phyFLEX-i.MX 95 FPSC BSP Manual ALPHA2

PHYTEC Messtechnik GmbH

Apr 02, 2026

CONTENTS

1 Supported Hardware **3**
1.1 phyFLEX Libra RDK Components . . . 3

2 Getting Started **5**
2.1 Get the Image . . . 5
2.2 Write the Image to SD Card . . . 5
2.3 First Start-up . . . 8

3 Building the BSP **9**
3.1 Basic Set-Up . . . 9
3.2 Get the BSP . . . 9

4 Installing the OS **13**
4.1 Bootmode Switch (S1) . . . 13
4.2 Flash e.MMC . . . 13

5 Development **19**
5.1 Standalone Build preparation . . . 19
5.2 U-Boot standalone build . . . 20
5.3 Kernel standalone build . . . 22
5.4 Working with UUU . . . 23
5.5 Host Network Preparation . . . 25
5.6 Booting the Kernel from a Network . . . 28
5.7 U-Boot PXE Boot . . . 29
5.8 Accessing the Development states . . . 30
5.9 Format SD card . . . 30
5.10 Ampliphy-boot . . . 36
5.11 Working with FIT images . . . 38

6 Device Tree (DT) **41**
6.1 Introduction . . . 41
6.2 PHYTEC i.MX 95 BSP Device Tree Concept . . . 41

7 Accessing Peripherals **45**
7.1 i.MX 95 Pin Muxing . . . 45
7.2 Ethernet . . . 46
7.3 WLAN/Bluetooth . . . 48
7.4 SD card . . . 50
7.5 e.MMC Devices . . . 51
7.6 GPIOs . . . 58
7.7 I²C Bus . . . 60

7.8 LEDs . . . 61
7.9 CAN FD . . . 61
7.10 RS232/RS485 . . . 63
7.11 EEPROM . . . 65
7.12 RTC . . . 66
7.13 USB Host Controller . . . 68
7.14 Video . . . 70
7.15 Display . . . 70
7.16 Power Management . . . 72
7.17 Thermal Management . . . 74
7.18 PWM fan . . . 75
7.19 TPM . . . 76
7.20 GPU . . . 77
7.21 Watchdog . . . 77
7.22 JTAG . . . 78

8 NXP GoPoint demo suite **81**
8.1 ML Benchmark . . . 81

The table below shows the Compatible BSPs for this manual:

Compatible BSPs	BSP Release Type	Yocto Version	BSP Release Date	BSP Status
BSP-Yocto-NXP-i.MX95-ALPHA2	Alpha	Walnascar	2026/03/06	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 **phyFLEX-i.MX 95 Libra Rapid Development 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 its products. This includes any or all of the following:

QS Guide
A short guide on how to set up and boot a phyCORE based board.

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/phyflex-imx-95-fpsc/ of our product.

CHAPTER

ONE

SUPPORTED HARDWARE

On our web page, you can see all supported Machines with the available Article Numbers for this release: BSP-Yocto-NXP-i.MX95-ALPHA2 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 phyFLEX Libra RDK Components

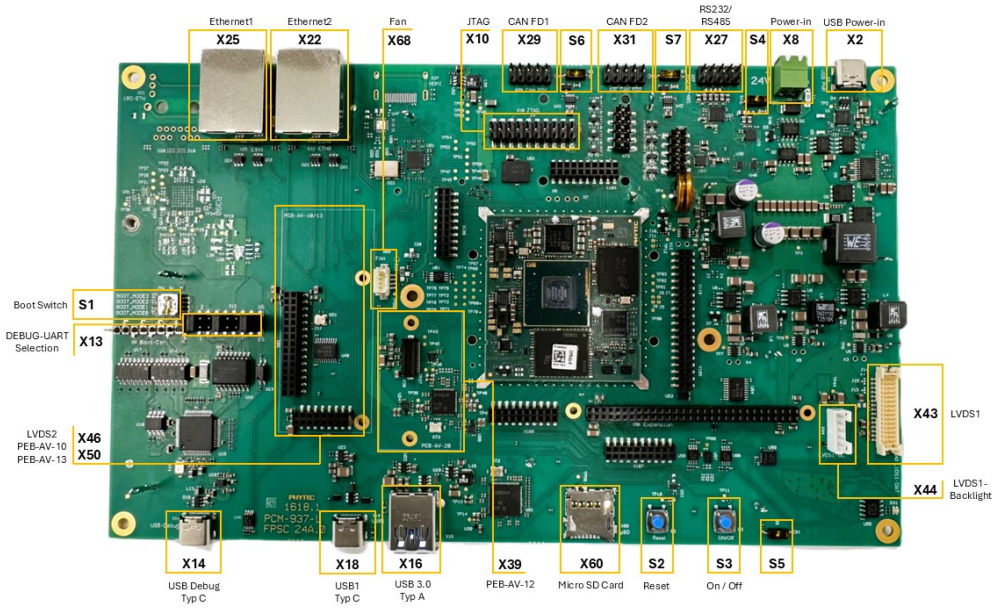


Fig. 1: **Libra FPSC Components (top)**

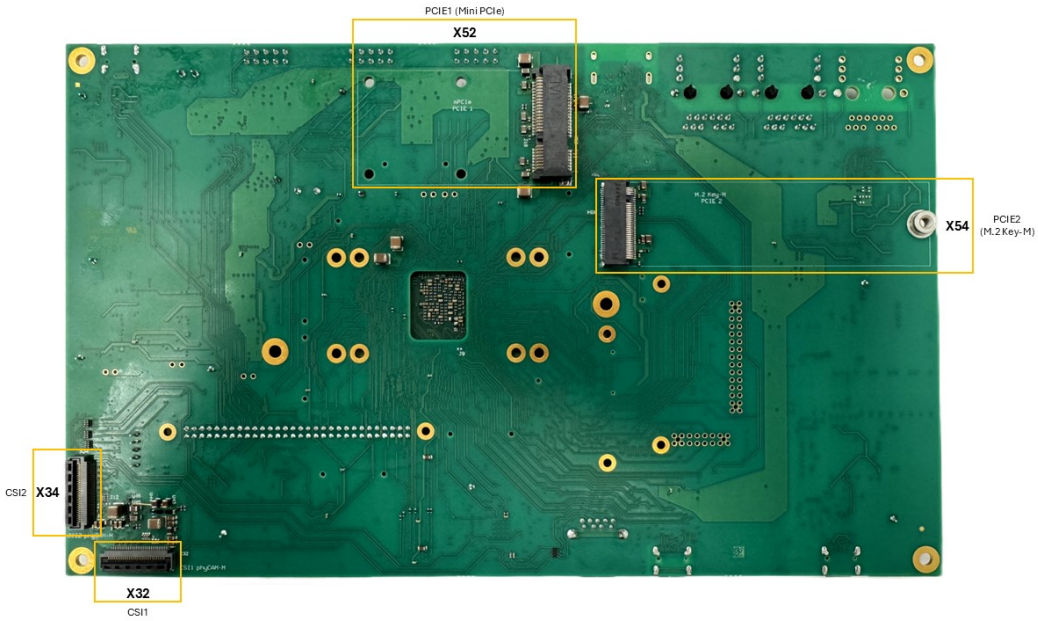


Fig. 2: **Libra FPSC Components (bottom)**

CHAPTER

TWO

GETTING STARTED

The **phyFLEX-i.MX 95 Libra Rapid Development 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 e.MMC is programmed with only a U-Boot by default. You can get all sources from the BSP downloads page. 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 e.MMC. 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 our BSP downloads page.

Note that you can find different image versions and variants on our download server. The images are located on the server by folders per "BSP-Version", "Distro-Name" and "Machine-Name".

Example to download a partup package and a WIC image from the download server:

```
host:~$ wget none
host:~$ wget https://download.phytec.de/Software/Linux/BSP-Yocto-i.MX95/BSP-Yocto-NXP-i.MX95-
↪ALPHA2/images/ampliphy-vendor/imx95-phyflex-libra-rdk-2/phytec-qt6demo-image-imx95-phyflex-
↪libra-rdk-2.rootfs.wic.xz
```

Note

For e.MMC, 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 `bmaptool`.

2.2.2 Using bmaptool

One way to prepare an SD card is using bmaptool. 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-imx95-phyflex-libra-rdk-2?(.rootfs).wic?(.xz) /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.

2.2.3 Using partup

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

```
host:~$ sudo partup install phytec-qt6demo-image-imx95-phyflex-libra-rdk-2?(.rootfs).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.

Warning

Host systems which are using resize2fs version 1.46.6 and older (e.g. Ubuntu 22.04) are not able to write partup packages created with Yocto Mickledore or newer to SD-Card. This is due to a new default option in resize2fs which causes an incompatibility. See release notes.

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 in the previous chapter.

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.

2.2.4 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-imx95-phyflex-libra-rdk-2?(.rootfs).wic of=/dev/<your_
↪device> bs=1M conv=fsync status=progress
```

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

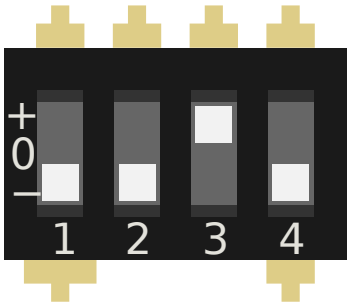
```
host:~$ xzcat phytec-qt6demo-image-imx95-phyflex-libra-rdk-2?(.rootfs).wic.xz | sudo dd of=/dev/
↪<your_device> bs=1M conv=fsync status=progress
```

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, the *bootmode switch (S1)* needs to be set to the following position:



- Insert the SD card
- Connect the target and the host with **USB-C** on *(X14)* 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 95 BSP using Yocto and the phyLinux script. For more information about our meta-layer or Yocto in general visit: Yocto Reference Manual (walnascar).

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 (walnascar).

3.2 Get the BSP

There are two ways to get the BSP sources. You can download the complete BSP sources from our BSP downloads page; 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 sources 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 (MACHINE) 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 the webpage of our BSP.

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

```
host:~/yocto$ DISTRO=ampliphy-vendor MACHINE=imx95-phyflex-libra-rdk-2 ./phyLinux init -p␣
↪imx95 -r BSP-Yocto-NXP-i.MX95-ALPHA2
```

After the execution of the init command, phyLinux will print a few important notes. For example, it will print your git identity, 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/.
- 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 imx95-phyflex-libra-rdk-2 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)

- **imx-boot-tools/bl31-imx95.bin**: ARM Trusted Firmware binary
- **imx-boot-tools/lpddr5_dmem_qb_v202409.bin, imx-boot-tools/lpddr5_dmem_v202409.bin, imx-boot-tools/lpddr5_imem_qb_v202409.bin, imx-boot-tools/lpddr5_imem_v202409.bin**: DDR PHY firmware images
- **imx-boot-tools/oei-m33-ddr.bin, oei-m33-tcm.bin**: OEI images
- **imx-boot-tools/m33_image-mx95libra.bin**: System Manager image
- **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.
- **fitImage**: Linux kernel FIT image
- **fitImage.its**
- **Image**: Linux kernel image
- **Image.config**: Kernel configuration
- **imx95-phyflex-libra-rdk*.dtb**: Kernel device tree file
- **imx95-phyflex-fpsc*.dtbo, imx95-phyflex-libra-rdk*.dtbo**: Kernel device tree overlay files
- **phytec-qt6demo-image*.tar.gz**: Root file system
- **phytec-qt6demo-image*.rootfs.wic.xz**: compressed SD card image

CHAPTER

FOUR

INSTALLING THE OS

4.1 Bootmode Switch (S1)

The phyFLEX Libra RDK features a boot switch with four individually switchable ports to select the phyFLEX-i.MX 95 FPSC default bootsource.

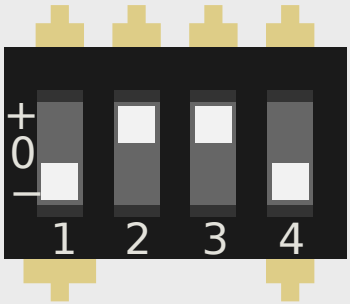


Fig. 1: USB serial download

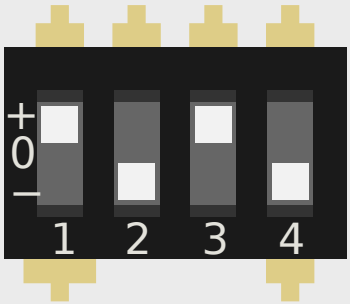


Fig. 2: e.MMC

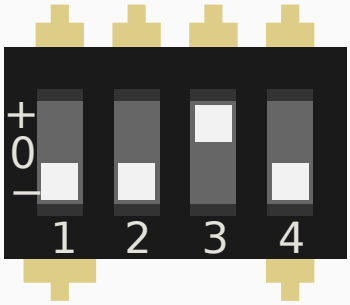


Fig. 3: SD card

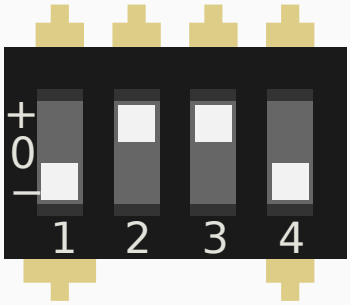


Fig. 4: SPI NOR flash

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

4.2 Flash e.MMC

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*.

4.2.1 Flash e.MMC from Network

i.MX 95 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 e.MMC 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 e.MMC. The image contains the bootloader, kernel, device tree, device tree overlays, and root file system.

