

---

# **phyCORE-i.MX 93 BSP Manual**

## **PD26.1.0**

**PHYTEC Messtechnik GmbH**

**Apr 27, 2026**



# CONTENTS

<b>1</b>	<b>Supported Hardware</b>	<b>3</b>
1.1	phyBOARD-Segin i.MX 91/93 Components . . . . .	3
1.2	phyBOARD-Nash i.MX 93 Components . . . . .	3
<b>2</b>	<b>Getting Started</b>	<b>7</b>
2.1	Get the Image . . . . .	7
2.2	Write the Image to SD Card . . . . .	7
2.3	First Start-up . . . . .	10
<b>3</b>	<b>Building the BSP</b>	<b>11</b>
3.1	Basic Set-Up . . . . .	11
3.2	Get the BSP . . . . .	11
<b>4</b>	<b>Installing the OS</b>	<b>15</b>
4.1	Bootmode Switch (S3) . . . . .	15
4.2	Flash eMMC . . . . .	15
4.3	RAUC . . . . .	20
<b>5</b>	<b>Development</b>	<b>21</b>
5.1	Standalone Build preparation . . . . .	21
5.2	U-Boot standalone build . . . . .	22
5.3	Kernel standalone build . . . . .	24
5.4	Working with UUU . . . . .	26
5.5	Host Network Preparation . . . . .	28
5.6	Booting the Kernel from a Network . . . . .	30
5.7	Accessing the Development states . . . . .	31
5.8	Format SD card . . . . .	32
5.9	Ampliphy-boot . . . . .	38
5.10	Working with FIT images . . . . .	40
<b>6</b>	<b>Device Tree (DT)</b>	<b>43</b>
6.1	Introduction . . . . .	43
6.2	PHYTEC i.MX 93 BSP Device Tree Concept . . . . .	43
<b>7</b>	<b>Accessing Peripherals</b>	<b>47</b>
7.1	i.MX 93 Pin Muxing . . . . .	47
7.2	Ethernet . . . . .	48
7.3	WLAN/Bluetooth . . . . .	50
7.4	SD card . . . . .	53
7.5	e.MMC Devices . . . . .	54
7.6	GPIOs . . . . .	61

7.7	ADC	64
7.8	LEDs	65
7.9	I <sup>2</sup> C Bus	65
7.10	EEPROM	66
7.11	RTC	67
7.12	USB Host Controller	69
7.13	USB OTG	70
7.14	RS232/RS485	72
7.15	CAN FD	73
7.16	Audio on phyBOARD-Segin i.MX 91/93	75
7.17	Audio on phyBOARD-Nash i.MX 93	76
7.18	Video	79
7.19	Display	79
7.20	Power Management	81
7.21	Thermal Management	83
7.22	Watchdog	84
7.23	Power Key	85
7.24	PXP	86
7.25	On-Chip OTP Controller (OCOTP_CTRL) - eFuses	86
7.26	TPM	87
7.27	JTAG	88
<b>8</b>	<b>i.MX 93 M33 Core</b>	<b>91</b>
8.1	Running M33 Core Examples	91
8.2	ARM Ethos-U NPU	93

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.MX91-PD26.1.0	Major	Walnascar	2026/03/13	Released
BSP-Yocto-NXP-i.MX93-PD26.1.0	Major	Walnascar	2026/03/13	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 **phyBOARD-Segin i.MX 93** and **phyBOARD-Nash i.MX 93 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/phycore-imx-91-93/#downloads> of our product.

## SUPPORTED HARDWARE

On our web page, you can see all supported Machines with the available Article Numbers for this release: BSP-Yocto-NXP-i.MX91-PD26.1.0 see [download](#), or BSP-Yocto-NXP-i.MX93-PD26.1.0 see [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.

### Tip

Console examples in this BSP manual only focus on phyBOARD-Segin i.MX 93. Similar commands can also be executed for/on phyBOARD-Nash i.MX 93

### 1.1 phyBOARD-Segin i.MX 91/93 Components

### 1.2 phyBOARD-Nash i.MX 93 Components

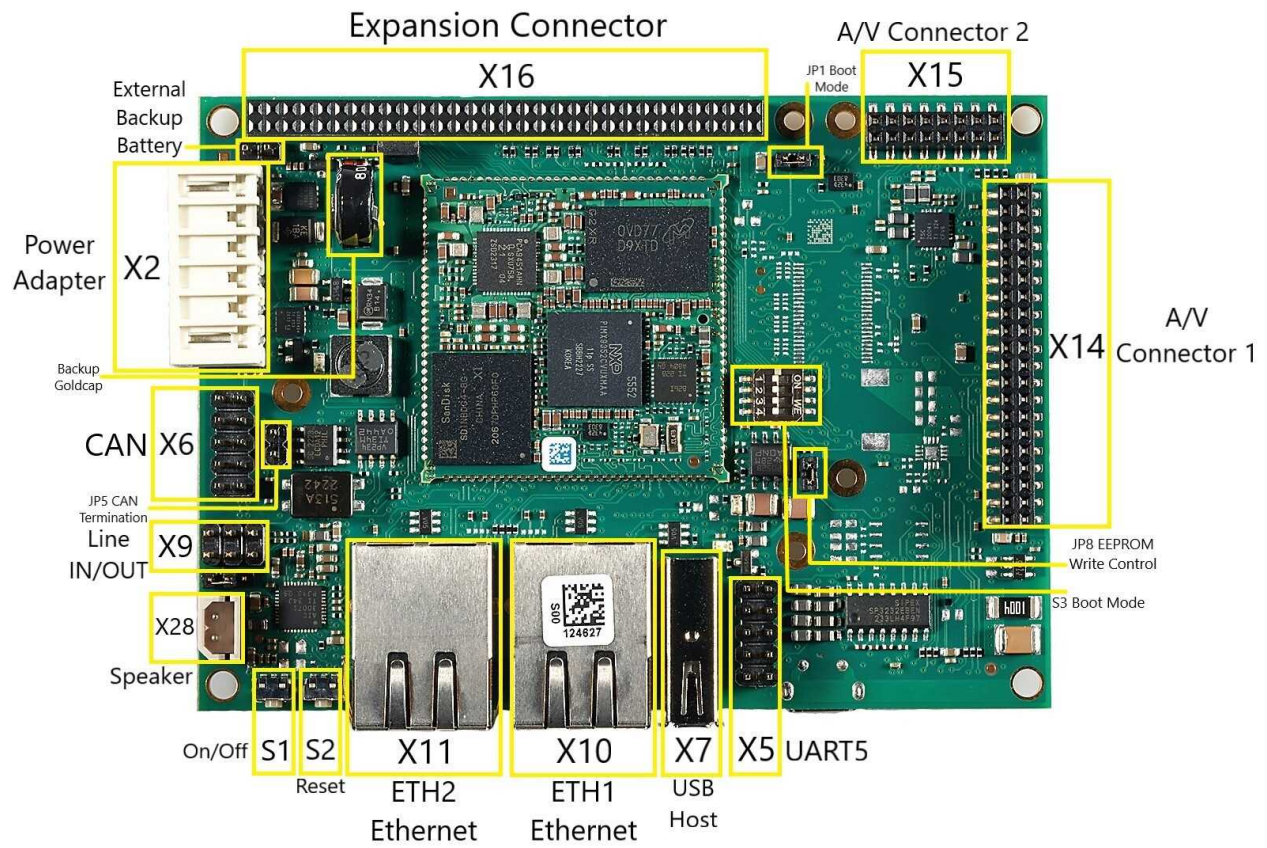


Fig. 1: phyBOARD-Segin i.MX 91/93 Components (top)

