands for the GPIO within the bank. <Y> is being counted from 0 to 31 (32 GPIOs on each bank).

By contrast, the Linux kernel uses a single integer to enumerate all available GPIOs in the system. The formula to calculate the right number is:

```
Linux GPIO number: <N> = (<X> - 1) * 32 + <Y>
```

Accessing GPIOs from userspace will be done using the libgpiod. It provides a library and tools for interacting with the Linux GPIO character device. Examples of some usages of various tools:

- Detecting the gpiochips on the chip:

```
target:~$ gpiodetect
gpiochip0 [30200000.gpio] (32 lines)
gpiochip1 [30210000.gpio] (32 lines)
gpiochip2 [30220000.gpio] (32 lines)
gpiochip3 [30230000.gpio] (32 lines)
gpiochip4 [30240000.gpio] (32 lines)
```

- Show detailed information about the gpiochips. Like their names, consumers, direction, active state, and additional flags:

```
target:~$ gpioinfo gpiochip0
```

- Read the value of a GPIO (e.g GPIO 20 from chip0):

```
target:~$ gpioget gpiochip0 20
```

- Set the value of GPIO 20 on chip0 to 0 and exit tool:

```
target:~$ gpioset --mode=exit gpiochip0 20=0
```

- Help text of gpioset shows possible options:

```
target:~$ gpioset --help
Usage: gpioset [OPTIONS] <chip name/number> <offset1>=<value1> <offset2>=<value2> ...
Set GPIO line values of a GPIO chip

Options:
  -h, --help:           display this message and exit
  -v, --version:        display the version and exit
  -l, --active-low:     set the line active state to low
  -m, --mode=[exit|wait|time|signal] (defaults to 'exit'):
               tell the program what to do after setting values
  -s, --sec=SEC:        specify the number of seconds to wait (only valid for --mode=time)
  -u, --usec=USEC:      specify the number of microseconds to wait (only valid for --
↪mode=time)
  -b, --background:     after setting values: detach from the controlling terminal

Modes:
  exit:         set values and exit immediately
  wait:         set values and wait for user to press ENTER
  time:         set values and sleep for a specified amount of time
  signal:       set values and wait for SIGINT or SIGTERM

Note: the state of a GPIO line controlled over the character device reverts to default
when the last process referencing the file descriptor representing the device file exits.
This means that it's wrong to run gpioset, have it exit and expect the line to continue
being driven high or low. It may happen if given pin is floating but it must be interpreted
as undefined behavior.
```

Warning

Some of the user IOs are used for special functions. Before using a user IO, refer to the schematic or the hardware manual of your board to ensure that it is not already in use.

7.7.1 GPIOs via sysfs

Warning

Accessing gpios via sysfs is deprecated and we encourage to use libgpiod instead.

Support to access GPIOs via sysfs is not enabled by default any more. It is only possible with manually enabling `CONFIG_GPIO_SYSFS` in the kernel configuration. To make `CONFIG_GPIO_SYSFS` visible in menuconfig the option `CONFIG_EXPERT` has to be enabled first.

You can also add this option for example to the defconfig you use in `arch/arm64/configs/` in the linux kernel sources. For our NXP based releases, this could be for example `imx8_phytec_distro.config`:

```
..
CONFIG_EXPERT=y
CONFIG_GPIO_SYSFS=y
..
```

Otherwise you can create a new config fragment. This is described in our Yocto Reference Manual.

Pinmuxing of some GPIO pins in the device tree imx8mm-phyboard-polis-rdk.dts:

```
pinctrl_leds: leds1grp {
        fsl,pins = <
                MX8MM_IOMUXC_GPIO1_IO01_GPIO1_IO1       0x16
                MX8MM_IOMUXC_GPIO1_IO14_GPIO1_IO14      0x16
                MX8MM_IOMUXC_GPIO1_IO15_GPIO1_IO15      0x16
        >;
};
```

7.8 LEDs

If any LEDs are connected to GPIOs, you have the possibility to access them by a special LED driver interface instead of the general GPIO interface (section GPIOs). You will then access them using `/sys/class/leds/` instead of `/sys/class/gpio/`. The maximum brightness of the LEDs can be read from the `max_brightness` file. The brightness file will set the brightness of the LED (taking a value from 0 up to max_brightness). Most LEDs do not have hardware brightness support and will just be turned on by all non-zero brightness settings.

Below is a simple example.

To get all available LEDs, type:

```
target:~$ ls /sys/class/leds
led-1@  led-2@  led-3@  mmc1::@  mmc2::@
```

Here the LEDs blue-mmc, green-heartbeat, and red-emmc are on the phyBOARD-Polis.

- To toggle the LEDs ON:

  ```
  target:~$ echo 255 > /sys/class/leds/led-1/brightness
  ```

- To toggle OFF:

  ```
  target:~$ echo 0 > /sys/class/leds/led-1/brightness
  ```

Device tree configuration for the User I/O configuration can be found here: https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/imx8mm-phyboard-polis-rdk.dts?h=v5.15.71_2.2.2-phy3#n36

7.9 I²C Bus

The i.MX 8M Mini contains several Multimaster fast-mode I²C modules. PHYTEC boards provide plenty of different I²C devices connected to the I²C modules of the i.MX 8M Mini. This section describes the basic device usage and its DT representation of some I²C devices integrated into our phyBOARD-Polis.

The device tree node for i2c contains settings such as clock-frequency to set the bus frequency and the pin control settings including scl-gpios and sda-gpios which are alternate pin configurations used for bus recovery.

General I²C1 bus configuration (e.g. imx8mm-phycore-som.dtsi): https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/imx8mm-phycore-som.dtsi?h=v5.15.71_2.2.2-phy3#n119

General I²C4 bus configuration (e.g. imx8mm-phyboard-polis-rdk.dts): https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/imx8mm-phyboard-polis-rdk.dts?h=v5.15.71_2.2.2-phy3#n244

7.10 EEPROM

On the phyCORE-i.MX8MM there is an i2c EEPROM flash populated. It has two addresses. The main EEPROM space (bus: I2C-0 address: 0x51) and the ID-page (bus: I2C-0 address: 0x59) can be accessed via the sysfs interface in Linux. The first 256 bytes of the main EEPROM and the ID-page are used for board detection and must not be overwritten. Overwriting reserved spaces will result in boot issues.