Flash e.MMC via Network in Linux on Host

It is also possible to install the OS at e.MMC 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-imx95-phyflex-libra-rdk-2.rootfs.wic.xz
phytec-qt6demo-image-imx95-phyflex-libra-rdk-2.rootfs.wic.bmap
```

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

```
host:~$ scp /srv/tftp/phytec-qt6demo-image-imx95-phyflex-libra-rdk-2.rootfs.wic.* root@192.168.3.
↪11:/tmp && ssh root@192.168.3.11 "bmaptool copy /tmp/phytec-qt6demo-image-imx95-phyflex-libra-
↪rdk-2.rootfs.wic.xz /dev/mmcblk2"
```

Flash e.MMC via Network in Linux on Target

You can update the e.MMC from your target.

Tip

A working network is necessary! *Setup Network Host*

Take a compressed or decompressed image with the accompanying block map file **.bmap* on the host and send it with *ssh* through the network to the e.MMC of the target with a one-line command:

```
target:~$ scp <USER>@192.168.3.10:/srv/tftp/phytec-qt6demo-image-imx95-phyflex-libra-rdk-2.
↪rootfs.wic.* /tmp && bmaptool copy /tmp/phytec-qt6demo-image-imx95-phyflex-libra-rdk-2.rootfs.
↪wic.xz /dev/mmcblk2
```

Flash e.MMC from Network in U-Boot on Target

These steps will show how to update the e.MMC 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 OP-TEE.

Tip

A working network is necessary! *Setup Network Host*

Uncompress your image

```
host:~$ unxz /srv/tftp/phytec-headless-image-imx95-phyflex-libra-rdk-2.rootfs.wic.xz
```

Load your image via network to RAM:

- when using dhcp

```
u-boot=> dhcp phytec-headless-image-imx95-phyflex-libra-rdk-2.rootfs.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-headless-image-imx95-phyflex-libra-rdk-2.rootfs.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-headless-image-imx95-phyflex-libra-rdk-2.rootfs.wic
Using ethernet@30be0000 device
TFTP from server 192.168.3.10; our IP address is 192.168.3.11
Filename 'phytec-headless-image-imx95-phyflex-libra-rdk-2.rootfs.wic'.
Load address: 0x40480000
Loading: #####################################
         #####################################
         #####################################
         ...
         ...
         ...
         #####################################
         #############
         11.2 MiB/s
done
Bytes transferred = 911842304 (36599c00 hex)
```

Write the image to the e.MMC:

```
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
```

4.2.2 Flash e.MMC 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 e.MMC is located in the e.MMC user area.

Tip

A working network is necessary! *Setup Network Host*

Load image over tftp into RAM and then write it to e.MMC:

```
u-boot=> tftp ${loadaddr} imx-boot
u-boot=> setexpr nblk ${filesize} / 0x200
u-boot=> mmc dev 2
u-boot=> mmc write ${loadaddr} 0x40 ${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 e.MMC from USB stick

Flash e.MMC from USB in Linux

These steps will show how to flash the e.MMC 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-imx95-phyflex-libra-rdk-2.|yocto-imageext|). 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-imx95-phyflex-libra-rdk-2.rootfs.wic.xz
phytec-qt6demo-image-imx95-phyflex-libra-rdk-2.rootfs.wic.bmap
```

- Write the image to the phyFLEX-i.MX 95 FPSC e.MMC (MMC device 2 without partition):

```
target:~$ bmaptool copy /mnt/phytec-qt6demo-image-imx95-phyflex-libra-rdk-2.rootfs.wic.xz /
↪dev/mmcblk2
```

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

Tip

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

4.2.4 Flash e.MMC from SD card

Even if there is no network available, you can update the e.MMC. 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 need to allocate more SD card space. 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 e.MMC from SD card in Linux on Target

You can also flash the e.MMC 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-imx95-phyflex-libra-rdk-2.rootfs.partup
phytec-qt6demo-image-imx95-phyflex-libra-rdk-2.rootfs.wic.xz
phytec-qt6demo-image-imx95-phyflex-libra-rdk-2.rootfs.wic.bmap
```

- Write the image to the phyFLEX-i.MX 95 FPSC e.MMC (MMC device 2 **without** partition) using partup:

```
target:~$ partup install phytec-qt6demo-image-imx95-phyflex-libra-rdk-2.rootfs.partup /dev/
↪mmcblk2
```

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

Note

Alternatively, `bmaptool` may be used instead:

```
target:~$ bmaptool copy phytec-qt6demo-image-imx95-phyflex-libra-rdk-2.rootfs.wic.xz /
↪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 e.MMC.

> **Warning**
>
> Before this will work, you need to configure the *bootmode switch (S1)* to e.MMC.

Flash e.MMC 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 e.MMC 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.rootfs.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-headless-image-imx95-phyflex-libra-rdk-2.
↪rootfs.wic
reading
911842304 bytes read in 39253 ms (22.2 MiB/s)
```

- Switch the mmc dev to e.MMC:

```
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.rootfs.wic) from the SD card to e.MMC. This will partition the card and copy imx-boot, Image, dtb, dtbo, and root file system to e.MMC.

```
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 e.MMC.

CHAPTER

FIVE

DEVELOPMENT

5.1 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 GitHub.

5.1.1 Git Repositories

- Used U-Boot repository:

```
https://github.com/phytec/u-boot-phytec-imx
```

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

```
https://github.com/phytec/linux-phytec-imx
```

- Our i.MX 95 kernel is based on the linux-phytec-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/recipes-kernel/linux/linux-phytec-imx_*.bb
meta-phytec/recipes-bsp/u-boot/u-boot-phytec-imx_*.bb
```

5.1.2 Build the SDK

You can build the SDK 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
```

5.1.3 Install the SDK

- Set correct permissions and install the SDK:

```
host:~$ chmod +x phytec-ampliphy-vendor-glibc-x86_64-phytec-qt6demo-image-cortexa55-crypto-
↪toolchain-5.2.4.sh
host:~$ ./phytec-ampliphy-vendor-glibc-x86_64-phytec-qt6demo-image-cortexa55-crypto-
↪toolchain-5.2.4.sh
```

(continues on next page)

(continued from previous page)

```
==========================================================================================================================
Enter target directory for SDK (default: /opt/ampliphy-vendor/5.2.4):
You are about to install the SDK to "/opt/ampliphy-vendor/5.2.4". Proceed [Y/n]? Y
Extracting SDK...done
Setting it up...done
SDK has been successfully set up and is ready to be used.
```

5.1.4 Using the SDK

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

```
host:~$ source /opt/ampliphy-vendor/5.2.4/environment-setup-cortexa55-crypto-phytec-linux
```

5.1.5 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
```

5.2 U-Boot standalone build

5.2.1 Get the source code

- Get the U-Boot sources:

  ```
  host:~$ git clone https://github.com/phytec/u-boot-phytec-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 v2025.04_2.1.0-phy8
- Check out the needed *U-Boot* **tag**:

  ```
  host:~$ cd ~/u-boot-phytec-imx/
  host:~/u-boot-phytec-imx$ git fetch --all --tags
  host:~/u-boot-phytec-imx$ git checkout tags/v2025.04_2.1.0-phy8
  ```

- Set up a build environment:

  ```
  host:~/u-boot-phytec-imx$ source /opt/ampliphy-vendor/5.2.4/environment-setup-cortexa55-
  ↪crypto-phytec-linux
  ```

5.2.2 Build the bootloader

- build flash.bin (imx-boot):

  ```
  host:~/u-boot-phytec-imx$ make imx95-phyflex-libra-rdk_defconfig
  host:~/u-boot-phytec-imx$ make flash.bin
  ```

5.2.3 Build the imx-boot binary with imx-mkimage

`xxd` is required for compiling the boot container.

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

Clone the repository

First clone the imx-mkimage git repository from NXP.

```
host:~$ git clone git@github.com:nxp-imx/imx-mkimage.git
```

Get the needed binaries

Then get the needed binaries and copy them to the imx-mkimage/iMX95 folder.

- **DDR firmware files** (*mkimage tool* compatible format **lpddr5_[i,d]mem_*.bin**): lpddr5_dmem_qb_*.bin, lpddr5_dmem_*.bin, lpddr5_imem_qb_*.bin, lpddr5_imem_*.bin
- **ARM Trusted firmware binary** (*mkimage tool* compatible format **bl31.bin**): bl31.bin
- **OPTEE image**: tee.bin
- **NXP Systemmanager**: m33_image.bin
- **OEI-DDR and OEI-TCM**: oei-m33-ddr.bin, oei-m33-tcm.bin
- **AHAB container image**: mx95a0-ahab-container.img

If you already built our BSP with Yocto, you can get the binaries from the directory mentioned here: *BSP Images*

Also copy the **U-Boot** (u-boot.bin) and **U-Boot SPL** (u-boot-spl.bin) binaries from your U-Boot folder. The SPL binary is located in the spl subfolder.

Build the flash.bin binary

Go to the imx-mkimage folder and execute:

```
host:~/imx-mkimage$ make SOC=iMX95 OEI=YES flash_lpboot_sm_a55
```

The flash.bin can be found in the iMX95 subfolder.

5.2.4 Flash the bootloader to a block device

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

SoC	Offset User Area	Offset Boot Partition	eMMC Device
i.MX 95	32 kiB	0 kiB	/dev/mmcblk0

E.g. flash SD card:

```
host:~/u-boot-phytec-imx$ sudo dd if=flash.bin of=/dev/sd[x] bs=1024 seek=32 conv=fsync
```

Hint

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

5.3 Kernel standalone build

The kernel is packaged in a FIT image together with the device tree. U-Boot has been adapted to be able to load a FIT image and boot the kernel contained in it. As a result, the kernel Image has to packaged in a FIT image.

5.3.1 Setup sources

- The used linux-phytec-imx branch can be found in the release notes
- The tag needed for this release is called v6.12.34-2.1.0-phy9
- Check out the needed linux-phytec-imx tag:

```
host:~$ git clone https://github.com/phytec/linux-phytec-imx
host:~$ cd ~/linux-phytec-imx/
host:~/linux-phytec-imx$ git fetch --all --tags
host:~/linux-phytec-imx$ git checkout tags/v6.12.34-2.1.0-phy9
```

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

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

- Set up a build environment:

```
host:~/linux-phytec-imx$ source /opt/ampliphy-vendor/5.2.4/environment-setup-cortexa55-
↪crypto-phytec-linux
```

5.3.2 Build the kernel

- Build the linux kernel:

```
host:~/linux-phytec-imx$ make imx9_phytec_defconfig
host:~/linux-phytec-imx$ make -j$(nproc)
```

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

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

- The Image can be found at ~/linux-phytec-imx/arch/arm64/boot/Image.gz
- The dtb can be found at ~/linux-phytec-imx/arch/arm64/boot/dts/freescale/imx95-phyflex-libra-rdk.dtb
- For (re-)building only Devicetrees and -overlays, it is sufficient to run

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

or, to build a specific dtb (e.g. imx95-phyflex-libra-rdk.dtb):

```
host:~/linux-phytec-imx$ make freescale/imx95-phyflex-libra-rdk.dtb
```

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
```

5.3.3 Package the kernel in a FIT image

To simply replace the kernel, you will need an `image tree source` (.its) file. If you already built our BSP with Yocto, you can get the its file from the directory mentioned here: *BSP Images* Or you can download the file here: https://download.phytec.de/Software/Linux/BSP-Yocto-i.MX95/BSP-Yocto-NXP-i.MX95-ALPHA2/images/ampliphy-vendor/imx95-phyflex-libra-rdk-2/

Copy the .its file to the current working directory, create a link to the kernel image and create the final fitImage with mkimage.

```
host:~/linux-phytec-imx$ cp /path/to/yocto/deploydir/fitimage-its*.its .
                  && ln -s arch/arm64/boot/Image.gz linux.bin
                  && mkimage -f fitImage-its*.its fitImage
```

5.3.4 Copy FIT image and kernel modules to SD card

When one-time boot via netboot is not sufficient, the FIT image along with the kernel modules may be copied directly to a mounted SD card.

```
host:~/linux-phytec-imx$ cp fitImage /path/to/sdcard/boot/
host:~/linux-phytec-imx$ make INSTALL_MOD_PATH=/path/to/sdcard/root/ modules_install
```

5.4 Working with UUU

The Universal Update Utility (UUU) by NXP is software to execute on the host for loading and running 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.

5.4.1 Host preparations for UUU 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
```

5.4.2 Get Images

Download imx-boot from our server or get it from your Yocto build directory at build/deploy-ampliphy-vendor/images/imx95-phyflex-libra-rdk-2/. For flashing a wic image to e.MMC, you will also need phytec-qt6demo-image-imx95-phyflex-libra-rdk-2.rootfs.wic.

5.4.3 Prepare Target

Set the *bootmode switch (S1)* to **USB Serial Download**. Also, connect USB port *X18 (upper connector)* to your host.

5.4.4 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 *(X14)*, as usual.

Note

The default boot command when booting with UUU 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 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.