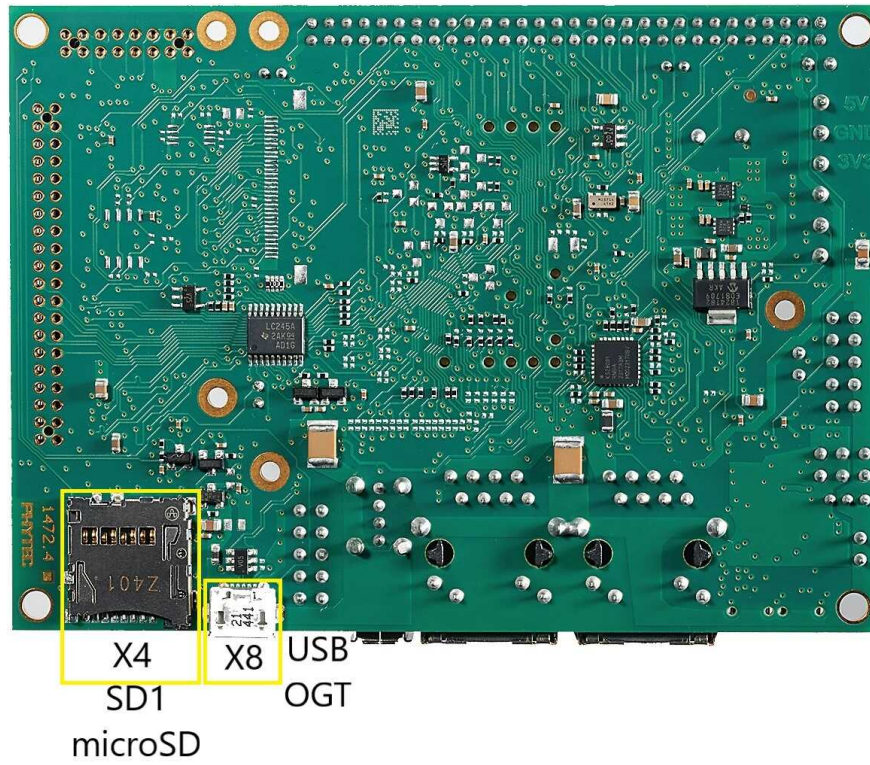


Fig. 2: phyBOARD-Segin i.MX 91/93 Components (bottom)

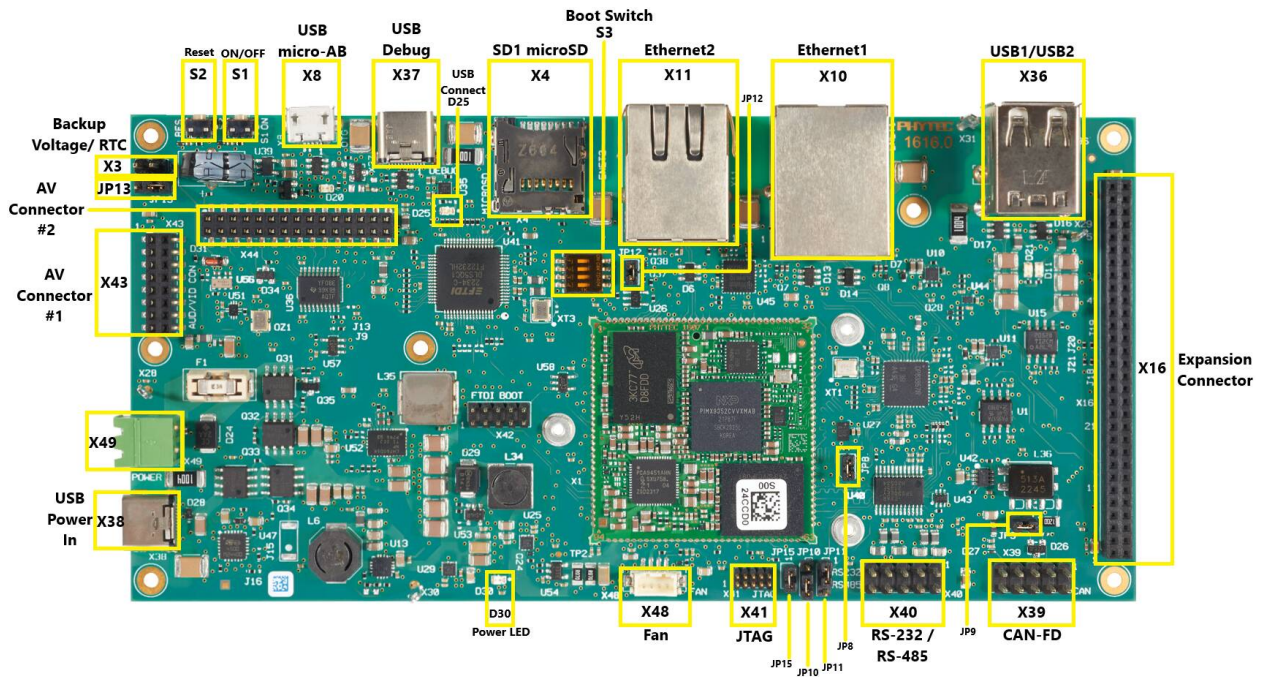


Fig. 3: phyBOARD-Nash i.MX 93 Components (top)

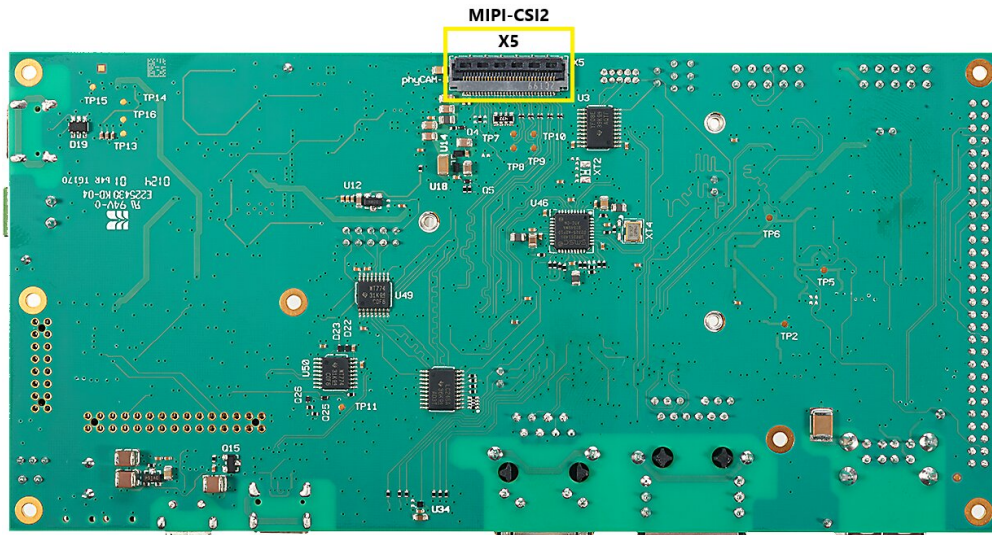


Fig. 4: phyBOARD-Nash i.MX 93 Components (bottom)