7.10.1 I2C EEPROM on phyCORE-i.MX8MM

Warning

The EEPROM ID page (bus: I2C-0 addr: 0x59) and the first 265 bytes of the normal EEPROM area (bus: I2C-0 addr: 0x51) should not be erased or overwritten. As this will influence the behavior of the bootloader. The board might not boot correctly anymore.

The I2C EEPROM on the phyCORE-i.MX8MM SoM is connected to I2C address 0x51 on the I2C-0 bus. It is possible to read and write directly to the device populated:

```
target:~$ hexdump -c /sys/class/i2c-dev/i2c-0/device/0-0051/eeprom
```

To read and print the first 1024 bytes of the EEPROM as a hex number, execute:

```
target:~$ dd if=/sys/class/i2c-dev/i2c-0/device/0-0051/eeprom bs=1 count=1024  | od -x
```

To fill the 4KiB EEPROM (bus: I2C-0 addr: 0x51) with zeros leaving out the EEPROM data use:

```
target:~$ dd if=/dev/zero of=/sys/class/i2c-dev/i2c-0/device/0-0051/eeprom seek=1 bs=256 count=15
```

7.10.2 EEPROM SoM Detection

The I2C EEPROM, populated on the phyCORE-i.MX8MM, has a separate ID page that is addressable over I2C address 0x59 on bus 0 and a normal area that is addressable over I2C address 0x51 on bus 0. PHYTEC uses this data area of 32 Bytes to store information about the SoM. This includes PCB revision and mounting options.

The EEPROM data is read at a really early stage during startup. It is used to select the correct RAM configuration. This makes it possible to use the same bootloader image for different RAM sizes and choose the correct DTS overlays automatically.

If the EEPROM ID page data and the first 265 bytes of the normal area are deleted, the bootloader will fall back to the phyCORE-i.MX8MM Kit RAM setup, which is 2GiB RAM.

Warning

The EEPROM ID page (bus: I2C-0 addr: 0x59) and the first 265 bytes of the normal EEPROM area (bus: I2C-0 addr: 0x51) should not be erased or overwritten. As this will influence the behavior of the bootloader. The board might not boot correctly anymore.

SoMs that are flashed with data format API revision 2 will print out information about the module in the early stage.

7.10.3 Rescue EEPROM Data

The hardware introspection data is pre-written on both EEPROM data spaces. If you have accidentally deleted or overwritten the normal area, you can copy the hardware introspection from the ID area to the normal area.

```
target:~$ dd if=/sys/class/i2c-dev/i2c-0/device/0-0059/eeprom of=/sys/class/i2c-dev/i2c-0/device/
↪0-0051/eeprom bs=1
```

Note

If you deleted both EEPROM spaces, please contact our support!

DT representation, e.g. in phyCORE-i.MX 8M Mini file imx8mm-phycore-som.dtsi can be found in our PHYTEC git: https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/imx8mm-phycore-som.dtsi?h=v5.15.71_2.2.2-phy3#n311

7.11 RTC

RTCs can be accessed via `/dev/rtc*`. Because PHYTEC boards have often more than one RTC, there might be more than one RTC device file.

- To find the name of the RTC device, you can read its sysfs entry with:

  ```
  target:~$ cat /sys/class/rtc/rtc*/name
  ```

- You will get, for example:

  ```
  rtc-rv3028 0-0052
  snvs_rtc 30370000.snvs:snvs-rtc-lp
  ```

Tip

This will list all RTCs including the non-I²C RTCs. Linux assigns RTC device IDs based on the device tree/aliases entries if present.

Date and time can be manipulated with the `hwclock` tool and the date command. To show the current date and time set on the target:

```
target:~$ date
Thu Jan  1 00:01:26 UTC 1970
```

Change the date and time with the date command. The date command sets the time with the following syntax “YYYY-MM-DD hh:mm:ss (+|-)hh:mm”:

```
target:~$ date -s "2022-03-02 11:15:00 +0100"
Wed Mar  2 10:15:00 UTC 2022
```

Note

Your timezone (in this example +0100) may vary.

Using the date command does not change the time and date of the RTC, so if we were to restart the target those changes would be discarded. To write to the RTC we need to use the `hwclock` command. Write the current date and time (set with the date command) to the RTC using the `hwclock` tool and reboot the target to check if the changes were applied to the RTC:

```
target:~$ hwclock -w
target:~$ reboot
    .
    .
    .
target:~$ date
Wed Mar  2 10:34:06 UTC 2022
```

To set the time and date from the RTC use:

```
target:~$ date
Thu Jan  1 01:00:02 UTC 1970
target:~$ hwclock -s
target:~$ date
Wed Mar  2 10:45:01 UTC 2022
```

7.11.1 RTC Wakealarm

It is possible to issue an interrupt from the RTC to wake up the system. The format uses the Unix epoch time, which is the number of seconds since UTC midnight on 1 January 1970. To wake up the system after 4 minutes from suspend to ram state, type:

```
target:~$ echo "+240" > /sys/class/rtc/rtc0/wakealarm
target:~$ echo mem > /sys/power/state
```

Note

Internally the wake alarm time will be rounded up to the next minute since the alarm function doesn't support seconds.

7.11.2 RTC Parameters

RTCs have a few abilities which can be read/set with the help of `hwclock` tool.