5.4.5 Flashing U-boot Image to e.MMC via UUU

Warning

UUU flashes U-boot into e.MMC BOOT (hardware) boot partitions, and it sets the BOOT_PARTITION_ENABLE in the e.MMC! This is a problem since we want the bootloader to reside in the e.MMC 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 0 0 0 0
u-boot=> mmc partconf 0
EXT_CSD[179], PARTITION_CONFIG:
BOOT_ACK: 0x0
BOOT_PARTITION_ENABLE: 0x0
PARTITION_ACCESS: 0x0
```

or check Disable booting from e.MMC boot partitions from Linux.

This way the bootloader is still flashed to e.MMC BOOT partitions but it is not used!

When using **partup** tool and `.partup` package for e.MMC 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
```

5.4.6 Flashing wic Image to e.MMC via UUU

Execute and power up the board:

```
host:~$ sudo uuu -b emmc_all imx-boot phytec-qt6demo-image-imx95-phyflex-libra-rdk-2.rootfs.wic
```

5.4.7 Flashing U-Boot to SD Card via UUU

Execute and power up the board:

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

5.4.8 Flashing wic Image to SD Card via UUU

Execute and power up the board:

```
host:~$ sudo uuu -b sd_all imx-boot phytec-qt6demo-image-imx95-phyflex-libra-rdk-2.rootfs.wic
```

5.5 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
```

5.5.1 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
```

- Temporarily export the nfs directory: The NFS server is not restricted to a certain file system location, so all we have to do is to export our root file system to the embedded network. In this example, 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:

```
host:~$ sudo exportfs -i -o rw,no_root_squash,sync,no_subtree_check 192.168.3.0/
↪255.255.255.0:/srv/nfs
```

- unexport the rootfs when finished:

```
host:~$ sudo exportfs -u 192.168.3.0/255.255.255.0:/srv/nfs
```

Permanent export

- To make the export persistent across reboots on most distributions, modify the `/etc/exports` file and export it:

```
/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" }
        ]
    }]
  }
}
```

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.

Note

DHCP server setup is only needed when using dynamic IP addresses. For our vendor BSPs, static IP addresses are used by default.

```
u-boot=> env print ip_dyn
ip_dyn=no
```

To use dynamic IP addresses for netboot, `ip_dyn` needs to be set to `yes`.

5.6 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.

5.6.1 Place Images on Host for Netboot

- Copy the kernel fitimage to your tftp directory:

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

- Copy the bootscript to your tftp directory:

```
host:~$ cp net_boot_fit.scr.uimg /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-imx95-phyflex-libra-rdk-2.rootfs.tar.gz -C /srv/
↪nfs
```

Note

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

5.6.2 Network Settings on Target

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

5.6.3 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=> setenv boot_targets ethernet
u-boot=> bootflow scan -lb
```

To persistently boot from network, save the environment after setting boot_targets to ethernet.

```
u-boot=> env set boot_targets "ethernet"
u-boot=> env save
```

To use DHCP for booting from a network:

```
u-boot=> env set ip_dyn true
u-boot=> env save
```

When not using DHCP (ip_dyn set to false), U-Boot needs ipaddr and serverip to be set. If addresses different from the PHYTEC provided defaults are desired, they can be set in the U-Boot environment:

```
u-boot=> env set ipaddr <xxx.xxx.xxx.xxx>
u-boot=> env set serverip <xxx.xxx.xxx.xxx>
u-boot=> env save
```

5.7 U-Boot PXE Boot

U-Boot PXE boot allows network-based booting of kernels, with their kernel command line parameters stored on the server. Also, it allows a choice of different kernels, depending on what is present on the server. This makes it ideal for kernel development: Debugging, driver development, device tree changes (including overlays).

```
u-boot=> pxe get
u-boot=> pxe boot
```

In the example below, `C0A8030B` is the Syslinux configuration file which will be loaded by U-boot. The name is the ip address PHYTEC sets for its boards in hex. If the board gets a different ip address, this file will be ignored. A catchall is `default`, which is the last filename to be tried. For more information, see the official PXELINUX documentation.

```
├── fdts
│   ├── nightly-imx93-phyflex-libra-rdk.dtb
│   ├── nightly-imx95-phyflex-libra-rdk.dtb
│   └── nightly-imx95-phyflex-libra-rdk-lvds-ph128800t006-zhc01.dtbo
├── kernels
│   └── nightly-image
└── pxelinux.cfg
    └── C0A8030B
```

The contents of `C0A8030B` may look like this:

```
menu title Linux selections
timeout 300

label netboot-imx95-phyflex
    menu label Netboot target for Libra-phyFLEX i.MX 95
    kernel kernels/nightly-image
    fdt fdts/nightly-imx95-phyflex-libra-rdk.dtb
    fdtoverlays fdts/nightly-imx95-phyflex-libra-rdk-lvds-ph128800t006-zhc01.dtbo
    append console=ttyLP3,115200 rw root=/dev/nfs ip=dhcp nfsroot=/srv/nfs,vers=4,tcp

label netboot-imx93-phyflex
    menu label Netboot target for Libra-phyFLEX i.MX 93
    kernel kernels/nightly-image
    fdt fdts/nightly-imx93-phyflex-libra-rdk.dtb
    append console=ttyLP3,115200 rw root=/dev/nfs ip=dhcp nfsroot=/srv/nfs,vers=4,tcp

default netboot-imx95-phyflex
```

Note

The device tree directives are specific to U-Boot. See the PXE Boot and extlinux.conf section in the

U-Boot documentation on which directives U-Boot recognizes.

Hint

The default FIT image does not work with pxeboot. This is due to a conflicting entry and load address for the kernel. The FIT image will we loaded to `kernel_addr_r` where the kernel will then be extracted to, leading to an error (since the FIT image already resides at this address).

Kernel nfs info: https://www.kernel.org/doc/html/latest/admin-guide/nfs/nfsroot.html

5.8 Accessing the Development states

5.8.1 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 imx95 -r BSP-Yocto-NXP-i.MX95-PD26.1.y
```

This will initialize a BSP that will track the latest development state of the current release (BSP-Yocto-NXP-i.MX95-ALPHA2). From now on *repo sync* in this folder will pull all the latest changes from our Git repositories:

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

5.8.2 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.MX95-ALPHA2) and `.y` at the end:

```
host:~$ ./phyLinux init -p imx95
```

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:

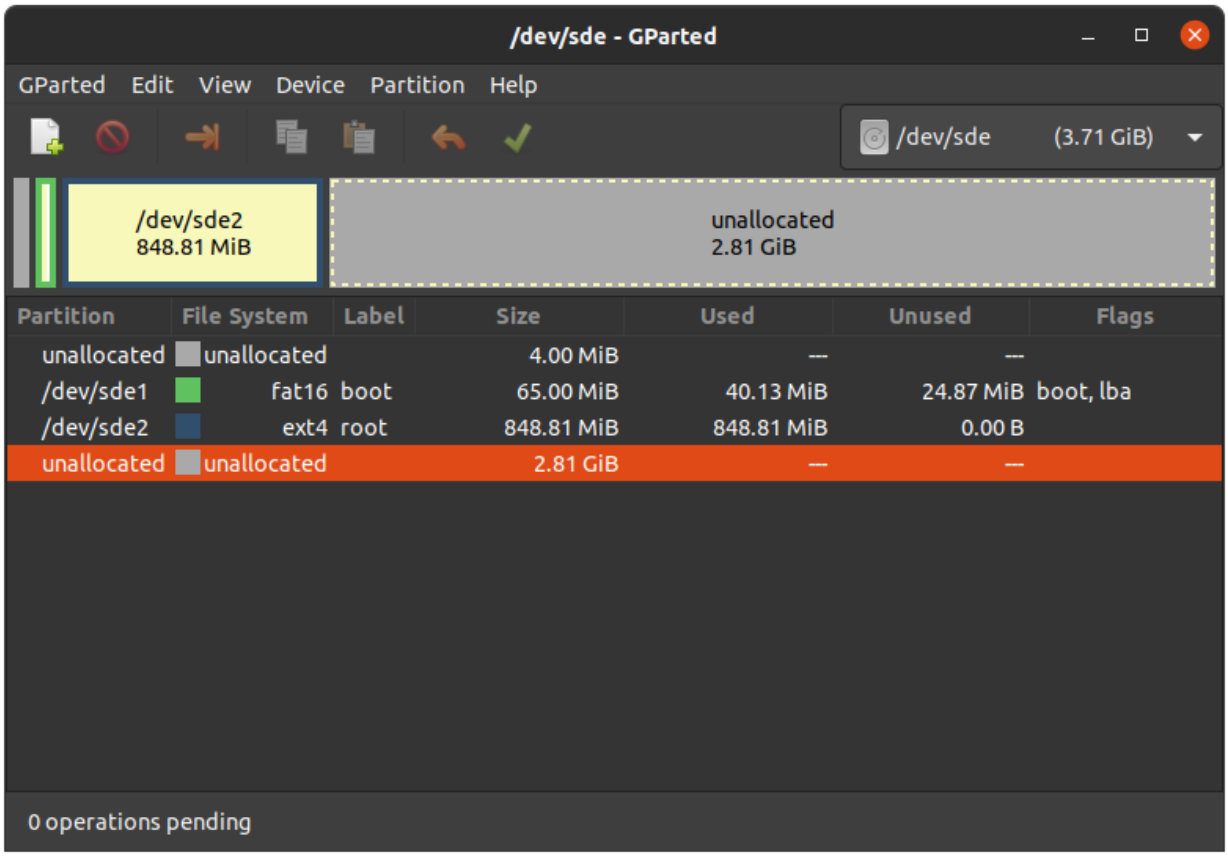
  ```
  host:~$ sudo apt install gparted
  ```