## GETTING STARTED

The **phyBOARD-Segin i.MX 93** and **phyBOARD-Nash i.MX 93 Kit** is shipped with a pre-flashed SD card. It contains the phytec-qt6demo-image and can be used directly as a boot source. The eMMC is programmed with only a U-Boot by default. You can get all sources from the [i.MX 91](#) or [i.MX 93](#) 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 eMMC. Most notably using simple, sequential writing with the Linux command line tool `dd`. An alternative way is to use PHYTEC's system initialization program called [partup](#), which makes it especially easy to format more complex systems. You can get [prebuilt Linux binaries of partup](#) from its release page. Also read [partup's README](#) for installation instructions.

### 2.1 Get the Image

The image contains all necessary files and makes sure partitions and any raw data are correctly written. Both the `partup` package and the WIC image, which can be flashed using `dd`, can be downloaded from our [i.MX 91](#) or [i.MX 93](#) 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 https://download.phytec.de/Software/Linux/BSP-Yocto-i.MX93/BSP-Yocto-NXP-i.MX93-
↳ PD26.1.0/images/ampliphy-vendor/phyboard-segin-imx93-2/phytec-qt6demo-image-phyboard-segin-
↳ imx93-2.rootfs.partup
host:~$ wget https://download.phytec.de/Software/Linux/BSP-Yocto-i.MX93/BSP-Yocto-NXP-i.MX93-
↳ PD26.1.0/images/ampliphy-vendor/phyboard-segin-imx93-2/phytec-qt6demo-image-phyboard-segin-
↳ imx93-2.rootfs.wic.xz
```

#### Note

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

### 2.2 Write the Image to SD Card

#### Warning

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

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

## 2.2.1 Finding the Correct Device

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

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

```
host:~$ lsblk
```

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

```
host:~$ lsblk
```

3. Compare the two outputs to find the new device names listed in the second output. These are the device names of the SD card (device and partitions if the SD card was formatted).
4. In order to verify the device names being found, execute the command `sudo dmesg`. Within the last lines of its output, you should also find the device names, e.g. `/dev/sde` or `/dev/mmcblk0` (depending on your system).

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

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

Unmount all those partitions, e.g.:

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

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

## 2.2.2 Using `bmptool`

One way to prepare an SD card is using `bmptool`. 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. `bmptool` 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:~$ bmptool copy phytec-qt6demo-image-phyboard-segin-imx93-2?(.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 `bmptool` knows which blocks to write and which to skip.

### Warning

`bmptool` 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-phyboard-segin-imx93-2?(.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 `bmptool` 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-phyboard-segin-imx93-2?(.rootfs).wic of=/dev/<your_  
↪device> bs=1M conv=fsync status=progress
```

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

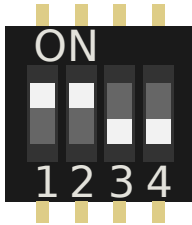
```
host:~$ xzcat phytec-qt6demo-image-phyboard-segin-imx93-2?(.rootfs).wic.xz | sudo dd of=/dev/  
↪<your_device> bs=1M conv=fsync status=progress
```

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 (S3)* needs to be set to the following position:



- Insert the SD card
- Connect the targets debug console with your host. Use UART1 console on PEB-EVAL-01 for **phyBOARD-Segin** and X-37 USB-C debug for **phyBOARD-Nash**.
- Power up the board
- Open serial/usb port with 115200 baud and 8N1 (you should see u-boot/linux start on the console)

## BUILDING THE BSP

This section will guide you through the general build process of the i.MX 93 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 Phytex 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 [i.MX 91 or i.MX 93 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 [i.MX 91 BSP](#) or [i.MX 93 BSP](#).

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

```
host:~/yocto$ DISTRO=ampliphy-vendor MACHINE=phyboard-segin-imx93-2 ./phyLinux init -p
↪ i.mx93 -r BSP-Yocto-NXP-i.MX93-PD26.1.0
```

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 phyboard-segin-imx93-2 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)

- **bl31-imx93.bin** or **bl31-imx91.bin**: ARM Trusted Firmware binary
- **lpddr4\_dmem\_1d\_v202201.bin**, **lpddr4\_dmem\_2d\_v202201.bin**,  
**lpddr4\_imem\_1d\_v202201.bin**, **lpddr4\_imem\_2d\_v202201.bin**: DDR PHY firmware images
- **imx-boot**: Bootloader build by imx-mkimage which includes SPL, U-Boot, ARM Trusted Firmware and DDR firmware. This is the final bootloader image which is bootable.
- **fitImage**: Linux kernel FIT image
- **fitImage.its**
- **Image**: Linux kernel image
- **Image.config**: Kernel configuration
- **imx93-phyboard-\*.dtb** or **imx91-phyboard-\*.dtb**: Kernel device tree file
- **imx93-phy\*.dtbo** or **imx91-phy\*.dtbo**: Kernel device tree overlay files
- **phytec-\*.tar.gz**: Root file system, of bitbake-image that was built.
  - **phytec-qt6demo-image-phyboard-\*-imx9\*-.tar.gz**: when bitbake-build was processed for phytec-qt6demo-image
  - **phytec-headless-image-phyboard-\*-imx9\*-.tar.gz**: when bitbake-build was processed for phytec-headless-image
- **phytec-\*.rootfs.wic.xz**: Compressed bootable SD card image of bitbake-image that was built. Includes bootloader, DTBs, Kernel and Root file system.
  - **phytec-qt6demo-image-phyboard-\*-imx9\*-.rootfs.wic.xz**: when bitbake-build was processed for phytec-qt6demo-image
  - **phytec-headless-image-phyboard-\*-imx9\*-.rootfs.wic.xz**: when bitbake-build was processed for phytec-headless-image
- **imx93-11x11-evk\_m33\_\*.bin**, binaries of demo applications for the Cortex-M33 MCU; can be manually loaded and started with U-Boot or Linux; are not part of builds for i.MX 91;



## INSTALLING THE OS

### 4.1 Bootmode Switch (S3)

#### Tip

Hardware revision baseboard:

- phyBOARD-Segin i.MX 91/93: 1472.5
- phyBOARD-Nash i.MX 93: 1616.0, 1616.1, 1616.2

The phyBOARD-Segin/Nash i.MX 91/93 features a boot switch with four individually switchable ports to select the phyCORE-i.MX 93 default bootsource.