- We can check RTC supported features with:

  ```
  target:~$ hwclock --param-get features
  The RTC parameter 0x0 is set to 0x11.
  ```

  What this value means is encoded in kernel, each set bit translates to:

  ```
  #define RTC_FEATURE_ALARM               0
  #define RTC_FEATURE_ALARM_RES_MINUTE    1
  ```

(continues on next page)

(continued from previous page)

```
#define RTC_FEATURE_NEED_WEEK_DAY      2
#define RTC_FEATURE_ALARM_RES_2S       3
#define RTC_FEATURE_UPDATE_INTERRUPT   4
#define RTC_FEATURE_CORRECTION         5
#define RTC_FEATURE_BACKUP_SWITCH_MODE 6
#define RTC_FEATURE_CNT                7
```

- We can check RTC BSM (Backup Switchover Mode) with:

```
target:~$ hwclock --param-get bsm
The RTC parameter 0x2 is set to 0x1.
```

- We can set RTC BSM with:

```
target:~$ hwclock --param-set bsm=0x2
The RTC parameter 0x2 will be set to 0x2.
```

  What BSM values mean translates to these values:

```
#define RTC_BSM_DISABLED   0
#define RTC_BSM_DIRECT     1
#define RTC_BSM_LEVEL      2
#define RTC_BSM_STANDBY    3
```

Tip

You should set BSM mode to DSM or LSM for RTC to switch to backup power source when the initial power source is not available. Check **RV-3028** RTC datasheet to read what LSM (Level Switching Mode) and DSM (Direct Switching Mode) actually mean.

DT representation for I²C RTCs: https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/imx8mm-phycore-som.dtsi?h=v5.15.71_2.2.2-phy3#n319

7.12 USB Host Controller

The USB controller of the i.MX 8M Mini 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 (HighSpeed 'HS'). The USB subsystem has two independent USB controller cores. Both cores are On-The-Go (OTG) controller cores and are capable of acting as a USB peripheral device or a USB host. Each is connected to a USB 2.0 PHY.

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, all mass storage devices connected get unique IDs and can be found in `/dev/disk/by-id`. These IDs can be used in `/etc/fstab` to mount the different USB memory devices in different ways.

User USB2 (host) configuration is in the kernel device tree imx8mm-phyboard-polis-rdk.dts:

```
&usbotg2 {
        dr_mode = "host";
        picophy,pre-emp-curr-control = <3>;
```

(continues on next page)

(continued from previous page)

```
        picophy,dc-vol-level-adjust = <7>;
        status = "okay";
};
```

DT representation for USB Host: https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/imx8mm-phyboard-polis-rdk.dts?h=v5.15.71_2.2.2-phy3#n347

7.13 USB OTG

Most PHYTEC boards provide a USB OTG interface. USB OTG ports automatically act as a USB device or USB host. The mode depends on the USB hardware attached to the USB OTG port. If, for example, a USB mass storage device is attached to the USB OTG port, the device will show up as `/dev/sda`.

7.13.1 USB Device

In order to connect the board's USB device to a USB host port (for example a PC), you need to configure the appropriate USB gadget. With USB configfs you can define the parameters and functions of the USB gadget. The BSP includes USB configfs support as a kernel module.

```
target:~$ modprobe libcomposite
```

Example:

- First, define the parameters such as the USB vendor and product IDs, and set the information strings for the English (0x409) language:

Hint

To save time, copy these commands and execute them in a script

```
cd /sys/kernel/config/usb_gadget/
mkdir g1
cd g1/
echo "0x1d6b" > idVendor
echo "0x0104" > idProduct
mkdir strings/0x409
echo "0123456789" > strings/0x409/serialnumber
echo "Foo Inc." > strings/0x409/manufacturer
echo "Bar Gadget" > strings/0x409/product
```

- Next, create a file for the mass storage gadget:

```
target:~$ dd if=/dev/zero of=/tmp/file.img bs=1M count=64
```

- Now you should create the functions you want to use:

```
cd /sys/kernel/config/usb_gadget/g1
mkdir functions/acm.GS0
mkdir functions/ecm.usb0
mkdir functions/mass_storage.0
echo /tmp/file.img > functions/mass_storage.0/lun.0/file
```

- *acm*: Serial gadget, creates serial interface like `/dev/ttyGS0`.
- *ecm*: Ethernet gadget, creates ethernet interface, e.g. usb0
- *mass_storage*: The host can partition, format, and mount the gadget mass storage the same way as any other USB mass storage.
- Bind the defined functions to a configuration:

```
cd /sys/kernel/config/usb_gadget/g1
mkdir configs/c.1
mkdir configs/c.1/strings/0x409
echo "CDC ACM+ECM+MS" > configs/c.1/strings/0x409/configuration
ln -s functions/acm.GS0 configs/c.1/
ln -s functions/ecm.usb0 configs/c.1/
ln -s functions/mass_storage.0 configs/c.1/
```

- Finally, start the USB gadget with the following commands:

```
target:~$ cd /sys/kernel/config/usb_gadget/g1
target:~$ ls /sys/class/udc/
ci_hdrc.0
target:~$ echo "ci_hdrc.0" >UDC
```

If your system has more than one USB Device or OTG port, you can pass the right one to the USB Device Controller (UDC).

- To stop the USB gadget and unbind the used functions, execute:

```
target:~$ echo "" > /sys/kernel/config/usb_gadget/g1/UDC
```

Both USB interfaces are configured as host in the kernel device tree imx8mm-phyboard-polis-rdk.dts. See: https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/imx8mm-phyboard-polis-rdk.dts?h=v5.15.71_2.2.2-phy3#n335

7.14 CAN FD

The phyBOARD-Polis two flexCAN interfaces supporting CAN FD. They are supported by the Linux standard CAN framework which builds upon then the Linux network layer. Using this framework, the CAN interfaces behave like an ordinary Linux network device, with some additional features special to CAN. More information can be found in the Linux Kernel documentation: https://www.kernel.org/doc/html/latest/networking/can.html

Hint

phyBOARD-Polis has an external CANFD chip MCP2518FD connected over SPI. There are different interfaces involved, which limits the datarate capabilities of CANFD.

Hint

On phyBOARD-Polis-i.MX8MM a terminating resistor can be enabled by setting S5 to ON if required.

- Use:

```
target:~$ ip link
```

to see the state of the interfaces. The two CAN interfaces should show up as can0 and can1.