- 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
```

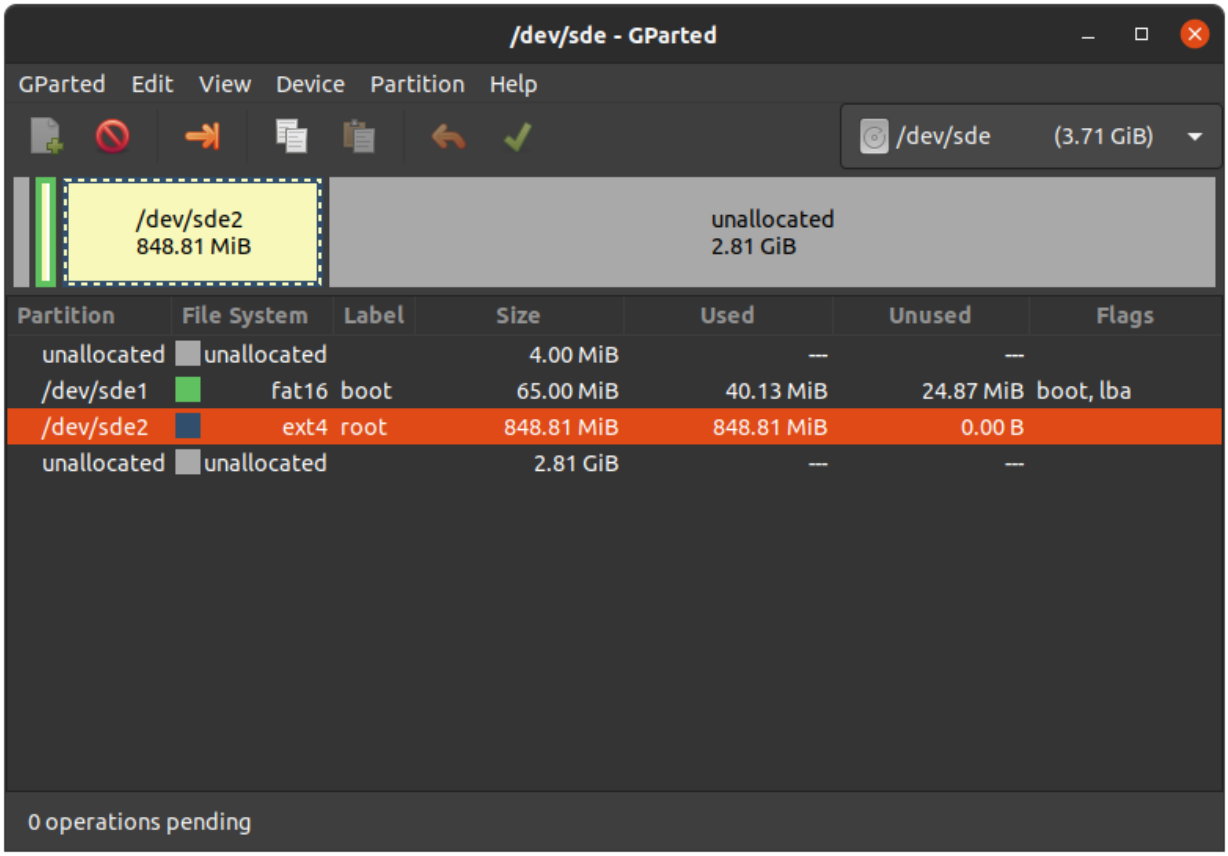


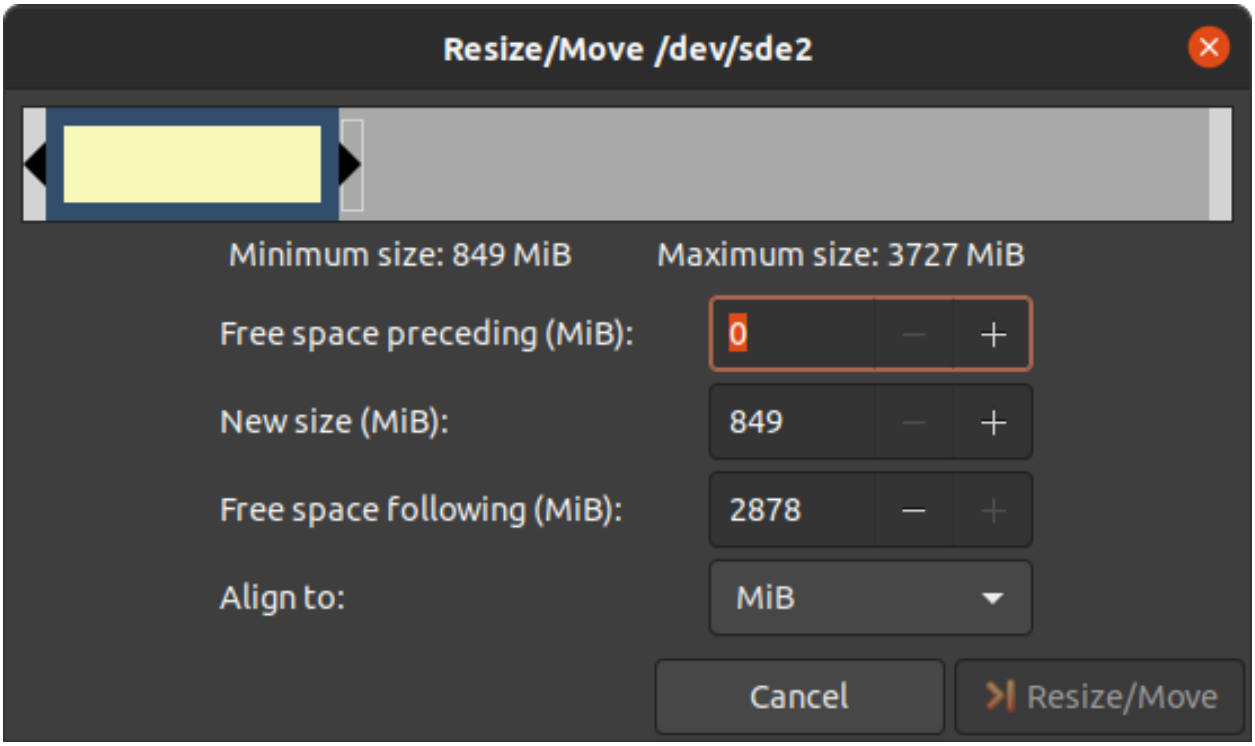
Expand rootfs

Warning

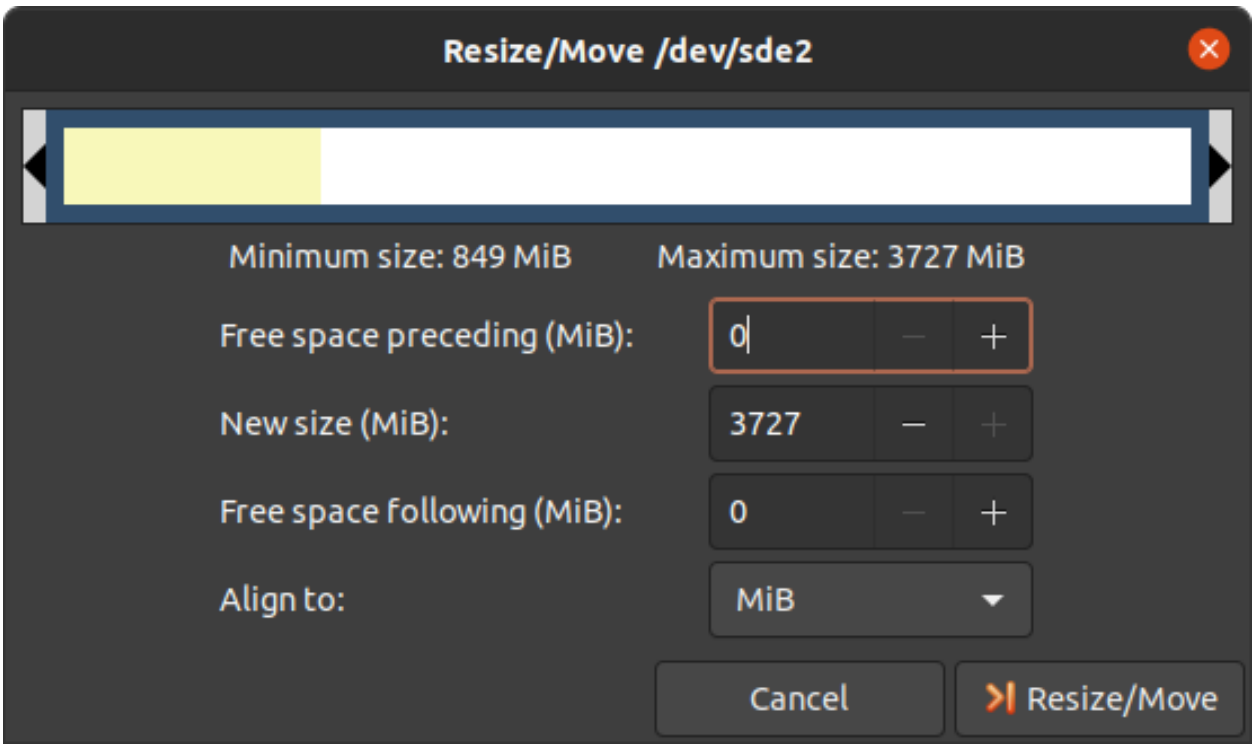
Running gparted on host systems which are using resize2fs version 1.46.6 and older (e.g. Ubuntu 22.04) are not able to expand the ext4 partition created with Yocto Mickledore and newer. This is due to a new default option in resize2fs which causes a incompatibility. See release notes.

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

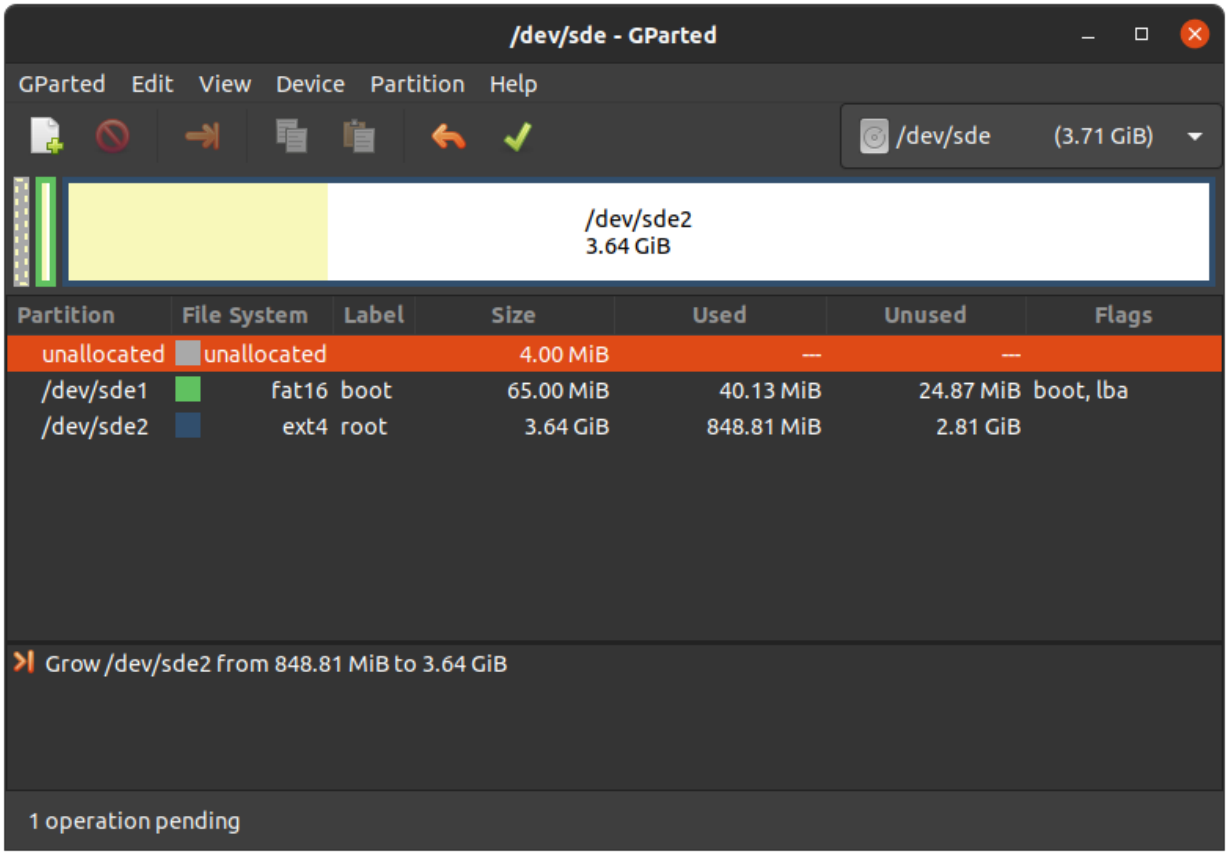




- Drag the slider as far as you like or enter the size manually.



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



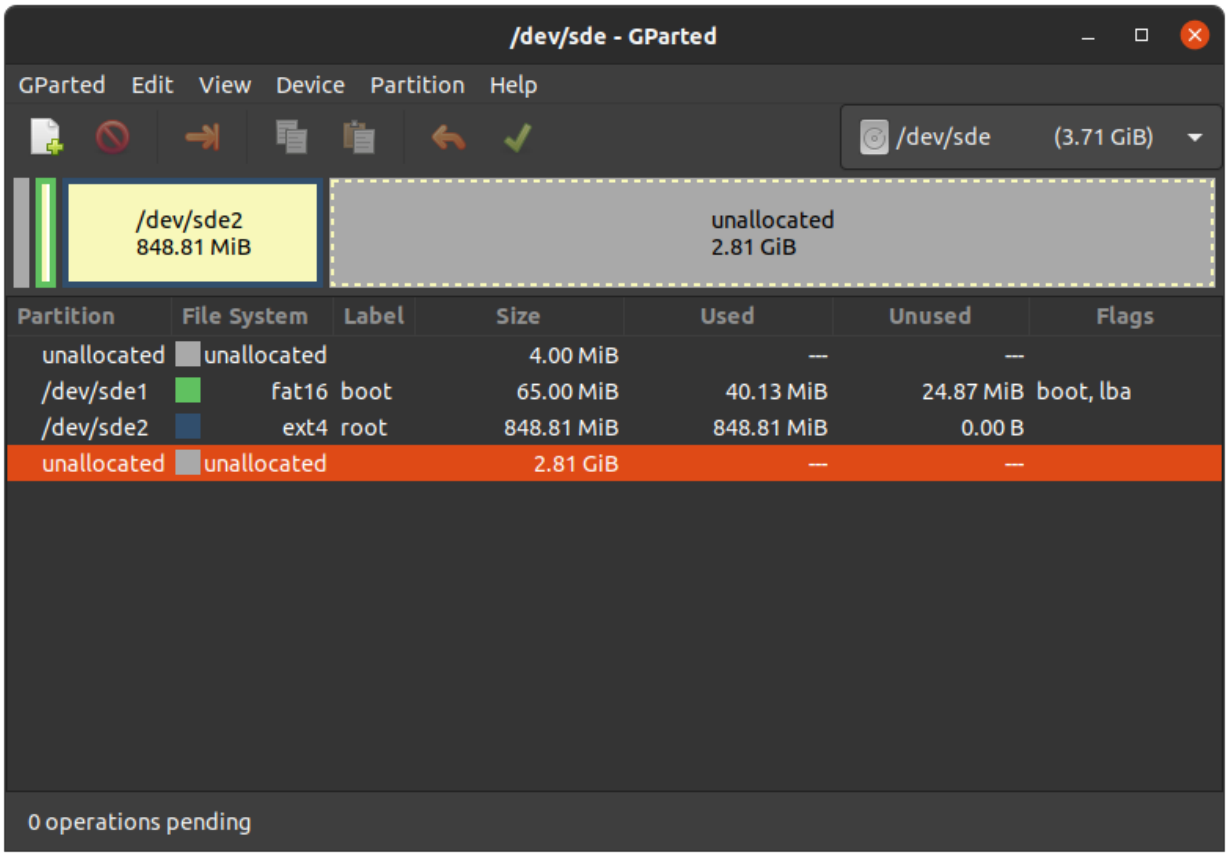
- To apply your changes, press the green tick.

- Now you can mount the root partition and copy e.g. the phytec-qt6demo-image-imx95-phyflex-libra-rdk-2.wic image to it. Then unmount it again:

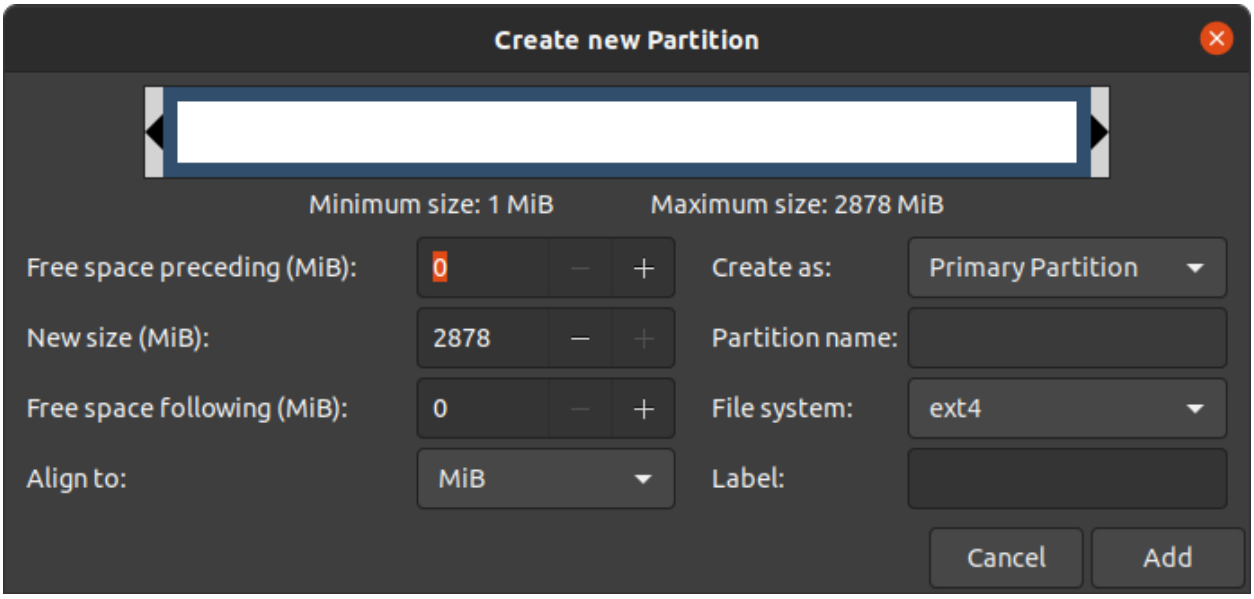
```
host:~$ sudo cp phytec-qt6demo-image-imx95-phyflex-libra-rdk-2.wic /mnt/ ; sync
host:~$ umount /mnt
```

Create the Third Partition

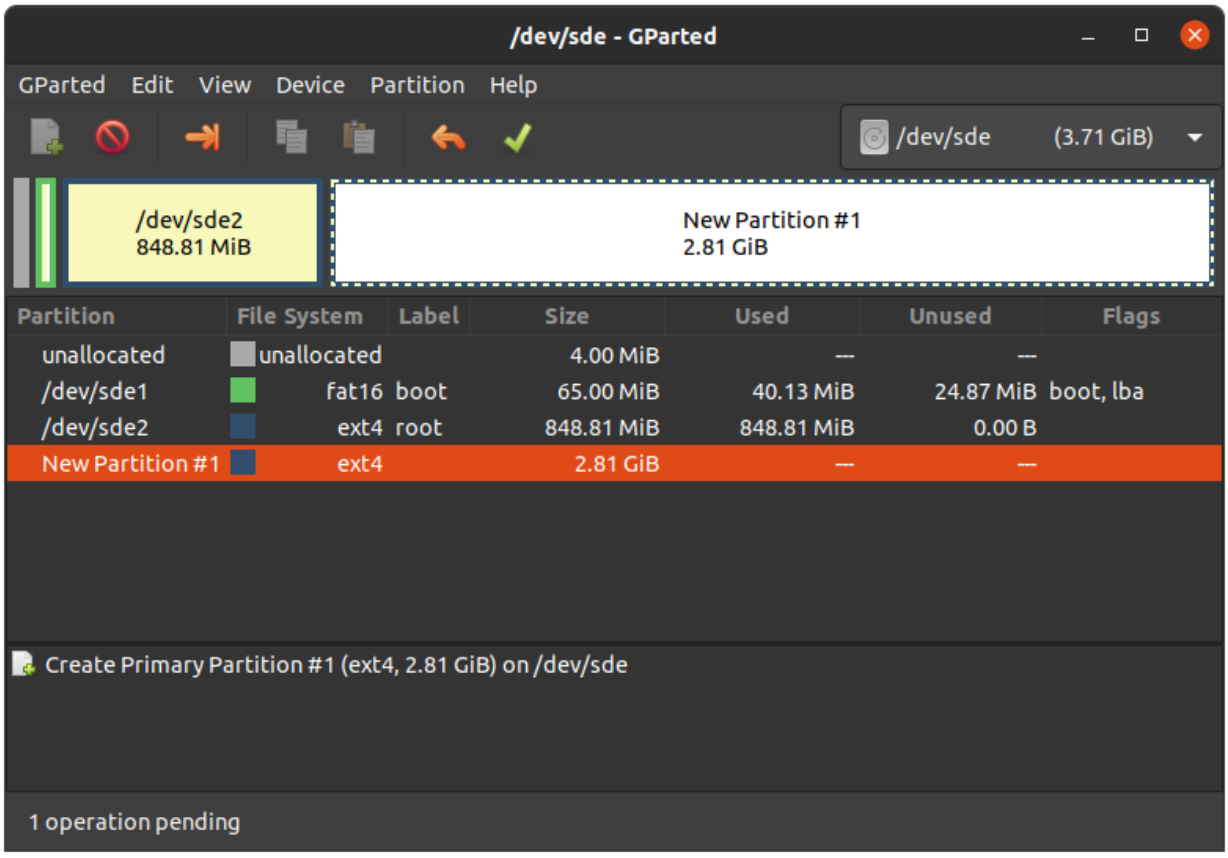
- Choose your SD card device at the drop-down menu on the top right



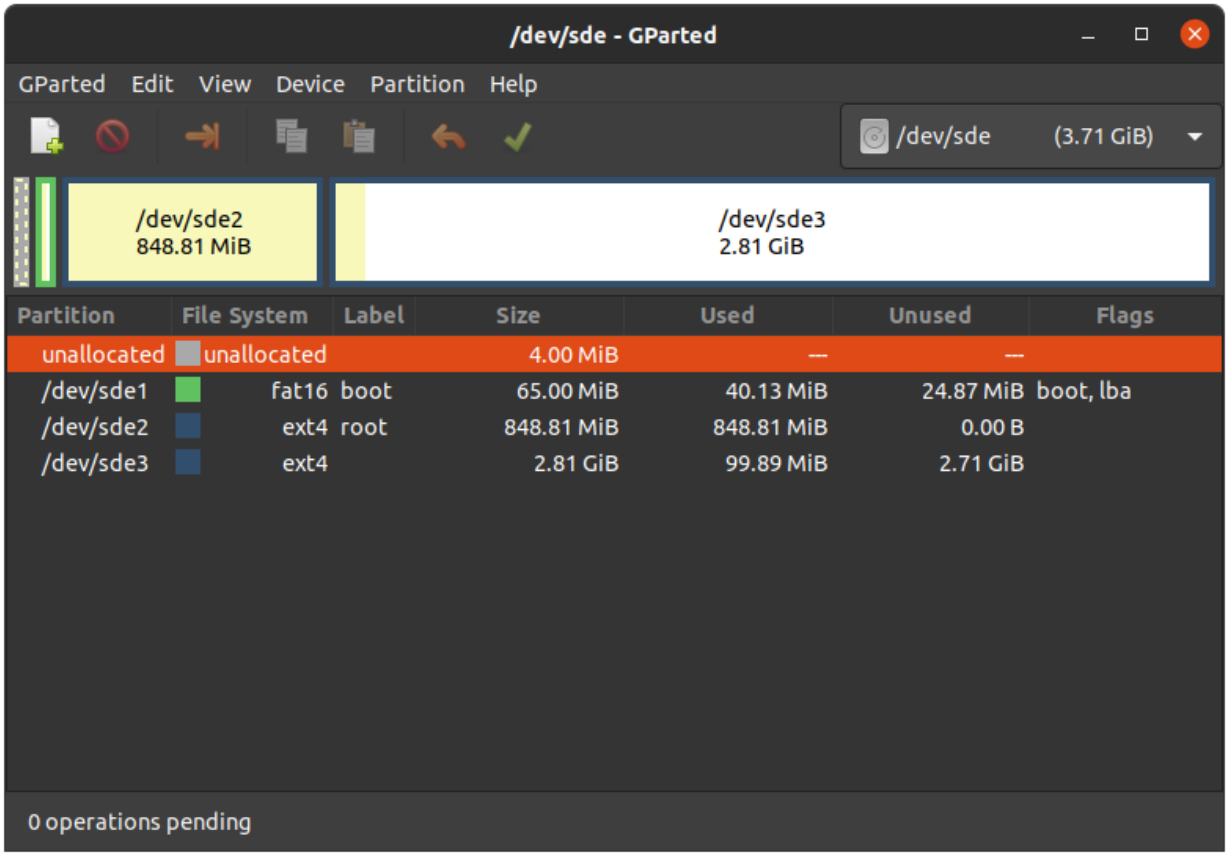
- Choose the bigger unallocated area and press “New”:



- Click “Add”



- Confirm your changes by pressing the green tick.



- Now you can mount the new partition and copy e.g. phytec-qt6demo-image-imx95-phyflex-libra-rdk-2.wic image to it. Then unmount it again:

```
host:~$ sudo mount /dev/sde3 /mnt
host:~$ sudo cp phytec-qt6demo-image-imx95-phyflex-libra-rdk-2.wic /mnt/ ; sync
host:~$ umount /mnt
```

5.10 Ampliphy-boot

Ampliphy-boot is a collection of bootscripts for booting the Ampliphy distro, which work uniformly across different PHYTEC platforms. The bootscripts are developed to work with the standard boot U-Boot feature. More details about ampliphy-boot, the bootscripts and their Yocto implementation can be found in the Yocto Reference Manual (walnascar).

Ampliphy-boot currently provides these bootscripts:

```
mmc_boot
mmc_boot_fit
net_boot_fit
spi_boot_fit
```

Not all platforms support all scripts. The **phyFLEX-i.MX 95 Libra Rapid Development Kit** supports the following bootscripts:

```
mmc_boot_fit
net_boot_fit
```

5.10.1 U-Boot standard boot

U-Boot standard boot is a built-in way to boot different operating systems. It handles scanning devices for the supported boot methods and starts the bootflow.

For ampliphy-boot the `script` bootmethod is used. This means a bootscript is placed inside the boot partition on the bootdevice that should be booted. This bootscript contains the logic for booting the ampliphy distro that previously was stored inside the U-Boot environment. This leads to the environment being a lot less cluttered and better readable for the user.

U-Boot standard boot uses and sets some environment variables, which are documented here. The most interesting variables for the BSP user are:

`ip_dyn`
: determines if the bootscript is loaded with dhcp (dynamic ip) or tftp (static ip). Values can be [yes/no], [true/false], [1/0]. If the variable is not set, dhcp will be used as default.

`boot_script_dhcp`
: Sets the name of the bootscript loaded from network.

`boot_targets`
: Determines the scanned bootsources and their order.

`bootmeths`
: Determines the scanned bootmethods and their order.

Default values for `boot_targets` and `bootmeths` for standard boot can be set in the devicetree or in the environment. The environment variables will overwrite the devicetree values if set.

For the **phyFLEX-i.MX 95 Libra Rapid Development Kit**, the default values are defined in the U-Boot devicetree (e.g. arch/arm/dts/imx95-phyflex-libra-rdk-u-boot.dtsi):

```
bootstd {
        bootph-verify;
        compatible = "u-boot,boot-std";

        filename-prefixes = "/", "/boot/";
        bootdev-order = "mmc0", "mmc1", "ethernet";

        script {
                compatible = "u-boot,script";
        };
};
```

The filename-prefixes property describes the paths that will be searched for the bootscripts. In this case this is the root of the partition as well as the boot folder. The bootdev-order property sets the default value for the boot_targets variable. The supported bootmeths will also be named. In this case only the script method is supported.

5.11 Working with FIT images

Flattened Image Tree (FIT) images can be used to pack binaries into a devicetree blob. They are used in our BSPs to pack the kernel, devicetree and devicetree overlays.

5.11.1 Show FIT image content

Printing the FIT image content on the host can be done with the following commands:

```
host:~$ dumpimage -l fitImage
```

or

```
host:~$ mkimage -l fitImage
```

Alternatively the content can also be printed on target in U-Boot:

```
u-boot=> load mmc ${mmcdev}:1 ${loadaddr} fitImage
u-boot=> iminfo ${loadaddr}
```

5.11.2 Building fitImages standalone

For building a FIT, an image tree source file (`*.its`) and binaries to be packed are needed. The `fitImage.its` file and kernel and devicetree binaries can be fetched from the Yocto deploy folder, the BSP downloads page or can be built standalone. The paths of the binaries in the `data` properties may need adjustment. For the kernel, the default data property will mention `linux.bin`, but the binary from the Yocto build is named `Image`.

Default:

```
images {
             kernel-1 {
                     description = "Linux kernel";
                     data = /incbin/("linux.bin");
                     type = "kernel";
...
```

Customized:

```
images {
             kernel-1 {
                     description = "Linux kernel";
                     data = /incbin/("Image");
                     type = "kernel";
...
```

Data properties for devicetrees also may need to be adjusted. After that the FIT image can be built using the `mkimage` command.