Fig. 1: eMMC

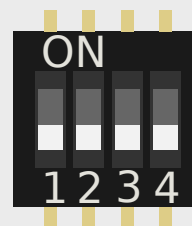


Fig. 2: Internal Fuses

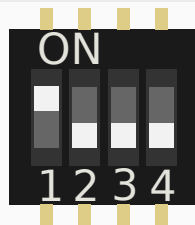


Fig. 3: USB Serial Download

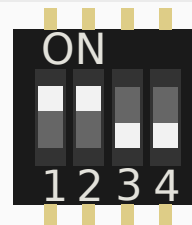


Fig. 4: SD Card

### 4.2 Flash eMMC

To boot from eMMC, make sure that the BSP image is flashed correctly to the eMMC and the *bootmode switch (S3)* is set to eMMC.

**Warning**

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

```
target:~$ blkid
```

can be run to inspect whether the current setup is affected. If mmcblk0p1 and mmcblk1p1 have an identical UUID, the setup is affected.

**Note**

**Filenames WIC and partup:** Command lines listed in this chapter contain exemplary file names for WIC images and partup packages. The actual file name is always a combination of the machine and the image you want to use. The supported machines are listed on the webpage of our i.MX 91 BSP or i.MX 93 BSP.

## 4.2.1 Flash eMMC from SD Card

If there is no network available, you can update the eMMC from SD card. For that, you only need a ready-to-use image file (\*.wic) located on the SD card. Because the image file is quite large, you have to enlarge your SD card to use its full space (if it was not enlarged before). To enlarge your SD card, see Resizing ext4 Root Filesystem.

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

### Flash eMMC from SD card in Linux on Target

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

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

```
target:~$ ls
phytec-qt6demo-image-phyboard-segin-imx93-2.rootfs.partup
phytec-qt6demo-image-phyboard-segin-imx93-2.rootfs.wic.xz
```

- Show list of available MMC devices:

```
target:~$ ls /dev | grep mmc
mmcblk1
mmcblk1p1
mmcblk1p2
mmcblk0
mmcblk0boot0
mmcblk0boot1
mmcblk0p1
mmcblk0p2
mmcblk0rpb
```

- The eMMC device can be recognized by the fact that it contains two boot partitions: (mmcblk0boot0; mmcblk0boot1)
- Write the image to the phyCORE-i.MX 93 eMMC (/dev/mmcblk0 **without** partition) using partup:

```
target:~$ partup install phytec-qt6demo-image-phyboard-segin-imx93-2.rootfs.partup /dev/
↳mmcblk0
```

### Tip

Using `partup` is highly recommended since it is easier to use and has the advantage of using the full capacity of the eMMC device, adjusting partitions accordingly.

### Note

Alternatively, `dd` may be used instead.

For uncompressed WIC images (\*.wic):

```
target:~$ dd if=phytec-qt6demo-image-phyboard-segin-imx93-2.rootfs.wic of=/dev/mmcblk0
↳bs=1M conv=fsync status=progress
```

For compressed WIC images (\*.wic.xz):

```
target:~$ zstdcat phytec-qt6demo-image-phyboard-segin-imx93-2.rootfs.wic.xz | sudo dd
↳of=/dev/mmcblk0 bs=1M conv=fsync status=progress
```

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

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

### Warning

Before this will work, you need to configure the *bootmode switch (S3)* to eMMC.

## 4.2.2 Flash eMMC from Network

i.MX 93 boards have an Ethernet connector and can be updated over a network. Be sure to set up the development host correctly. The IP needs to be set to 192.168.3.10, the netmask to 255.255.255.0, and a TFTP server needs to be available. From a high-level point of view, an eMMC device is like an SD card. Therefore, it is possible to flash the **WIC image** (<name>.wic) from the Yocto build system directly to the eMMC. The image contains the bootloader, kernel, device tree, device tree overlays, and root file system.

### Note

Some PHYTECs BSPs produce compressed .wic.xz images. In this case, the compressed image must first be uncompressed.

```
host:~$ unxz phytec-qt6demo-image-phyboard-segin-imx93-2.rootfs.wic.xz
```

### Flash eMMC via Network in Linux on Target

You can update the eMMC from your target.

**Tip**

A working network is necessary! [Setup Network Host](#)

Take an uncompressed image on the host and send it with ssh through the network to the eMMC of the target with a one-line command:

```
target:~$ ssh <USER>@192.168.3.10 "dd if=<path_to_file>/phytec-qt6demo-image-phyboard-segin-
↳imx93-2.rootfs.wic" | dd of=/dev/mmcblk0 bs=1M conv=fsync status=progress
```

**Flash eMMC via Network in Linux on Host**

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

**Tip**

A working network is necessary! [Setup Network Host](#)

Show your available image files on the host:

```
host:~$ ls
phytec-qt6demo-image-phyboard-segin-imx93-2.rootfs.wic
```

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

```
host:~$ dd if=phytec-qt6demo-image-phyboard-segin-imx93-2.rootfs.wic bs=1M status=progress | ssh
↳root@192.168.3.11 "dd of=/dev/mmcblk0 conv=fsync"
```

**4.2.3 Flash eMMC U-Boot image via Network from running U-Boot**

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

**Tip**

A working network is necessary! [Setup Network Host](#)

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

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

## 4.2.4 Flash eMMC from USB

### Flash eMMC from USB in Linux

These steps will show how to flash the eMMC on Linux with a USB stick. You only need a complete image saved on the USB stick and a bootable WIC image. (e.g. phytec-qt6demo-image-phyboard-segin-imx93-2.rootfs.wic). Set the *bootmode switch (S3)* 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, e.g. like:

```
target:~$ cd /mnt
target:~$ ls
phytec-qt6demo-image-phyboard-segin-imx93-2.rootfs.wic
```

- Show list of available MMC devices:

```
target:~$ ls /dev | grep mmc
mmcblk1
mmcblk1p1
mmcblk1p2
mmcblk0
mmcblk0boot0
mmcblk0boot1
mmcblk0p1
mmcblk0p2
mmcblk0rpb
```

- The eMMC device can be recognized by the fact that it contains two boot partitions: (**mmcblk0boot0**; **mmcblk0boot1**)
- Write the image to the phyCORE-i.MX 93 eMMC (/dev/mmcblk0 without partition):

```
target:~$ dd if=phytec-qt6demo-image-phyboard-segin-imx93-2.rootfs.wic of=/dev/mmcblk0
↪bs=1M conv=fsync status=progress
```

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

#### Tip

Before this will work, you need to configure the *bootmode switch (S3)* to **eMMC**.