- To get information on can0, such as bit rate and error counters, type:

```
target:~$ ip -d -s link show can0
```

The information for can0 will look like:

```
2: can0: <NOARP,UP,LOWER_UP,ECHO> mtu 16 qdisc pfifo_fast state UNKNOWN mode DEFAULT group␣
↪default qlen 10
    link/can  promiscuity 0 minmtu 0 maxmtu 0
    can state ERROR-ACTIVE (berr-counter tx 0 rx 0) restart-ms 0
          bitrate 500000 sample-point 0.875
          tq 50 prop-seg 17 phase-seg1 17 phase-seg2 5 sjw 1
          mcp25xxfd: tseg1 2..256 tseg2 1..128 sjw 1..128 brp 1..256 brp-inc 1
          mcp25xxfd: dtseg1 1..32 dtseg2 1..16 dsjw 1..16 dbrp 1..256 dbrp-inc 1
          clock 20000000
          re-started bus-errors arbit-lost error-warn error-pass bus-off
          0          0          0          0          0          0         numtxqueues 1␣
↪numrxqueues 1 gso_max_size 65536 gso_max_segs 65535
    RX: bytes  packets  errors  dropped overrun mcast
    0          0        0       0       0       0
    TX: bytes  packets  errors  dropped carrier collsns
    0          0        0       0       0       0
```

The output contains a standard set of 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:

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

The CAN configuration is done in the systemd configuration file `/lib/systemd/network/can0.network`. For a persistent change of (as an example, the default bitrates), change the configuration in the BSP under `./meta-ampliphy/recipes-core/systemd/systemd-conf/can0.network` in the root filesystem and rebuild the root filesystem.

```
[Match]
Name=can0

[Can]
BitRate=500000
```

The bitrate can also be changed manually, for example, to make use of the flexible bitrate:

```
target:~$ ip link set can0 down
target:~$ ip link set can0 txqueuelen 10 up type can bitrate 500000 sample-point 0.75 dbitrate␣
↪4000000 dsample-point 0.8 fd on
```

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

```
target:~$ cansend can0 123#45.67
target:~$ candump can0
```

To generate random CAN traffic for testing purposes, use cangen:

```
target:~$ cangen
```

`cansend --help` and `candump --help` provide help messages for further information on options and usage.

Warning

The mcp2518fd SPI to CANfd supports only baudrates starting from 125kB/s. Slower rates can be selected but may not work correctly.

Device Tree CAN configuration of imx8mm-phyboard-polis-rdk.dts: https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/imx8mm-phyboard-polis-rdk.dts?h=v5.15.71_2.2.2-phy3#n175

7.15 PCIe

The phyCORE-i.MX 8M Mini has one Mini-PCIe slot. In general, 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 PCIe devices recognized.

- Type:

  ```
  target:~$ lspci -v
  ```

- You will receive:

  ```
  00:00.0 PCI bridge: Synopsys, Inc. Device abcd (rev 01) (prog-if 00 [Normal decode])
          Flags: bus master, fast devsel, latency 0, IRQ 218
          Memory at 18000000 (64-bit, non-prefetchable) [size=1M]
          Bus: primary=00, secondary=01, subordinate=ff, sec-latency=0
          I/O behind bridge: None
          Memory behind bridge: 18100000-181fffff [size=1M]
          Prefetchable memory behind bridge: None
          [virtual] Expansion ROM at 18200000 [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: [148] L1 PM Substates
          Kernel driver in use: dwc3-haps

  01:00.0 Network controller: Intel Corporation WiFi Link 5100
          Subsystem: Intel Corporation WiFi Link 5100 AGN
  ```

(continues on next page)

(continued from previous page)

```
        Flags: fast devsel
        Memory at 18100000 (64-bit, non-prefetchable) [disabled] [size=8K]
        Capabilities: [c8] Power Management version 3
        Capabilities: [d0] MSI: Enable- Count=1/1 Maskable- 64bit+
        Capabilities: [e0] Express Endpoint, MSI 00
        Capabilities: [100] Advanced Error Reporting
        Capabilities: [140] Device Serial Number 00-24-d6-ff-ff-84-0d-1e
        Kernel modules: iwlwifi
```

In this example, the PCIe device is the *Intel Corporation WiFi Link 5100*.

For PCIe devices, you have to enable the correct driver in the kernel configuration. This WLAN card, for example, is manufactured by Intel. The option for the driver, which must be enabled, is named `CONFIG_IWLWIFI` and can be found under *Intel Wireless WiFi Next Gen AGN - Wireless-N/Advanced-N/Ultimat* in the kernel configuration.

- In order to activate the driver, follow the instructions from our Yocto manual: Kernel and Bootloader Configuration
  - The linux-imx is represented by: **virtual/kernel**

For some devices like the WLAN card, additional binary firmware blobs are needed. These firmware blobs have to be placed in `/lib/firmware/` before the device can be used.

- Type:

```
host:~$ scp -r <firmware> root@192.168.3.11:/lib/firmware
```

- For example, if you try to bring up the network interface:

```
target:~$ ip link set up wlp1s0
```

- You will get the following output on the serial console:

```
[   58.682104] iwlwifi 0000:01:00.0: L1 Disabled - LTR Disabled
[   58.690822] iwlwifi 0000:01:00.0: L1 Disabled - LTR Disabled
[   58.696577] iwlwifi 0000:01:00.0: Radio type=0x1-0x2-0x0
[   58.831022] iwlwifi 0000:01:00.0: L1 Disabled - LTR Disabled
[   58.839679] iwlwifi 0000:01:00.0: L1 Disabled - LTR Disabled
[   58.845435] iwlwifi 0000:01:00.0: Radio type=0x1-0x2-0x0
[   58.902797] IPv6: ADDRCONF(NETDEV_UP): wlp1s0: link is not ready
```

Tip

Some PCIe devices, e.g. the Ethernet card, may function properly even if no firmware blob is loaded from `/lib/firmware/` and you received an error message as shown in the first line of the output above. This is because some manufacturers provide the firmware as a fallback on the card itself. In this case, the behavior and output depend strongly on the manufacturer's firmware.