```
mkimage -f fitImage.its fitImage
```

5.11.3 Adding overlays to a FIT image

Adding overlays to the FIT image can be done by editing the `fitImage.its` file. For that the overlays need to be added to the `images` node and configurations need to be created for them. Already existing overlays can be used for reference. Only the names need to be adjusted.

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 95 BSP Device Tree Concept

The following sections explain some rules PHYTEC has defined on how to set up device trees for our i.MX 95 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 95 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 9 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 folder `arch/arm64/boot/dts/freescale/`.

Available overlays for imx95-phyflex-libra-rdk-2.conf are:

```
imx95-phyflex-libra-rdk-bluetooth-88w8987.dtbo
imx95-phyflex-libra-rdk-lvds-ph128800t006-zhc01.dtbo
```

(continues on next page)

(continued from previous page)

```
imx95-phyflex-libra-rdk-neoisp.dtbo
imx95-phyflex-libra-rdk-vm016-csi1.dtbo
imx95-phyflex-libra-rdk-vm016-fpdlink-port0-csi1.dtbo
imx95-phyflex-libra-rdk-vm016-fpdlink-port1-csi1.dtbo
imx95-phyflex-libra-rdk-vm016-csi2.dtbo
imx95-phyflex-libra-rdk-vm016-fpdlink-port0-csi2.dtbo
imx95-phyflex-libra-rdk-vm016-fpdlink-port1-csi2.dtbo
imx95-phyflex-libra-rdk-vm017-csi1.dtbo
imx95-phyflex-libra-rdk-vm017-fpdlink-port0-csi1.dtbo
imx95-phyflex-libra-rdk-vm017-fpdlink-port1-csi1.dtbo
imx95-phyflex-libra-rdk-vm017-csi2.dtbo
imx95-phyflex-libra-rdk-vm017-fpdlink-port0-csi2.dtbo
imx95-phyflex-libra-rdk-vm017-fpdlink-port1-csi2.dtbo
imx95-phyflex-libra-rdk-vm020-csi1.dtbo
imx95-phyflex-libra-rdk-vm020-fpdlink-port0-csi1.dtbo
imx95-phyflex-libra-rdk-vm020-fpdlink-port1-csi1.dtbo
imx95-phyflex-libra-rdk-vm020-csi2.dtbo
imx95-phyflex-libra-rdk-vm020-fpdlink-port0-csi2.dtbo
imx95-phyflex-libra-rdk-vm020-fpdlink-port1-csi2.dtbo
imx95-phyflex-fpsc-g-som-temperature.dtbo
```

Otherwise you can show the content of a FIT image including all overlay configs in the FIT image with this command in Linux:

```
host:~$ dumpimage -l fitImage
```

or in U-Boot:

```
u-boot=> load mmc ${mmcdev}:1 ${loadaddr} fitImage
u-boot=> iminfo
```

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.

Loading overlays

FIT image bootscripts

In ampliphy-boot FIT image bootscripts, the `fit_overlay_conf` U-Boot environment variable contains a number-sign (#) separated list of overlay configurations that will be applied during boot. This variable is used for overlays describing expansion boards and cameras that can not be detected during run time. The overlays listed in the fit_overlay_conf variable must be included in the FIT image. Overlays set in the $KERNEL_DEVICETREE Yocto machine variable will automatically be added to the FIT image.

The `fit_overlay_conf` variable can either be set directly in the U-Boot environment or can be part of the external `overlays.txt` environment file. By default, the fit_overlay_conf variable comes from the external overlays.txt environment file which is located in the boot partition. Otherwise if the fit_overlay_conf variable is set in U-Boot environment, the overlays.txt file value will not be loaded. The content from the file is defined in the Yocto recipe bootenv found in meta-phytec: https://github.com/phytec/meta-phytec/tree/walnascar/recipes-bsp/bootenv You can read and write the file on booted target from linux:

```
target:~$ cat /boot/overlays.txt
fit_overlay_conf=conf-imx95-phyflex-libra-rdk-overlay1.dtbo#conf-imx95-phyflex-libra-rdk-
↪overlay2.dtbo
```

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

```
u-boot=> setenv fit_overlay_conf conf-imx95-phyflex-libra-rdk-overlay1.dtbo
u-boot=> printenv fit_overlay_conf
fit_overlay_conf=conf-imx95-phyflex-libra-rdk-overlay1.dtbo
u-boot=> boot
```

Non FIT image bootscripts

In ampliphy-boot non FIT image bootscripts, the `overlays` U-Boot environment variable contains a space ( ) separated list of overlays that will be applied during boot. The overlays variable can also be set directly in the U-Boot environment or can be part of the external `overlays.txt` environment file.

By default, the overlays variable comes from the external overlays.txt environment file which is located in the boot partition. Otherwise if the overlays variable is set in U-Boot environment, the overlays.txt file value will not be loaded. The content from the file is defined in the Yocto recipe bootenv found in meta-phytec: https://github.com/phytec/meta-phytec/tree/walnascar/recipes-bsp/bootenv You can read and write the file on booted target from linux:

```
target:~$ cat /boot/overlays.txt
overlays=imx95-phyflex-libra-rdk-overlay1.dtbo imx95-phyflex-libra-rdk-overlay2.dtbo
```

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

```
u-boot=> setenv overlays imx95-phyflex-libra-rdk-overlay1.dtbo
u-boot=> printenv overlays
overlays=imx95-phyflex-libra-rdk-overlay1.dtbo
u-boot=> boot
```

Overlay detection for SoM Variants (fit_extensions)

For FIT image bootscripts, additional overlays are applied automatically to 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=> env print fit_extensions
```

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

To prevent application of the SoM variant related overlays the `no_extensions` variable can be set to *1* in the bootloader environment.

```
u-boot=> setenv no_extensions 1
u-boot=> env save
u-boot=> reset
```

6.2.3 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 phyFLEX-i.MX 95 FPSC 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 reused 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 95 BSP Device Tree Concept to get an understanding of our i.MX 9 BSP device tree model.

The following sections provide an overview of the supported hardware components and their operating system drivers on the i.MX 9 platform. Further changes can be ported upon customer request.

7.1 i.MX 95 Pin Muxing

The i.MX 95 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 95 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 95 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 lpuart7 device in imx95-phyflex-fpsc-g-som.dtsi:

```
pinctrl_lpuart7: lpuart7grp {
        fsl,pins = <
                IMX95_PAD_GPIO_IO37__LPUART7_RX      0x31e   /* UART3_RXD */
                IMX95_PAD_GPIO_IO36__LPUART7_TX      0x31e   /* UART3_TXD */
        >;
};
```

The first part of the string IMX95_PAD_GPIO_IO37___LPUART7_RX names the pad (in this example IMX95_PAD_GPIO_IO37). The second part of the string (LPUART7_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 enabled.

The device tree representation for UART1 pinmuxing: https://github.com/phytec/linux-phytec-imx/tree/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx95-phyflex-fpsc-g-som.dtsi#L264

7.2 Ethernet

phyFLEX Libra RDK-i.MX 95 provides three ethernet interfaces. A gigabit Ethernet is provided by our module and board. Additionally there is a 10Gbit Ethernet. Currently only the one Gigabit Ethernet ports are supported (Ethernet1 and Ethernet2).

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 interfaces are configured to static IP as default. The default IP address and netmask for end1 is:

```
end1: 192.168.3.11/24
```

To configure end1 to dynamic IP over DHCP, go to `/lib/systemd/network/\*-end1.network` and delete the line:

```
Address=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 ethernet where the PHY is populated on the SoM can be found here: https://github.com/phytec/linux-phytec-imx/tree/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx95-phyflex-fpsc-g-som.dtsi#L90.

The device tree set up for the ethernet where the PHY is populated on the phyFLEX Libra RDK can be found here: https://github.com/phytec/linux-phytec-imx/tree/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx95-phyflex-libra-rdk.dts#L196.

7.2.1 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.

Multiple network interfaces in U-Boot

Some boards may have support for multiple network interfaces in U-Boot. U-Boot lists all supported network interfaces during startup. This generic example illustrates support for multiple network interfaces:

```
Net:
eth0: ethernet@10000000, eth1: ethernet@10010000
```

There are three environment variables that will contribute to the selection of the ethernet interface for example when loading a file over tftp.

ethrotate
: Determines if other interfaces will be probed if the first one fails. Valid values are yes/no. Default is yes if unset.

ethact
: Sets the current active ethernet interface. This interface will be used (first). It is usually unset at boot, but will be set automatically by U-Boot to the last used interface as soon as ethernet is used for the first time. U-Boot will overwrite existing values. Only one ethernet interface can be set, lists are not allowed.

ethprime
: This will determine the preferred ethernet interface if ethact is unset. Otherwise this will be ignored. Only one ethernet interface can be set, lists are not allowed.

Note

The variables `ipaddr`, `serverip` and `netmask` are global and not specific to a certain ethernet interface, so they might need to be adapted to the ethernet interface in use.

Kernel network-environment

- Find the ethernet settings for end1 in the target kernel:

```
target:~$ ip -statistics address show end1
2: end1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state UP group default qlen␣
↪1000
    link/ether 50:2d:f4:19:d6:33 brd ff:ff:ff:ff:ff:ff
    RX:  bytes packets errors dropped  missed   mcast
               0       0      0       0       0       0
    TX:  bytes packets errors dropped carrier collsns
               0       0      0       0       0       0
```

- Temporary adaption of the end1 configuration:

```
target:~$ ip address add 192.168.3.11/24 dev end1
```

7.3 WLAN/Bluetooth

WLAN and Bluetooth on the phyFLEX Libra RDK are provided by the M.2 ublox MAYA-W271-00B expansion card. This module supports 2,4 and 5 GHz bandwidth and can be run in several modes, like client mode and Access Point (AP). More information about the module can be found at https://www.u-blox.com/en/product/maya-w2-series

To use the bluetooth connection, the overlay needs to be activated first, otherwise the UART connection isn't configured as needed.

```
target:~$ vi /boot/bootenv.txt
```

Afterwards the bootenv.txt file should look like this (it can also contain other devicetree overlays!):

```
fit_overlay_conf=conf-imx95-phyflex-libra-rdk-bluetooth-88w8987.dtbo
```

The changes will be applied after a reboot:

```
target:~$ reboot
```

For further information about devicetree overlays, read the *device tree* chapter.

Note

The following WLAN chapter assumes wireless network interface name is `wlan0`. However with MAYA-W271-00B adapter the name of the WLAN interface is actually `mlan0`. Thus when using commands to configure wireless network, substitute `wlan0` with `mlan0` when using MAYA-W271-00B.

7.3.1 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:

```
Successfully initialized wpa_supplicant
```

The ip address is automatically configured over DHCP. For other possible IP configurations, see section *Changing the Network Configuration* in the Yocto Reference Manual (walnascar).

7.3.2 Bluetooth

Bluetooth is connected to UART3 interface. The Bluetooth device needs to be set up manually:

```
target:~$ hciconfig hci0 up

target:~$ hciconfig -a
```

(continues on next page)

(continued from previous page)

```
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>
```

7.3.3 Visibility

To activate visibility:

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

To disable visibility:

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

7.3.4 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
```

7.4 SD card

The i.MX 95 supports a slot for Secure Digital 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 card, the kernel will generate new device nodes in /dev. The full device can be reached via its /dev/mmcblk1 device node. SD 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://github.com/phytec/linux-phytec-imx/tree/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx95-phyflex-fpsc-g-som.dtsi#L697

DT configuration for the eMMC interface can be found here: https://github.com/phytec/linux-phytec-imx/tree/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx95-phyflex-fpsc-g-som.dtsi#L684

7.5 e.MMC Devices

PHYTEC modules like phyFLEX-i.MX 95 FPSC are populated with an e.MMC memory chip as the main storage. e.MMC 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 95 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 e.MMC manufacturer’s datasheet is relatively short and meant to be read together with the supported version of the JEDEC e.MMC standard.

PHYTEC currently utilizes the e.MMC chips with JEDEC Version 5.0 and 5.1

7.5.1 Extended CSD Register

e.MMC 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/mmcblk0
```

You will see:

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

```

(continues on next page)

(continued from previous page)

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

7.5.2 Enabling Background Operations (BKOPS)

In contrast to raw NAND Flash, an e.MMC 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 e.MMC 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 e.MMC 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/mmcblk0 | 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/mmcblk0
```

- 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 e.MMC device/partition.
2. Reliable Write for single write transactions.

Tip

Do not confuse e.MMC 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 e.MMCs mounted on the phyFLEX-i.MX 95 FPSC SoMs. To check this on the running target:

```
target:~$ mmc extcsd read /dev/mmcblk0 | 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.
```

(continues on next page)

(continued from previous page)

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

- Get the current device size:

```
target:~$ parted /dev/mmcblk0 print
```

- The output looks like this:

```
Model: MMC Q2J55L (sd/mmc)
Disk /dev/mmcblk0: 7617MB
Sect[ 1799.850385]  mmcblk0: 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/mmcblk0 resizepart 2 100%
Information: You may need to update /etc/fstab.

target:~$ parted /dev/mmcblk0 print
Model: MMC Q2J55L (sd/mmc)
Disk /dev/mmcblk0: 7617MB
Sector size (logical/physical): 512[ 1974.191657]  mmcblk0: 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/mmcblk0p2
resize2fs 1.46.1 (9-Feb-2021)
Filesystem at /dev/mmcblk0p2 is mounted on /; on-line resizing required
[ 131.609512] EXT4-fs (mmcblk0p2): resizing filesystem
from 454136 to 7367680 blocks
old_desc_blocks = 4, new_desc_blocks = 57
[ 131.970278] EXT4-fs (mmcblk0p2): resized filesystem to 7367680
The filesystem on /dev/mmcblk0p2 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 e.MMC partition while it is not mounted.

7.5.5 Enable pseudo-SLC Mode

e.MMC 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 e.MMC device with:

  ```
  target:~$ parted -m /dev/mmcblk0 unit B print
  ```

  You will receive:

  ```
  BYT;
  /dev/mmcblk0:63652757504B: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/mmcblk0 | 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/mmcblk0
```

  You will get:

```
Done setting ENH_USR area on /dev/mmcblk0
setting OTP PARTITION_SETTING_COMPLETED!
Setting OTP PARTITION_SETTING_COMPLETED on /dev/mmcblk0 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/mmcblk0 | 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/mmcblk0 unit B print
BYT;
/dev/mmcblk0:31742492672B:sd/mmc:512:512:unknown:MMC S0J58X:;
```

7.5.6 Erasing the Device

It is possible to erase the e.MMC device directly rather than overwriting it with zeros. The e.MMC 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/mmcblk0
```

  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/mmcblk0 conv=fsync
```

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

7.5.7 e.MMC Boot Partitions

An e.MMC 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 95 uses to load the bootloader is controlled by the boot configuration of the e.MMC 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 e.MMC partitions with partitions of a DOS, MBR, or GPT partition table.

The current PHYTEC BSP does not use the extra partitioning feature of e.MMC 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 e.MMC flash storage it is possible to use the dedicated boot partitions for redundantly storing the bootloader. The Bootloader 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.