## 4.3 RAUC

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

## 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:

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

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

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

- Our i.MX 93 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-imx_*.bb
```

#### 5.1.2 Get the SDK

You can download the SDK from the [i.MX 91 SDK](#) or [i.MX 93 SDK](#) downloads page, or build it yourself with Yocto:

- Move to the Yocto build directory:

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

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

#### 5.1.3 Install the SDK

- Set correct permissions and install the SDK:

```

host:~$ chmod +x phytec-ampliphy-vendor-glibc-x86_64-phytec-qt6demo-image-cortexa55-
↳toolchain-5.2.2.sh
host:~$ ./phytec-ampliphy-vendor-glibc-x86_64-phytec-qt6demo-image-cortexa55-toolchain-5.2.
↳2.sh
=====
Enter target directory for SDK (default: /opt/ampliphy-vendor/5.2.2):
You are about to install the SDK to "/opt/ampliphy-vendor/5.2.2". 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/5.2.2/environment-setup-cortexa55-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 git://git.phytec.de/u-boot-imx
```

- To get the correct *U-Boot tag* you need to take a look at our release notes, which can be found here: [release notes](#)
- The **tag** needed at this release is called v2025.04-2.1.0-phy9
- Check out the needed *U-Boot tag*:

```

host:~$ cd ~/u-boot-imx/
host:~/u-boot-imx$ git fetch --all --tags
host:~/u-boot-imx$ git checkout tags/v2025.04-2.1.0-phy9

```

- Technically, you can now build the U-Boot, but practically there are some further steps necessary:
  - Create your own development branch:

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

#### Note

You can name your development branch as you like, this is just an example.

- Set up a build environment:

```
host:~/u-boot-imx$ source /opt/ampliphy-vendor/5.2.2/environment-setup-cortexa55-phytec-
↪linux
```

## 5.2.2 Get the needed binaries

To build the imx-boot, you need to **gather** these **files** for later use with **imx-mkimage** tool:

- **ARM Trusted firmware binary** (*mkimage* tool compatible format **bl31.bin**): bl31-imx\*.bin
- **OPTEE image** (optional): tee.bin
- **DDR firmware files** (*mkimage* tool compatible format **lpddr4\_[i,d]mem\_\*d\_\*.bin**):  
lpddr4\_dmem\_1d\_\*.bin, lpddr4\_dmem\_2d\_\*.bin, lpddr4\_imem\_1d\_\*.bin,  
lpddr4\_imem\_2d\_\*.bin
- **Container image**: mx\*-ahab-container.img

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

Or you can download the files here: for [i.MX 91](#) or for [i.MX 93](#)

### Warning

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

## 5.2.3 Build the bootloader

- Build u-boot:

```
host:~/u-boot-imx$ make <defconfig>
host:~/u-boot-imx$ make
host:~/u-boot-imx$ cd ..
```

### Note

In command above replace <defconfig> with imx93-phycore\_defconfig or imx91-phycore\_defconfig

### Build final flash.bin with imx-mkimage

- Get imx-mkimage:

```
host:~$ git clone https://github.com/nxp-imx/imx-mkimage
host:~$ cd imx-mkimage/
host:~/imx-mkimage$ git checkout tags/lf-6.12.34_2.1.0
```

- Copy firmware binaries into imx-mkimage

```
host:~/imx-mkimage$ cp ../artefacts/bl31-imx*.bin ./iMX9/bl31.bin
host:~/imx-mkimage$ cp \
    ../artefacts/lpddr4_* \
    ../artefacts/mx*-ahab-container.img \
```

(continues on next page)

(continued from previous page)

```
../artefacts/tee.bin |
./iMX9/
```

- Copy u-boot binaries and DTB into imx-mkimage

```
host:~/imx-mkimage$ cp ../u-boot-imx/spl/u-boot-spl.bin ../u-boot-imx/u-boot.bin ./iMX9/
host:~/imx-mkimage$ cp ../u-boot-imx/arch/arm/dts/<dtb> ./iMX9/
```

**Note**

In command above replace <dtb> with imx93-phyboard-segin.dtb

- Build final flash.bin binary

```
host:~/imx-mkimage$ make SOC=iMX9 REV=A1 flash_singleboot
```

## 5.2.4 Flash the bootloader to a block device

The flash.bin can be found at imx-mkimage/iMX9/ directory and now can be flashed. A chip-specific offset is needed:

SoC	Offset User Area	Offset Boot Partition	eMMC Device
i.MX 93	32 kiB	0 kiB	/dev/mmcblk0

E.g. flash SD card:

```
host:~/imx-mkimage$ sudo dd if=./iMX9/flash.bin of=<sd-card> bs=1024 seek=32 conv=fsync
```

**Note**

In the command above, replace <sd-card> with your sd-card device name. For more information on how to find the device name, see the section Finding the Correct Device in Getting Started.

**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 be 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 git://github.com/phytec/linux-phytec-imx.git
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/5.2.2/environment-setup-cortexa55-
↳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/imx93-phyboard-segin.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. imx93-phyboard-segin.dtb):

```
host:~/linux-phytec-imx$ make freescale/imx93-phyboard-segin.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: for *i.MX 91* or for *i.MX 93*

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/phyboard-segin-imx93-2/`. For flashing a `wic` image to eMMC, you will also need `phytec-qt6demo-image-phyboard-segin-imx93-2.rootfs.wic`.

### 5.4.3 Prepare Target

Set the *bootmode switch (S3)* to **USB Serial Download**. Also, connect USB port *X8 (USB micro/OTG 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 UART1 console on PEB-EVAL-01 for **phyBOARD-Segin** and X-37 USB-C debug for **phyBOARD-Nash**, 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-phyboard-segin-imx93-2.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-phyboard-segin-imx93-2.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": {
```

(continues on next page)

(continued from previous page)

```

    "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-phyboard-segin-imx93-2.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 Accessing the Development states

### 5.7.1 Development state of current release

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

```
host:~$ ./phyLinux init -p imx93 -r |yocto-manifestname-y|
```

This will initialize a BSP that will track the latest development state of the current release (BSP-Yocto-NXP-i.MX93-PD26.1.0). From now on *repo sync* in this folder will pull all the latest changes from our Git repositories:

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

## 5.7.2 Development state of upcoming release

Also development states of upcoming releases can be accessed this way. For this execute the following command and look for a release with a higher PDXX.Y number than the latest one (BSP-Yocto-NXP-i.MX93-PD26.1.0) and *.y* at the end:

```
host:~$ ./phyLinux init -p imx93
```

## 5.8 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.

### 5.8.1 Gparted

- Get GParted:

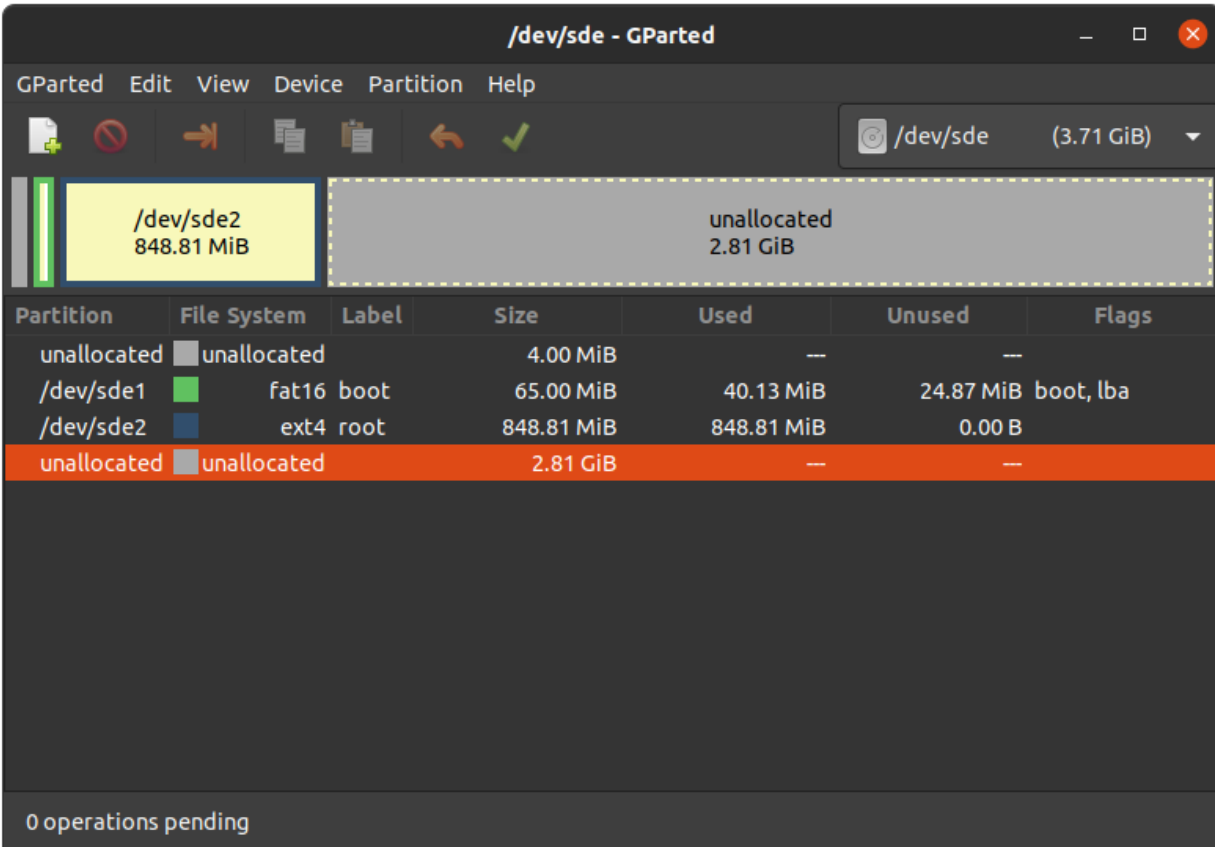
```
host:~$ sudo apt install gparted
```

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

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

- Unmount all SD card partitions.
- Launch GParted:

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

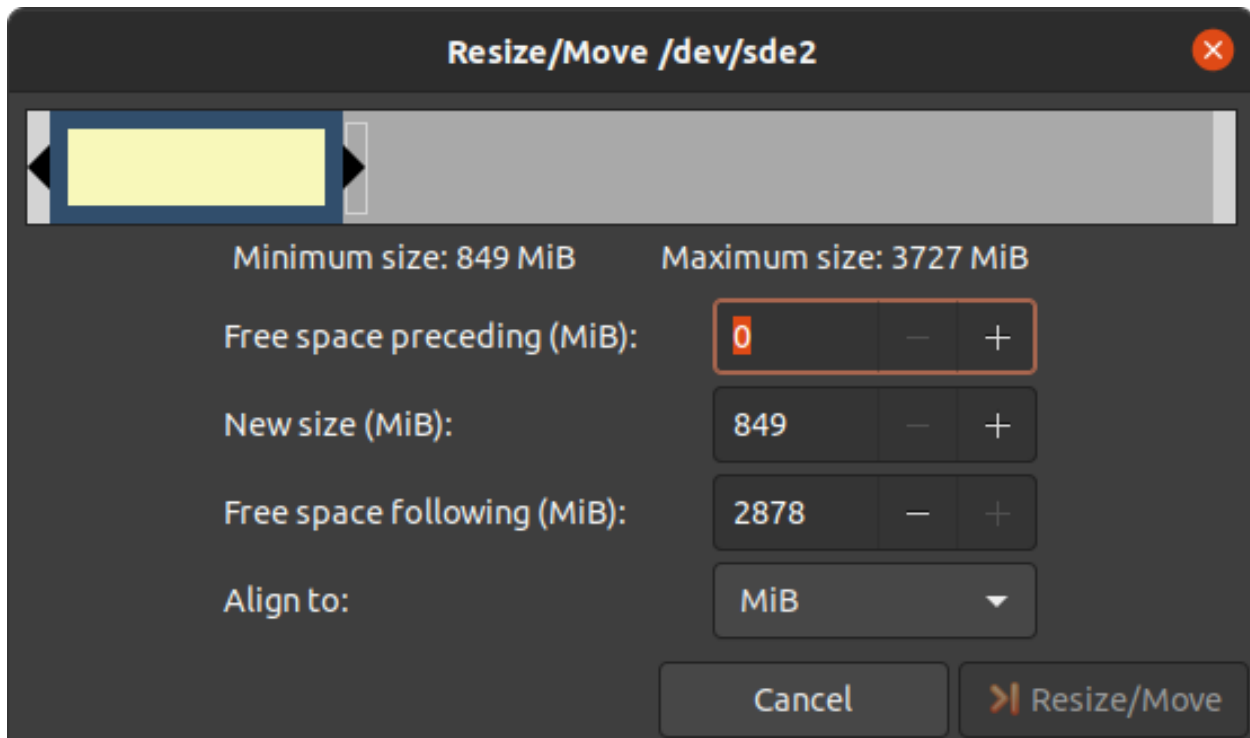
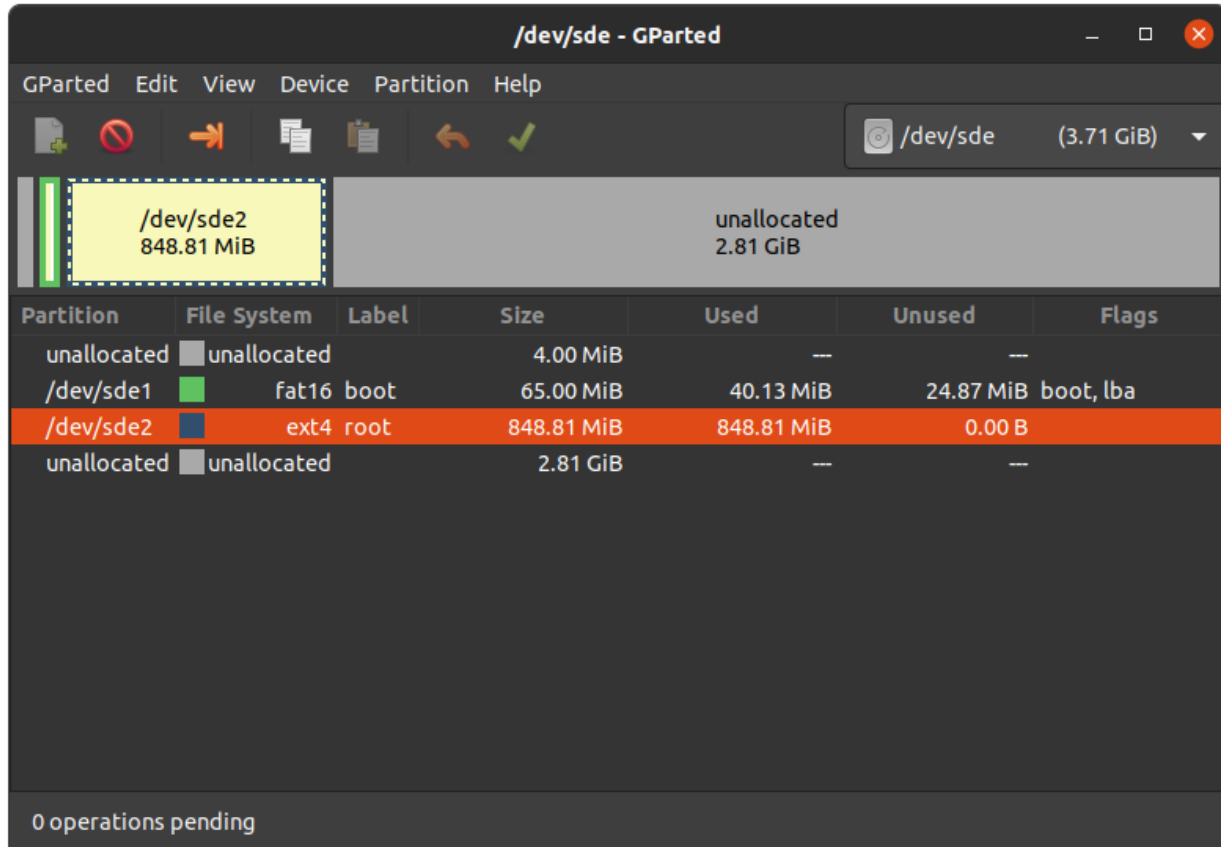