7.16 Audio

The PEB-AV-10-Connector exists in two versions and the 1531.1 version is populated with a TI TLV320AIC3007 audio codec. Audio support is done via the I2S interface and controlled via I2C.

There is a 3.5mm headset jack with OMTP standard and an 8-pin header to connect audio devices with the AV-Connector. The 8-pin header contains a mono speaker, headphones, and line-in signals.

To check if your soundcard driver is loaded correctly and what the device is called, type for playback devices:

```
target:~$ aplay -L
```

Or type for recording devices:

```
target:~$ arecord -L
```

7.16.1 Alsamixer

To inspect the capabilities of your soundcard, call:

```
target:~$ alsamixer
```

You should see a lot of options as the audio-IC has many features you can experiment with. It might be better to open alsamixer via ssh instead of the serial console, as the console graphical effects are better. You have either mono or stereo gain controls for all mix points. "MM" means the feature is muted (both output, left & right), which can be toggled by hitting '**m**'. You can also toggle by hitting '<' for left and '>' for right output. With the **tab** key, you can switch between controls for playback and recording.

7.16.2 Restore default volumes

There are some default settings stored in `/var/lib/alsa/asound.state`. You can save your current alsa settings with:

```
target:~$ alsactl --file </path/to/filename> store
```

You can restore saved alsa settings with:

```
target:~$ alsactl --file </path/to/filename> restore
```

7.16.3 ALSA configuration

Our BSP comes with a ALSA configuration file `/etc/asound.conf`.

The ALSA configuration file can be edited as desired or deleted since it is not required for ALSA to work properly.

```
target:~$ vi /etc/asound.conf
```

To set PEB-AV-10 as output, set *playback.pcm* from "dummy" to "pebav10":

```
[...]

pcm.asymed {
        type asym
        playback.pcm "pebav10"
        capture.pcm "dsnoop"
}

[...]
```

If the sound is not audible change playback devices to the software volume control playback devices, set *playback.pcm* to the respective softvol playback device e.g. “softvol_pebav10”. Use alsamixer controls to vary the volume levels.

```
[...]

pcm.asymed {
        type asym
        playback.pcm "softvol_pebav10"
        capture.pcm "dsnoop"
}

[...]
```

7.16.4 Pulseaudio Configuration

For applications using *Pulseaudio*, check for available sinks:

```
target:~$ pactl list short sinks
```

To select the output device, type:

```
target:~$ pactl set-default-sink <sink_number>
```

7.16.5 Playback

Run speaker-test to check playback availability:

```
target:~$ speaker-test -c 2 -t wav
```

To playback simple audio streams, you can use aplay. For example to play the ALSA test sounds:

```
target:~$ aplay /usr/share/sounds/alsa/*
```

To playback other formats like mp3 for example, you can use Gstreamer:

```
target:~$ gst-launch-1.0 playbin uri=file:/path/to/file.mp3
```

7.16.6 Capture

`arecord` is a command-line tool for capturing audio streams which use Line In as the default input source. To select a different audio source you can use `alsamixer`. For example, switch on *Right PGA Mixer Mic3R* and *Left PGA Mixer Mic3R* in order to capture the audio from the microphone input of the TLV320-Codec using the 3.5mm jack.

```
target:~$ amixer -c "sndpebav10" sset 'Left PGA Mixer Mic3R' on
target:~$ amixer -c "sndpebav10" sset 'Right PGA Mixer Mic3R' on
```

```
target:~$ arecord -t wav -c 2 -r 44100 -f S16_LE test.wav
```

Hint

Since playback and capture share hardware interfaces, it is not possible to use different sampling rates and formats for simultaneous playback and capture operations.

Device Tree Audio configuration: https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/overlays/imx8mm-phyboard-polis-peb-av-010.dtsi?h=v5.15.71_2.2.2-phy3#n54

7.17 Video

7.17.1 Videos with Gstreamer

One example video is installed by default in the BSP at */usr/share/qtphy/videos/*. Start the video playback with one of these commands:

```
target:~$ gst-launch-1.0 playbin uri=file:///usr/share/qtphy/videos/caminandes_3_llamigos_720p_
↪vp9.webm
```

- Or:

```
target:~$ gst-launch-1.0 -v filesrc location=/usr/share/qtphy/videos/caminandes_3_llamigos_720p_
↪vp9.webm ! decodebin name=decoder decoder. ! videoconvert ! waylandsink
```

- Or:

```
target:~$ gst-play-1.0 /usr/share/qtphy/videos/caminandes_3_llamigos_720p_vp9.webm --videosink␣
↪waylandsink
```

7.18 Display

The 10” Display is always active. If the PEB-AV-Connector is not connected, an error message may occur at boot.

7.18.1 Qt Demo

With the `phytec-qt6demo-image`, Weston starts during boot. Our Qt6 demo application named “qtphy” can be stopped with:

```
target:~$ systemctl stop qtphy
```

- To start the demo again, run:

  ```
  target:~$ systemctl start qtphy
  ```

- To disable autostart of the demo, run:

  ```
  target:~$ systemctl disable qtphy
  ```

- To enable autostart of the demo, run:

  ```
  target:~$ systemctl enable qtphy
  ```

- Weston can be stopped with:

```
target:~$ systemctl stop weston
```

Note

The Qt demo must be closed before Weston can be closed.

7.18.2 Backlight Control

If a display is connected to the PHYTEC board, you can control its backlight with the Linux kernel sysfs interface. All available backlight devices in the system can be found in the folder /sys/class/backlight. Reading the appropriate files and writing to them allows you to control the backlight.

Note

Some boards with multiple display connectors might have multiple backlight controls in /sys/class/backlight. For example: backlight0 and backlight1

- To get, for example, the maximum brightness level (max_brightness) execute:

  ```
  target:~$ cat /sys/class/backlight/backlight/max_brightness
  ```

  Valid brightness values are 0 to <max_brightness>.