Via userspace Commands

On the host, run:

```
host:~$ scp <bootloader> 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 e.MMC boot partitions, first disable the write protection:

```
target:~$ echo 0 > /sys/block/mmcblk0boot0/force_ro
target:~$ echo 0 > /sys/block/mmcblk0boot1/force_ro
```

Write the bootloader to the e.MMC boot partitions:

```
target:~$ dd if=/tmp/<bootloader> of=/dev/mmcblk0boot0
target:~$ dd if=/tmp/<bootloader> of=/dev/mmcblk0boot1
```

The following table is for the offset of the i.MX 95 SoC:

SoC	Offset User Area	Offset Boot Partition	e.MMC Device
i.MX 95	32 kiB	0 kiB	/dev/mmcblk0

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

(for 'boot0') :

```
target:~$ mmc bootpart enable 1 0 /dev/mmcblk0
```

(for 'boot1') :

```
target:~$ mmc bootpart enable 2 0 /dev/mmcblk0
```

To disable booting from the e.MMC boot partitions simply enter the following command:

```
target:~$ mmc bootpart enable 0 0 /dev/mmcblk0
```

To explicitly enable booting from the e.MMC user area, run:

```
target:~$ mmc bootpart enable 7 0 /dev/mmcblk0
```

7.6 GPIOs

The phyFLEX Libra RDK has a set of pins especially dedicated to user I/Os. Those pins are connected directly to i.MX 95 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 95 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 -c gpiochip0
```

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

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

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

```
target:~$ gpioset -z -c gpiochip0 20=0
```

- Help text of gpioset shows possible options:

```
target:~$ gpioset --help
Usage: gpioset [OPTIONS] <line=value>...

Set values of GPIO lines.

Lines are specified by name, or optionally by offset if the chip option
is provided.
Values may be '1' or '0', or equivalently 'active'/'inactive' or 'on'/'off'.

The line output state is maintained until the process exits, but after that
is not guaranteed.

Options:
      --banner            display a banner on successful startup
  -b, --bias <bias>       specify the line bias
                  Possible values: 'pull-down', 'pull-up', 'disabled'.
                  (default is to leave bias unchanged)
      --by-name           treat lines as names even if they would parse as an offset
  -c, --chip <chip>       restrict scope to a particular chip
  -C, --consumer <name>   consumer name applied to requested lines (default is 'gpioset')
  -d, --drive <drive>     specify the line drive mode
                  Possible values: 'push-pull', 'open-drain', 'open-source'.
                  (default is 'push-pull')
  -h, --help              display this help and exit
  -l, --active-low        treat the line as active low
  -p, --hold-period <period>
                  the minimum time period to hold lines at the requested values
  -s, --strict            abort if requested line names are not unique
  -t, --toggle <period>[,period]...
                  toggle the line(s) after the specified period(s)
                  If the last period is non-zero then the sequence repeats.
      --unquoted  don't quote line names
  -v, --version           output version information and exit
  -z, --daemonize set values then detach from the controlling terminal

Chips:
    A GPIO chip may be identified by number, name, or path.
    e.g. '0', 'gpiochip0', and '/dev/gpiochip0' all refer to the same chip.

Periods:
    Periods are taken as milliseconds unless units are specified. e.g. 10us.
```

(continues on next page)

(continued from previous page)

```
    Supported units are 's', 'ms', and 'us'.

*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.

7.6.1 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 imx9_phytec_defconfig:

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

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

7.7 I²C Bus

The i.MX 95 contains several Multimaster fast-mode I²C modules. PHYTEC boards provide plenty of different I²C devices connected to the I²C modules of the i.MX 95. This section describes the basic device usage and its DT representation of some I²C devices integrated into our phyFLEX Libra RDK.

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 I²C bus configuration from SoM (e.g. imx95-phyflex-fpsc-g-som.dtsi): https://github.com/phytec/linux-phytec-imx/tree/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx95-phyflex-fpsc-g-som.dtsi#L146

General I²C bus configuration from carrierboard (e.g. imx95-phyflex-libra-rdk.dts) https://github.com/phytec/linux-phytec-imx/tree/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/

imx95-phyflex-libra-rdk.dts#L242

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::@
```

The phyFLEX Libra RDK provides the following LED indicators: red:status, green:status and blue:status.

- To toggle the LEDs ON:

  ```
  target:~$ echo 255 > /sys/class/leds/red:status/brightness
  ```

- To toggle OFF:

  ```
  target:~$ echo 0 > /sys/class/leds/red:status/brightness
  ```

7.9 CAN FD

The phyFLEX Libra RDK has two flexCAN interfaces supporting CAN FD. They are supported by the Linux standard CAN framework which builds upon 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

- Use:

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

  to see the state of the interfaces.

- To get information on fcan1, such as bit rate and error counters, type:

  ```
  target:~$ ip -d -s link show fcan1
  4: fcan1: <NOARP,UP,LOWER_UP,ECHO> mtu 72 qdisc pfifo_fast state UP mode DEFAULT group␣
  ↪default qlen 10
     link/can  promiscuity 0 allmulti 0 minmtu 0 maxmtu 0
     can <FD> state ERROR-ACTIVE (berr-counter tx 0 rx 0) restart-ms 0
           bitrate 500000 sample-point 0.875
           tq 25 prop-seg 37 phase-seg1 32 phase-seg2 10 sjw 5 brp 1
           flexcan: tseg1 2..96 tseg2 2..32 sjw 1..16 brp 1..1024 brp_inc 1
           dbitrate 2000000 dsample-point 0.750
           dtq 25 dprop-seg 7 dphase-seg1 7 dphase-seg2 5 dsjw 2 dbrp 1
           flexcan: dtseg1 2..39 dtseg2 2..8 dsjw 1..4 dbrp 1..1024 dbrp_inc 1
  ```

(continues on next page)

(continued from previous page)

```
        clock 40000000
        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 tso_max_size 65536 tso_max_segs 65535␣
↪gro_max_size 65536
gso_ipv4_max_size 65536 gro_ipv4_max_size 65536 parentbus platform parentdev 443a0000.can
   RX:  bytes packets errors dropped  missed   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:

fcan1	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/11-fcan.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/11-fcan.network` in the root filesystem and rebuild the root filesystem.

```
[Match]
Name=can*

[CAN]
BitRate=500000
DataBitRate=2000000
FDMode=yes
```

Note

By default, we enable CAN-FD (flexible datarate) in our BSP. In case CAN Classic is required one needs to remove options `FDMode` and `DataBitRate` from the `/lib/systemd/network/11-fcan.network` file.

To disable flexible datarate manually, one can use:

```
target:~$ ip link set fcan1 down
target:~$ ip link set fcan1 type can bitrate 500000 dbitrate 0 fd off
target:~$ ip link set fcan1 up
```

The bitrate can also be changed manually, for example:

```
target:~$ ip link set fcan1 down
target:~$ ip link set fcan1 txqueuelen 10 up type can bitrate 500000 sample-point 0.75 dbitrate␣
↪4000000 dsample-point 0.8 fd on
target:~$ ip link set fcan1 up
```

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

```
target:~$ cansend fcan1 123#45.67
target:~$ candump fcan1
```

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.

Device Tree CAN configuration of imx95-phyflex-libra-rdk.dts:
https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy6/arch/arm64/boot/dts/freescale/imx95-phyflex-libra-rdk.dts#L208

7.10 RS232/RS485

7.10.1 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/ttyLP1 115200 crtscts raw -echo
  ```

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

  ```
  target:~$ echo 123 > /dev/ttyLP1
  ```

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

The host should print out “123”.

7.10.2 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/ttyLP1 -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/ttyLP1
```

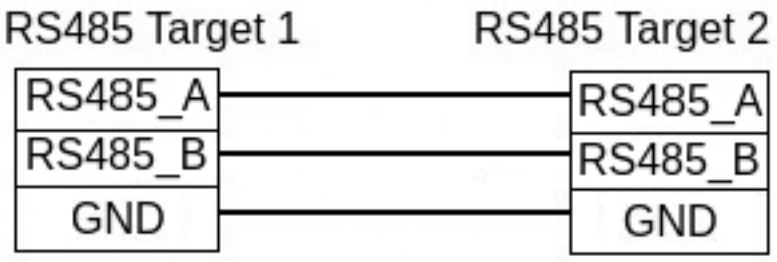
You can show all options with:

```
target:~$ rs485conf /dev/ttyLP1 -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/ttyLP1 -e 1 -r 0
target:~$ rs485conf /dev/ttyLP1
= Current configuration:
RS485 enabled:                true
RTS on send:                  high
RTS after send:               low
RTS delay before send:        0
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/ttyLP1
target2:~$ echo test > /dev/ttyLP1
```

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/ttyLP1 -b 115200 --rs485 0 -t -i 8
...
```

(continues on next page)

(continued from previous page)

```
/dev/ttyLP1: count for this session: rx=57330, tx=0, rx err=0
target2:~$ linux-serial-test -s -e -f -p /dev/ttyLP1 -b 115200 --rs485 0 -r -o 5
...
/dev/ttyLP1: 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.

7.11 EEPROM

The system features three I2C EEPROM devices distributed across the SoM and carrier board:

On the phyFLEX-i.MX 95 FPSC SoM:

- SoM Detection EEPROM (write-protected)
  - Bus: I2C-5
  - Address: 0x51
  - Purpose: Factory configuration for SoM identification
- User EEPROM
  - Bus: I2C-5
  - Address: 0x50
  - Purpose: Available for user applications

Device Tree Reference for SoM EEPROMs: https://github.com/phytec/linux-phytec-imx/tree/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx95-phyflex-fpsc-g-som.dtsi#L181

And on the phyFLEX Libra RDK carrier board:

- Board Detection EEPROM
  - Bus: I2C-2
  - Address: 0x51
  - Purpose: Reserved for carrier board identification
- User EEPROM
  - Bus: I2C-2
  - Address: 0x52
  - Purpose: Available for user applications

Device Tree Reference for Carrier Board: https://github.com/phytec/linux-phytec-imx/tree/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx95-phyflex-libra-rdk.dts#L339

7.12 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
```

(continues on next page)

(continued from previous page)

```
target:~$ hwclock -s
target:~$ date
Wed Mar  2 10:45:01 UTC 2022
```

7.12.1 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.

7.12.2 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 0x71.
  ```

  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
  #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_ALARM_WAKEUP_ONLY   7
  #define RTC_FEATURE_CNT                 8
  ```

- 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://github.com/phytec/linux-phytec-imx/tree/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx95-phyflex-fpsc-g-som.dtsi#L197

And the additions on the carrierboard: https://github.com/phytec/linux-phytec-imx/tree/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx95-phyflex-libra-rdk.dts#L436

7.13 USB Host Controller

The USB controller of the i.MX 95 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 of up to 4 Gbit/s (SuperSpeed 'SS'). The USB subsystem has two independent USB controller cores. Both cores are capable of acting as a USB peripheral device or a USB host. Each is connected to a USB 3.0 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.

DT representation for USB Host: https://github.com/phytec/linux-phytec-imx/tree/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx95-phyflex-libra-rdk.dts#L538

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.

Example:

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

Hint

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

```
target:~$ cd /sys/kernel/config/usb_gadget/
target:~$ mkdir g1
target:~$ cd g1/
target:~$ echo "0x1d6b" > idVendor
target:~$ echo "0x0104" > idProduct
```

(continues on next page)

(continued from previous page)

```
target:~$ mkdir strings/0x409
target:~$ echo "0123456789" > strings/0x409/serialnumber
target:~$ echo "Foo Inc." > strings/0x409/manufacturer
target:~$ echo "Bar Gadget" > strings/0x409/product
```

Next, create a file with a filesystem for the mass storage gadget:

```
target:~$ dd if=/dev/zero of=/tmp/file.img bs=1M count=64
target:~$ mkfs.ext4 /tmp/file.img
```

Note

If you create the file in the tmp folder it will be gone after the next reboot. If you want to have it persistent, use an other directory.

Now you should create the functions you want to use:

```
target:~$ cd /sys/kernel/config/usb_gadget/g1
target:~$ mkdir functions/acm.GS0
target:~$ mkdir functions/ecm.usb0
target:~$ mkdir functions/mass_storage.0
```

- *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.

Now set the file you want to share with the host:

```
target:~$ echo /tmp/file.img > functions/mass_storage.0/lun.0/file
```

Hint

You can also insert partitions or a whole device to be shared with the host here, but the partition to be shared or the partitions on the device to be shared should not be mounted on the target while sharing with the host otherwise writing to them will not work from host.

Bind the defined functions to a configuration:

```
target:~$ cd /sys/kernel/config/usb_gadget/g1
target:~$ mkdir configs/c.1
target:~$ mkdir configs/c.1/strings/0x409
target:~$ echo "CDC ACM+ECM+MS" > configs/c.1/strings/0x409/configuration
target:~$ ln -s functions/acm.GS0 configs/c.1/
target:~$ ln -s functions/ecm.usb0 configs/c.1/
target:~$ 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 --indicator-style=none /sys/class/udc/
ci_hdrc.0
target:~$ echo "ci_hdrc.0" > UDC
```

`ci_hdrc.0` is an example, replace it with the actual name. Any trailing `@` might be shown by `ls` to show it is a link, remove it. 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
```

After stopping the sharing with the host, you can also mount the file on the target. This way files can be transferred between host and target.

```
target:~$ mount /tmp/file.img /mnt
```

Warning

Do not mount the file while it is shared with the host.

7.14 Video

7.14.1 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
```

7.15 Display

The phyFLEX Libra RDK supports up to 3 different display outputs. The following table shows the required extensions and devicetree overlays for the different interfaces. We support the `powertip,ph128800t006-zhc01` display.