- To obtain the current brightness level, type:

  ```
  target:~$ cat /sys/class/backlight/backlight/brightness
  ```

- Write to the file brightness to change the brightness:

  ```
  target:~$ echo 0 > /sys/class/backlight/backlight/brightness
  ```

  turns the backlight off for example.

  For documentation of all files, see https://www.kernel.org/doc/Documentation/ABI/stable/sysfs-class-backlight.

The device tree of PEB-AV-10 can be found here: https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/overlays/imx8mm-phyboard-polis-peb-av-010.dtsi?h=v5.15.71_2.2.2-phy3

7.19 Power Management

7.19.1 CPU Core Frequency Scaling

The CPU in the i.MX 8M Mini 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 to as 'Dynamic Voltage and Frequency Scaling' (DVFS). The i.MX 8M Mini BSP supports the DVFS feature. The Linux kernel provides a DVFS framework that allows each CPU core to have a min/max frequency and a governor that governs it. Depending on the i.MX 8 variant used, several different frequencies are supported.

Tip

Although the DVFS framework provides frequency settings for each CPU core, a change in the frequency settings of one CPU core always affects all other CPU cores too. So all CPU cores always share the same DVFS setting. An individual DVFS setting for each core is not possible.

- To get a complete list type:

```
target:~$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies
```

  In case you have, for example, i.MX 8MPlus CPU with a maximum of approximately 1,6 GHz, the result will be:

```
1200000 1600000
```

- To ask for the current frequency type:

```
target:~$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
```

So-called governors are automatically selecting one of these frequencies in accordance with their goals.

- List all governors available with the following command:

```
target:~$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_governors
```

  The result will be:

```
conservative ondemand userspace powersave performance schedutil
```

- **conservative** is much like the ondemand governor. It differs in behavior in that it gracefully increases and decreases the CPU speed rather than jumping to max speed the moment there is any load on the CPU.
- **ondemand** (default) 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.
- **powersave** always selects the lowest possible CPU core frequency.
- **performance** always selects the highest possible CPU core frequency.
- **userspace** allows the user or userspace program running as root to set a specific frequency (e.g. to 1600000). Type:
- In order to ask for the current governor, type:

```
target:~$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

  You will normally get:

```
ondemand
```

- Switching over to another governor (e.g. userspace) is done with:

```
target:~$ echo userspace > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

- Now you can set the speed:

```
target:~$ echo 1600000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
```

For more detailed information about the governors, refer to the Linux kernel documentation in the linux kernel repository at `Documentation/admin-guide/pm/cpufreq.rst`.

7.19.2 CPU Core Management

The i.MX 8M Mini SoC can have multiple processor cores on the die. The i.MX 8M Mini, for example, has 4 ARM Cores which can be turned on and off individually at runtime.

- To see all available cores in the system, execute:

  ```
  target:~$ ls /sys/devices/system/cpu  -1
  ```

- This will show, for example:

  ```
  cpu0    cpu1   cpu2   cpu3   cpufreq
  [...]
  ```

  Here the system has four processor cores. By default, all available cores in the system are enabled to get maximum performance.

- To switch off a single-core, execute:

  ```
  target:~$ echo 0 > /sys/devices/system/cpu/cpu3/online
  ```

  As confirmation, you will see:

  ```
  [  110.505012] psci: CPU3 killed
  ```

  Now the core is powered down and no more processes are scheduled on this core.

- You can use top to see a graphical overview of the cores and processes:

  ```
  target:~$ htop
  ```

- To power up the core again, execute:

  ```
  target:~$ echo 1 > /sys/devices/system/cpu/cpu3/online
  ```

7.19.3 Suspend to RAM

The phyCORE-i.MX8MM supports basic suspend and resume. Different wake-up sources can be used. Suspend/resume is possible with:

```
target:~$ echo mem > /sys/power/state
#resume with pressing on/off button
```

To wake up with serial console run

```
target:~$ echo enabled > /sys/class/tty/ttymxc2/power/wakeup
target:~$ echo mem > /sys/power/state
```

7.20 Thermal Management

7.20.1 U-Boot

The previous temperature control in the U-Boot was not satisfactory. Now the u-boot has a temperature shutdown to prevent the board from getting too hot during booting. The temperatures at which the shutdown occurs are identical to those in the kernel.

The individual temperature ranges with the current temperature are displayed in the boot log:

```
CPU:   Industrial temperature grade (-40C to 105C) at 33C
```

7.20.2 Kernel

The Linux kernel has integrated thermal management that is capable of monitoring SoC temperatures, reducing the CPU frequency, driving fans, advising other drivers to reduce the power consumption of devices, and – worst-case – shutting down the system gracefully (https://www.kernel.org/doc/Documentation/thermal/sysfs-api.txt).

This section describes how the thermal management kernel API is used for the i.MX 8M Mini SoC platform. The i.MX 8 has internal temperature sensors for the SoC.

- The current temperature can be read in millicelsius with:

  ```
  target:~$ cat /sys/class/thermal/thermal_zone0/temp
  ```

- You will get, for example:

  ```
  49000
  ```

There are two trip points registered by the imx_thermal kernel driver. These differ depending on the CPU variant. A distinction is made between Industrial and Commercial.

	Commercial	Industrial
passive (warning)	85°C	95°C
critical (shutdown)	90°C	100°C

(see kernel sysfs folder `/sys/class/thermal/thermal_zone0/`)

The kernel thermal management uses these trip points to trigger events and change the cooling behavior. The following thermal policies (also named thermal governors) are available in the kernel: Step Wise, Fair Share, Bang Bang, and Userspace. The default policy used in the BSP is step_wise. If the value of the SoC temperature in the sysfs file temp is above *trip_point_0*, the CPU frequency is set to the lowest CPU frequency. When the SoC temperature drops below *trip_point_0* again, the throttling is released.

Note

The actual values of the thermal trip points may differ since we mount CPUs with different temperature grades.