Interface	Expansion	devicetree overlay
LVDS1	phyFLEX Libra RDK	imx95-phyflex-libra-rdk-lvds-ph128800t006-zhc01.dtbo
LVDS2	PEB-AV-10	imx95-phyflex-libra-rdk-peb-av-10-ph128800t006-zhc01.dtbo

Note

- When changing Weston output, make sure to match the audio output as well.
- LVDS2 (PEB-AV-10) and LVDS1 (onboard) can not be used at the same time.

The default interface is LVDS1 (onboard LVDS).

Note

The current display driver limits the pixel clock for a display connected to LVDS to 74.25Mhz (or a divider of it). If this does not fit your display requirements, please contact Support for further help.

7.15.1 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.

7.15.2 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.

Note

On current hardware (1618.2) the backlight brightness 0 will not turn off the backlight enable. Therefore the backlight will be set to max brightness on brightness level 0.

Device tree description of LVDS-1 can be found here: https://github.com/phytec/linux-phytec-imx/tree/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx95-phyflex-libra-rdk-lvds.dtsi#L30

7.16 Power Management

7.16.1 CPU Core Frequency Scaling

The CPU in the i.MX 95 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 95 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 9 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`.

7.16.2 CPU Core Management

The i.MX 95 SoC can have multiple processor cores on the die. The i.MX 95, 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
```

7.16.3 Suspend to RAM

The phyFLEX-i.MX 95 FPSC 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/ttyLP3/power/wakeup
target:~$ echo mem > /sys/power/state
```

7.17 Thermal Management

7.17.1 U-Boot

There is no Thermal Management support in the first ALPHA release for the i.MX95 in U-Boot.

7.17.2 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 95 SoC platform.

There are nine temperature sensors on the SoM that are readable from Linux. The i.MX 9 has two internal temperature sensors for the SoC. Three internal sensors for the PMICs and four I²C temperature sensors located close to DRAM, eMMC, ethernet PHY and PMIC.

- The current temperatures of the system can be read in milli celsius over

```
target:~$ cat /sys/class/hwmon/hwmon*/temp*_input
```

- You will get, for example:

```
48590
48510
105000
105000
105000
43562
43812
43062
44875
```

Note

The PMIC temperature sensors return only the last triggered threshold values and not the actual temperature values. The thresholds are 110°C, 125°C, 140°C and 155°C. All temperatures lower than 110°C are shown as 105°C as seen in the example.

There are two trip points registered in the device tree. These may differ depending on the CPU variant. A distinction is made between Commercial, Industrial and Extended Industrial. For the ALPHA1 i.MX95 release there is only the Automotive/Extended Industrial temperature range available.

trip point	Extended Industrial
passive (warning)	105°C
critical (shutdown)	125°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 and Power Allocator. The default policy used in the BSP is step_wise.

Tip

If the value of the SoC temperature in the sysfs file temp reaches *trip_point_1* (critical), the board immediately shuts down to avoid any heat damage.

7.18 PWM fan

The phyFLEX Libra RDK features a PWM fan connector. The fan's operation is coupled to the temperature of the SoC and may either be controlled by the kernel or sensord. Inspecting the fan's parameters can be done via sysfs. Quoting from the Linux kernel documentation, the following parameters are available:

fan1_	ro	fan tachometer speed in RPM
pwm1	rw	keep enable mode, defines behaviour when pwm1=0 0 -> disable pwm and regulator 1 -> enable pwm; if pwm==0, disable pwm, keep regulator enabled 2 -> enable pwm; if pwm==0, keep pwm and regulator enabled 3 -> enable pwm; if pwm==0, disable pwm and regulator
pwm1	rw	relative speed (0-255), 255=max. speed.

These files are available under `/sys/class/hwmon/hwmonX`. If there are multiple directories (`hwmon0`, `hwmon1`, …), the directory corresponding to the fan can either be found by listing the contents and searching for present files as shown in the table above or looking for `pwmfan` in the `name` file:

```
target:~$ cat /sys/class/hwmon/hwmon1/name
pwmfan
```

When controlling the fan's behaviour through sensord, a configuration file called `fancontrol` will be present in `/etc/`. Otherwise, the kernel thermal driver may take control. See sysfs attribute structure for more detailed explanations.

7.19 TPM

The phyFLEX Libra RDK is equipped with a Trusted Platform Module (TPM) that provides hardware-based security functions.

Here are some useful examples to work with the TPM

Generate 4-byte random value with TPM2 tools:

```
target:~$ tpm2_getrandom --hex 4
```

Generate 4-byte random value with OpenSSL tools:

```
target:~$ openssl rand -engine libtpm2tss --hex 4
```

Generate RSA private key and validate its contents:

```
target:~$ openssl genrsa -engine libtpm2tss -out /tmp/priv_key 512
Engine "tpm2tss" set.
target:~$ openssl rsa -check -in /tmp/priv_key -noout
RSA key ok
target:~$ cat /tmp/priv_key
-----BEGIN PRIVATE KEY-----
MIIBVQIBADANBgkqhkiG9w0BAQEFAASCAT8wggE7AgEAAkEAxsvmcbxjwuKnYeuZ
2AVBmuLvYyqF/LpYOD3IB/v+YvEolxdGGmjiFLECU6xZ1j3+dIt4Y1zbcKS1OcWT
I8mbSwIDAQABAkBoy8wrYNhmP/1kzUJIclznPYJckGoZlFI1M7xjGSA9H1xDK6if
5g5CYCHPrbBp8e0mEokPRZoihxxzGTxGPiahAiEA/7OYMOpVZ5SD3YcRsWcQlkWI
MOSPUYg6vxvGG9xp4FcCIQDHB01RoHr+qXJwxIu3/3oQAUBI4ACJ4JRp0KelwhC0
LQIhANJzSvg/dak5l8pU55/99q3nbm7nPnnZSJiP0F6P62gjAiEAjf7qrfMF7Uyt
RkEjwbl2t5Z868FNARGGMVxZT4x+aF0CIGxlmP2pL8xFu1bWB282LSedqZUdQwel
Lxi7+svb2+uJ
-----END PRIVATE KEY-----
```

Note

Do NOT share your private RSA keys if you are going to use these keys for any security purposes.

Generate RSA public key and validate its contents:

```
target:~$ openssl rsa -in /tmp/test_key -pubout -out /tmp/pub_key
writing RSA key
target:~$ openssl pkey -inform PEM -pubin -in /tmp/pub_key -noout
target:~$ cat /tmp/pub_key
-----BEGIN PUBLIC KEY-----
MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAMbL5nG8Y8Lip2HrmdgFQZri72Mqhfy6
WDg9yAf7/mLxKJcXRhpo4hSxAlOsWdY9/nSLeGNc23CktTnFkyPJm0sCAwEAAQ==
-----END PUBLIC KEY-----
```

7.20 GPU

The i.MX 95 has a MALI G310 GPU.

As recommended by NXP in the i.MX Graphics User's Guide, when running benchmarks, set the /etc/environment on the target with the variable "WESTON_FORCE_RENDERER=1" and restart Weston with "systemctl".

This variable disables the use of hardware planes for compositing except for cursors. This avoids tearing and framerate drops caused by the lack of atomic Kernel Mode Setting support, ensuring more stable and consistent benchmark results.

The phytec-qt6demo-image has glmark2 as a GPU benchmark built in. To run the benchmark, execute:

```
target:~$ glmark2-es2-wayland
```

Note

For this benchmark to run properly, weston needs to be started successfully. For this display overlays might need to be loaded.

By default the benchmark is executed in 800x600 windowed mode. The benchmark can also be executed in fullscreen mode with the `--fullscreen` flag. Alternatively with the `--off-screen` flag the benchmark will not be shown on display.

7.21 Watchdog

The PHYTEC i.MX 95 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 JTAG

A JTAG debugger can be connected to phyFLEX Libra RDK. In this case, a J-Link is used. For many SoCs and their reference platforms there is a documentation available on the Segger website. It describes on how to connect and which features are supported currently: https://kb.segger.com/NXP_i.MX_95

7.22.1 Preparation

First download the J-Link Software (.deb archive) and install it.

```
host:~$ sudo apt install ./path/to/JLink_Linux_*.deb
```

Connect the J-Link debugger to the JTAG interface and to the host PC via USB. Ensure the connector to the phyFLEX Libra RDK is oriented correctly. The first pin is usually marked on the connector and the cable.

7.22.2 Connect to target

Boot the target into U-Boot and run `JLinkExe`. On the host, connect to the target using the `connect` command. When prompted to enter the device, either type in the device directly or type in `?` to get a Popup window, to list and search all devices and select the correct one. You will most likely find the exact model of the device printed on the SoC.

Hint

For USB-C cables with E-Marker chip, connection to the probe will not work. See https://kb.segger.com/USB-C_-_Connection_Issue

```
host:~$ JLinkExe
...
J-Link>connect
...
Device>?
Please specify target interface:
  J) JTAG (Default)
  S) SWD
  T) cJTAG
TIF>S
Specify target interface speed [kHz]. <Default>: 4000 kHz
Speed>
```

(continues on next page)

(continued from previous page)

```
...
J-Link>
```

Warning

There are some limitations:

- Connecting to A55 core only works in U-Boot not in Linux
- Booting Linux will fail when J-Link is connected
- J-Link needs to be reconnected after board reset
- Connecting to M7 core does not work, as there is currently not running any application on it
- Need to use SWD target protocol for connecting (at least for M33 core). This is also used in i.MX 95 example from Segger website.

7.22.3 J-Link basic functions

The JTAG debugger can pause and resume the CPU using the `halt` and `go` commands.

```
J-Link>halt
J-Link>go
```

The CPU core can be reset with the reset command.

```
J-Link>reset
J-Link>go
```

Reading from memory can be done with the `mem` command. The start address in memory needs to be specified as well as the length of memory to read. All the memory that is read needs to be accessible to the connected CPU Core.

```
J-Link>mem 0x90400000 0x100
```

Writing to memory can be done with different commands. The `Write1` command will write one byte into memory. There are also other variants of this command that will write more bytes. They can be listed with the `?` command.

```
J-Link>Write1 0x90400000 0xFF
```

There is also the `loadfile` command that will write a whole file raw into memory. If you do not specify `noreset` at the end, the CPU core will be reset after the file is loaded into memory.

Note

Writing larger files can take a while.

```
J-Link>loadfile file.bin 0x90400000 noreset
```

With the `verifybin` command, a file can be tested if it is present at a certain address in memory.

```
J-Link>verifybin file.bin 0x90400000
```

Warning

There are some limitations:

- According to Segger i.MX 95 documentation, reset command does not actually trigger a core reset.
- Writing to memory does not work from M33 core.

CHAPTER

EIGHT

NXP GOPOINT DEMO SUITE

NXP provides demos for their EVK SBCs. They are bundled in a demo suite called GoPoint. It is advertised as

> "GoPoint for i.MX Applications Processors is for users who are interested in showcasing the various features and capabilities of NXP provided SoCs. The demos included in this application are meant to be easy to run for users of all skill levels, making complex use cases accessible to anyone. Users need some knowledge when setting up equipment on Evaluation Kits (EVKs), such as changing Device Tree Blob (DTB) files."
>
> —GoPoint for i.MX Applications Processors User Guide

Since most of the demos require different accessory hardware to be connected to the SBC to function properly, the list of required hardware will be presented within each demo section.

8.1 ML Benchmark

> ML Benchmark tool allows to easily compare the performance of TensorFlow Lite models running on CPU (Cortex-A) and NPU, without the need to type in any command.
>
> —NXP ML benchmark tool

Note that NXP supplies instructions to run the demo as well. For completeness, references will be supplied.

8.1.1 Prerequisites

To be able to run the ML Benchmark demo application, you will need the following:

1. Yocto Project setup with the PHYTEC BSP being built
2. Ethernet cable, board connected to the internet
3. Display for any of the supported SoCs PHYTEC SBCs
4. Console connection to the SBC from host PC

Yocto Project

Modifications in the Yocto Project are necessary as PHYTEC BSPs do not have the GoPoint suite included by default. Add the following to your local.conf:

```
IMAGE_INSTALL:append = " packagegroup-imx-gopoint packagegroup-imx-ml"
```

Adding this causes the gopoint scripts/ui and the backend ml libraries to be installed, respectively. Build the phytec-qt6demo-image:

```
bitbake phytec-qt6demo-image
```

and flash the Image to the board.

Ethernet

Connect an Ethernet cable to the SBC or make otherwise sure that the SBC has access to the internet. Otherwise the demo application is unable to download a ml model and fails.

Display

Connect the Display accompanying the PHYTEC SBC to the SBC. You may also use your own display, however different hardware and/or software may be required. The results for this demo are put out as a graphical UI and when weston is unable to start the demo will not start, either.

Note

The accompanying display supports touch. In that case no mouse is necessary. When not using a touch display, a mouse is necessary as you need to click on gui elements.

Console

Connect a USB cable from your host PC to the respective port of the SBC.

8.1.2 Running the demo

Boot the board. Ensure the display is working. When using your own display, ensure you have the correct dtbo applied and weston starts. Connect to the SBC via debug console and execute:

```
gopoint tui
```

This will prompt you for a selection of different demos. Use the arrow keys to select the ML Benchmark demo and press Enter. On the display, a TFLite Benchmarking box should appear. The bottom text within the box should say `Models are ready for inference`. Click/press on `RUN BENCHMARKS!`. Sometimes the box may disappear, rerun the ML Benchmark in the terminal. NXP explains this and other issues here.