7.20.3 GPIO Fan

Note

Only GPIO fan supported.

A GPIO fan can be connected to the phyBOARD-Polis-i.MX 8M Mini. The SoC only contains one temperature sensor which is already used by the thermal frequency scaling. The fan can not be controlled by the kernel. We use lmsensors with hwmon for this instead. lmsensors reads the temperature periodically and enables or disables the fan at a configurable threshold. For the phyBOARD-Polis-i.MX 8M Mini, this is 60°C.

The settings can be configured in the configuration file:

```
/etc/fancontrol
```

Fan control is started by a systemd service during boot. This can be disabled with:

```
target:~$ systemctl disable fancontrol
```

7.21 Watchdog

The PHYTEC i.MX 8M Mini modules include a hardware watchdog that is able to reset the board when the system hangs. The watchdog is started on default in U-Boot with a timeout of 60s. So even during early kernel start, the watchdog is already up and running. The Linux kernel driver takes control over the watchdog and makes sure that it is fed. This section explains how to configure the watchdog in Linux using systemd to check for system hangs and during reboot.

7.21.1 Watchdog Support in systemd

Systemd has included hardware watchdog support since version 183.

- To activate watchdog support, the file system.conf in `/etc/systemd/` has to be adapted by enabling the options:

  ```
  RuntimeWatchdogSec=60s
  ShutdownWatchdogSec=10min
  ```

RuntimeWatchdogSec defines the timeout value of the watchdog, while *ShutdownWatchdogSec* defines the timeout when the system is rebooted. For more detailed information about hardware watchdogs under systemd can be found at http://0pointer.de/blog/projects/watchdog.html. The changes will take effect after a reboot or run:

```
target:~$  systemctl daemon-reload
```

7.22 On-Chip OTP Controller (OCOTP_CTRL) - eFuses

The i.MX 8M Mini provides one-time programmable fuses to store information such as the MAC address, boot configuration, and other permanent settings ("On-Chip OTP Controller (OCOTP_CTRL)" in the i.MX 8M Mini Reference Manual). The following list is an abstract from the i.MX 8M Mini Reference Manual and includes some useful registers in the OCOTP_CTRL (at base address 0x30350000):

Name	Bank	Word	Memory offset at 0x30350000	Description
OCOTP_MAC_	9	0	0x640	contains lower 32 bits of ENET0 MAC address
OCOTP_MAC_	9	1	0x650	contains upper 16 bits of ENET0 MAC address and the lower 16 bits of ENET1 MAC address
OCOTP_MAC_	9	2	0x660	contains upper 32 bits of ENET1 MAC address

A complete list and a detailed mapping between the fuses in the OCOTP_CTRL and the boot/mac/... configuration are available in the section "Fuse Map" of the i.MX 8M Mini Security Reference Manual.

7.22.1 Reading Fuse Values in uBoot

You can read the content of a fuse using memory-mapped shadow registers. To calculate the memory address, use the fuse Bank and Word in the following formula:

OCOTP_MAC_ADDR:

```
u-boot=> fuse read 9 0
```

7.22.2 Reading Fuse Values in Linux

To access the content of the fuses in Linux NXP provides the NVMEM_IMX_OCOTP module. All fuse content of the memory-mapped shadow registers is accessible via sysfs:

```
target:~$ hexdump /sys/devices/platform/soc@0/30000000.bus/30350000.efuse/imx-ocotp0/nvmem
```

CHAPTER

EIGHT

I.MX 8M MINI M4 CORE

In addition to the Cortex-A53 cores, there is a Cortex-M4 Core as MCU integrated into the i.MX 8M Mini SoC. Our Yocto-Linux-BSP runs on the A53-Cores and the M4 Core can be used as a secondary core for additional tasks using bare-metal or RTOS firmware. Both cores have access to the same peripherals and thus peripheral usage needs to be limited either in the M4 Core's firmware or the devicetree for the Linux operating system. This section describes how to build firmware examples and how to run them on phyBOARD-Polis.

The phyBOARD-Polis is currently supported by the NXP MCUXpresso SDK and by The Zephyr Project. This section only describes the NXP MCUXpresso SDK because the MCUXpresso SDK has more supported features at the moment.

If you want to use the M4 Core with The Zephyr Project, please refer to the The Zephyr Project documentation:

- https://docs.zephyrproject.org/latest/boards/phytec/mimx8mm_phyboard_polis/doc/index.html

8.1 Getting the Firmware Examples

The firmware can be built using the NXP MCUxpresso SDK with a compatible compiler toolchain using command-line tools.

8.1.1 Getting the Sources

The MCUX SDK and the examples for the i.MX 8M Mini can be obtained from PHYTEC's GitHub page:

- https://github.com/phytec/mcux-sdk/
- https://github.com/phytec/mcux-sdk-phytec-examples/

1. Initialize the MCUX SDK via west:

   ```
   host:~$ west init -m https://github.com/phytec/mcux-sdk/ --mr <VERSION> mcuxsdk
   ```

   This will create a mcuxsdk directory with the mcux-sdk repository in it. The `mcux-sdk-phytec-examples` repository will be automatically cloned into the mcuxsdk directory. The given argument <VERSION> is a the branch name of the mcux-sdk repository that represents the MCUX SDK version. For the newest tested version you can use 2.13.0.

   Note

   `west` is a repository management tool and a part of the Zephyr Project. To install west, you can use pip. In this example west is installed in a python virtual environment:

```
host:~$ sudo apt install python3-venv python3-pip
host:~$ python3 -m venv west-env
host:~$ source west-env/bin/activate
(west-env) host:~$ pip3 install west
```

2. Update the dependencies:

```
host:~$ cd mcuxsdk
host:~/mcuxsdk$ west update
```

The directory `examples-phytec` contains all examples ported and tested for phyBOARD-Polis with version 2.13.0 of MCUX.

To build the firmware, a compiler toolchain and make/cmake are required. The GNU Arm Embedded Toolchain may be available in your distribution's repositories, e.g. for Ubuntu.

```
host:~$ sudo apt install gcc-arm-none-eabi binutils-arm-none-eabi make cmake
```

The compiler toolchain can also be obtained directly from https://developer.arm.com/. After the archive has been extracted, the `ARMGCC_DIR` has to be added to the environment, e.g. for the ARM toolchain 10-2020-q4-major release located in the home directory:

```
host:~$ export ARMGCC_DIR=~/gcc-arm-none-eabi-10-2020-q4-major
```

8.1.2 Building the Firmware

To build the PHYTEC samples an environment has to be sourced

```
host:~/mcuxsdk$ source scripts/setenv
```

The scripts to build the firmware are located in <sdk-directory>/phytec-mcux-boards/phyboard-pollux/<example_category>/<example>/armgcc. There are scripts for each memory location the firmware is supposed to run in, e.g.

```
host:~$ ./build_release.sh
```

to build the firmware for the M4 Core's TCM. The output will be placed under release/ in the armgcc directory. .bin files and can be run in U-Boot and .elf files within Linux.

To build the firmware for the DRAM, run the script build_ddr_release. The script of the firmware that is supposed to run, e.g.

```
host:~$ ./build_ddr_release.sh
```

The output will be placed under ddr_release/ in the armgcc directory. .bin files and can be run in U-Boot and .elf files within Linux.

8.2 Running M4 Core Examples

There are two ways to run the M4 Core with the built firmware, U-Boot and Remoteproc within a running Linux.

To receive debug messages start your favorite terminal software (e.g. Minicom, Tio, or Tera Term) on your host PC and configure it for 115200 baud, 8 data bits, no parity, and 1 stop bit (8n1) with no handshake.

Once a micro-USB cable is connected to the USB-debug port on the phyBOARD-Polis, two ttyUSB devices are registered. One prints messages from A53-Core's debug UART and the other one from the M4 Core's debug UART.

8.2.1 Running Examples from U-Boot

To load firmware using the bootloader U-Boot, the bootaux command can be used:

1. Prepare an SD Card with our Yocto-BSP
2. Copy the generated .bin file to the SD Cards first partition
3. Stop the autoboot by pressing any key
4. Type the command depending on the type of firmware:

For firmware built to run in the M4 Core's TCM:

```
u-boot=> fatload mmc 1:1 0x48000000 firmware.bin;cp.b 0x48000000 0x7e0000 20000;
u-boot=> bootaux 0x7e0000
## Starting auxiliary core stack = 0x20020000, pc = 0x000004CD...
```

For firmware built to run in the DRAM:

```
u-boot=> fatload mmc 1:1 0x80000000 firmware.bin
u-boot=> dcache flush
u-boot=> bootaux 0x80000000
## Starting auxiliary core stack = 0x80400000, pc = 0x80000539...
```

The program's output should appear on the M4 Core's debug UART.

8.2.2 Running Examples from Linux using Remoteproc

Remoteproc is a module that allows you to control the M4 Core from Linux during runtime. Firmware built for TCM can be loaded and the execution started or stopped. To use Remoteproc a devicetree overlay needs to be set:

Edit the bootenv.txt file located in the /boot directory on the target by adding imx8mm-phycore-rpmsg.dtbo:

```
overlays=imx8mm-phycore-rpmsg.dtbo
```

Restart the target and execute in U-Boot:

```
u-boot=> run prepare_mcore
```

Firmware .elf files for the M4 Core can be found under /lib/firmware. To load the firmware, type:

```
target:~$ echo /lib/firmware/<firmware>.elf > /sys/class/remoteproc/remoteproc0/firmware
target:~$ echo start > /sys/class/remoteproc/remoteproc0/state
```

To load a different firmware, the M4 Core needs to be stopped:

```
target:~$ echo stop > /sys/class/remoteproc/remoteproc0/state
```

Note

The samples found in `/lib/firmware` on the target come from NXP's Yocto layer meta-imx. To use the samples you built yourself through MCUX SDK, please copy them to `/lib/firmware` on the target after building them.

8.2.3 Debugging Using J-Link

The Segger software can be obtained from https://www.segger.com/downloads/jlink/. As of version V7.20a of the Segger software, accessing the i.MX 8M Mini' M4 Core requires additional configuration files to be copied into the J-Link software directory: NXP J-Link files for i.MX 8M Mini

Together with the J-Link, GDB Server can be used for running and debugging the software. On the phyBOARD-Polis, the JTAG-Pins are accessible via the X8 Expansion Connector. The simplest way is to use a PEB-EVAL-01 board that has the JTAG-Pins reachable with a pin header on the top.

```
host:~$ sudo apt install gdb gdb-multiarch
```

To start the J-Link software, type:

```
host:~$ JLinkGDBServer -if JTAG -device MIMX8MM6_M4
...
Connected to target
Waiting for GDB connection...
```

To start GDB with a firmware example in another window, type:

```
host:~$ gdb-multiarch firmware.elf
...
(gdb) target remote localhost:2331
(gdb) monitor reset
Resetting target
(gdb) load
...
(gdb) monitor go
```

CHAPTER

NINE

BSP EXTENSIONS

9.1 Chromium

Our BSP for the phyBOARD-Polis-i.MX 8M Mini supports Chromium. You can include it in the phytec-qt6demo-image with only a few steps.

9.1.1 Adding Chromium to Your local.conf

To include Chromium you have to add the following line into your local.conf. You can find it in <yocto_dir>/build/conf/local.conf. This adds Chromium to your next image build.

```
IMAGE_INSTALL:append = " chromium-ozone-wayland"
```

Note

Compiling Chromium takes a long time.

9.1.2 Get Chromium Running on the Target

To run Chromium, it needs a few arguments to use the hardware graphics acceleration:

```
target$ chromium --use-gl=desktop --enable-features=VaapiVideoDecoder --no-sandbox
```

If you want to start Chromium via SSH, you must first define the display on which Chromium will run. For example:

```
target$ DISPLAY=:0
```

After you have defined this, you can start it via virtual terminal on Weston as shown above.