## Expand rootfs

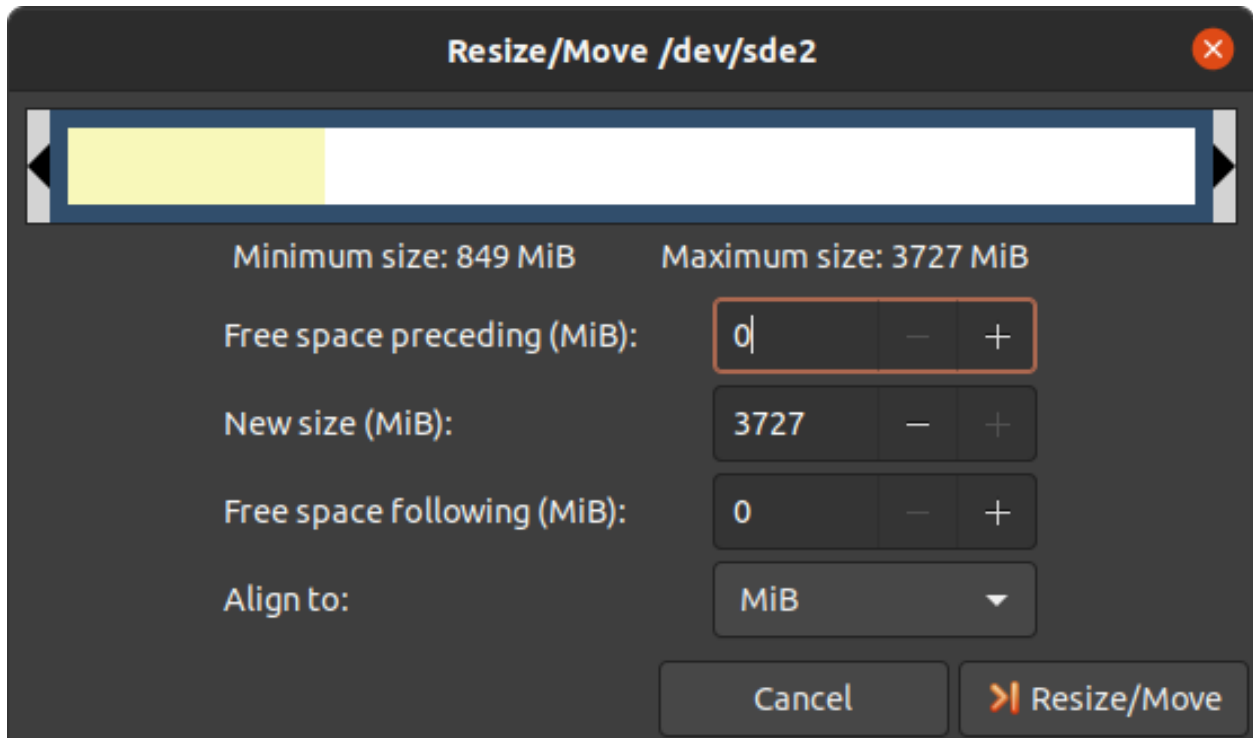
### Warning

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

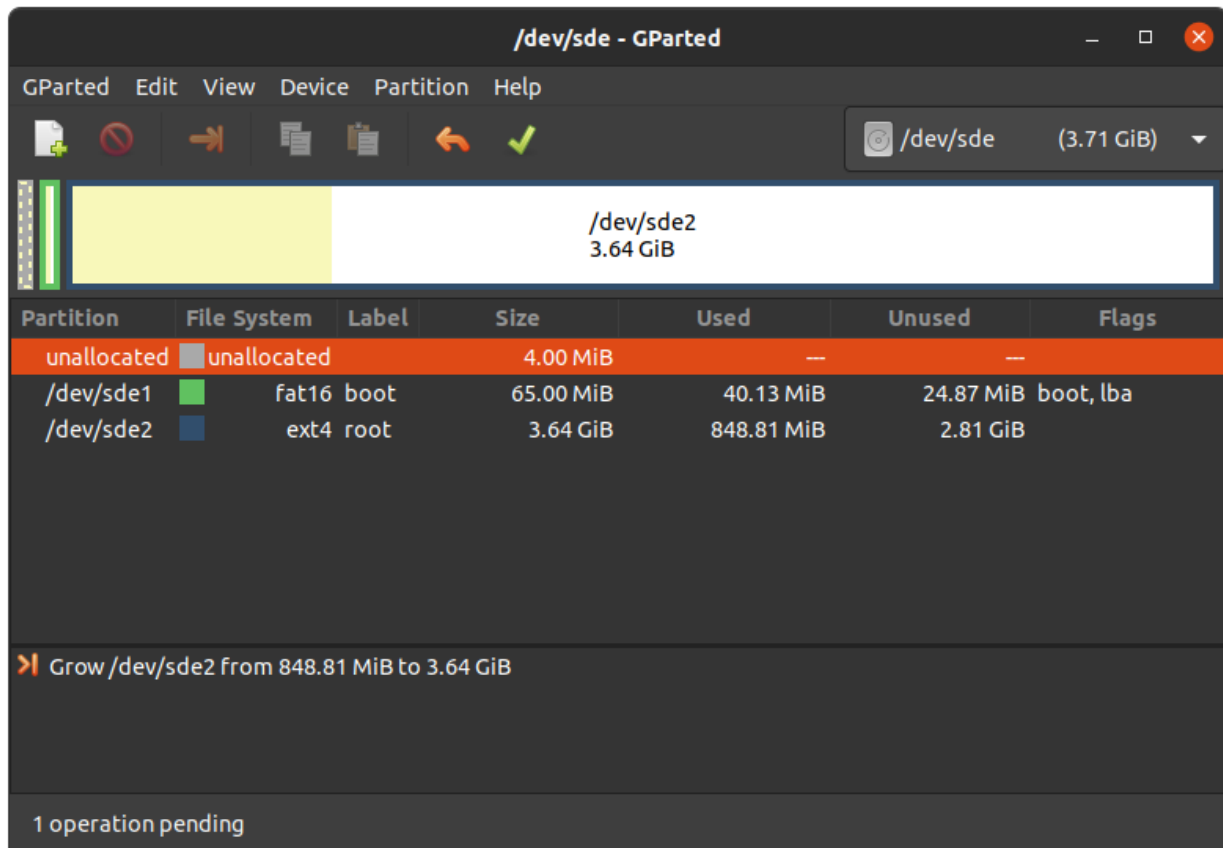
- Choose your SD card device at the drop-down menu on the top right
- Choose the ext4 root partition and click on resize:



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



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



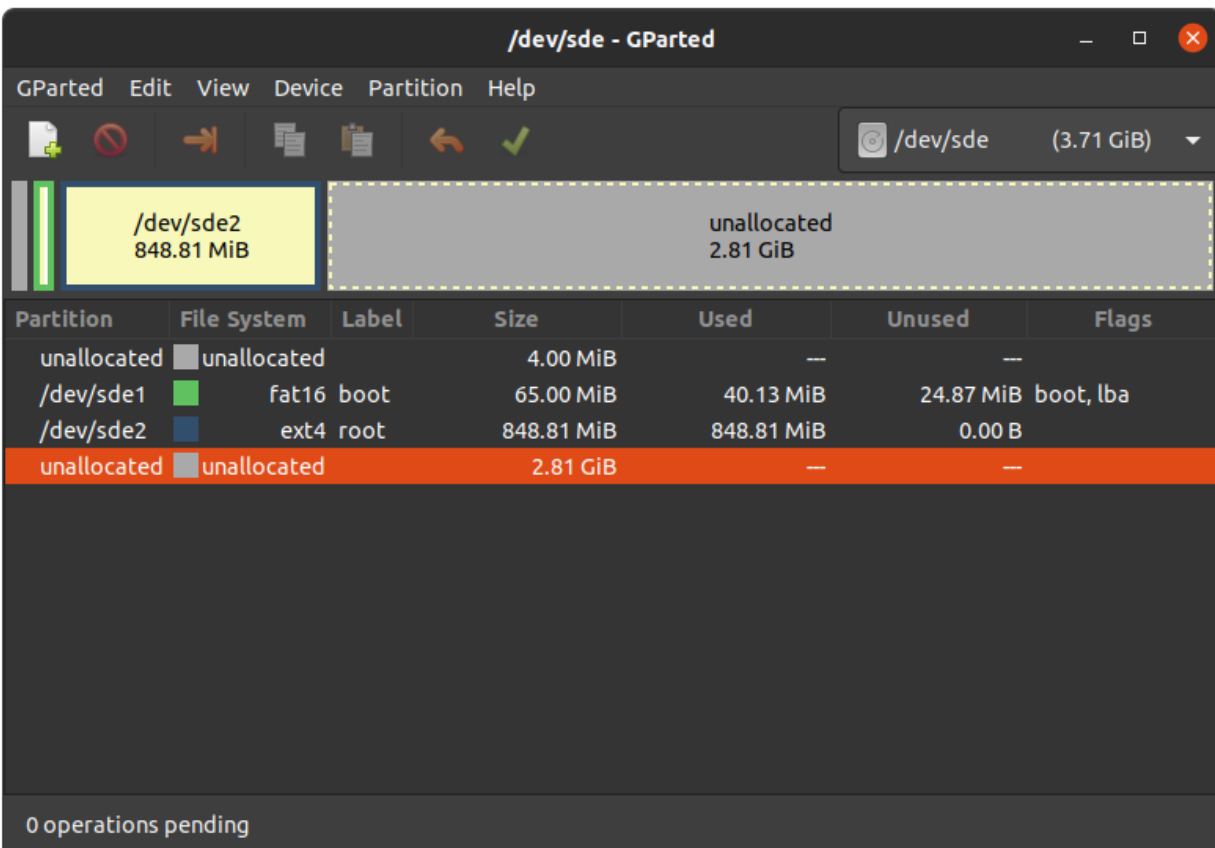
- To apply your changes, press the green tick.

- Now you can mount the root partition and copy e.g. the phytec-qt6demo-image-phyboard-segin-imx93-2.wic image to it. Then unmount it again:

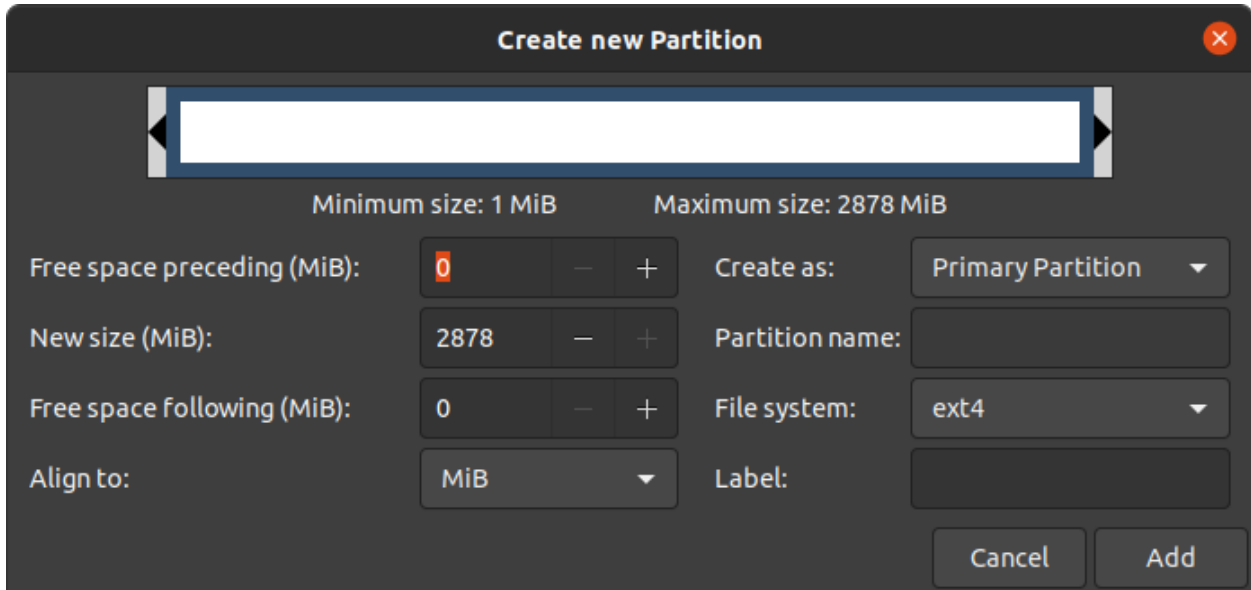
```
host:~$ sudo cp phytec-qt6demo-image-phyboard-segin-imx93-2.wic /mnt/ ; sync
host:~$ umount /mnt
```

### Create the Third Partition

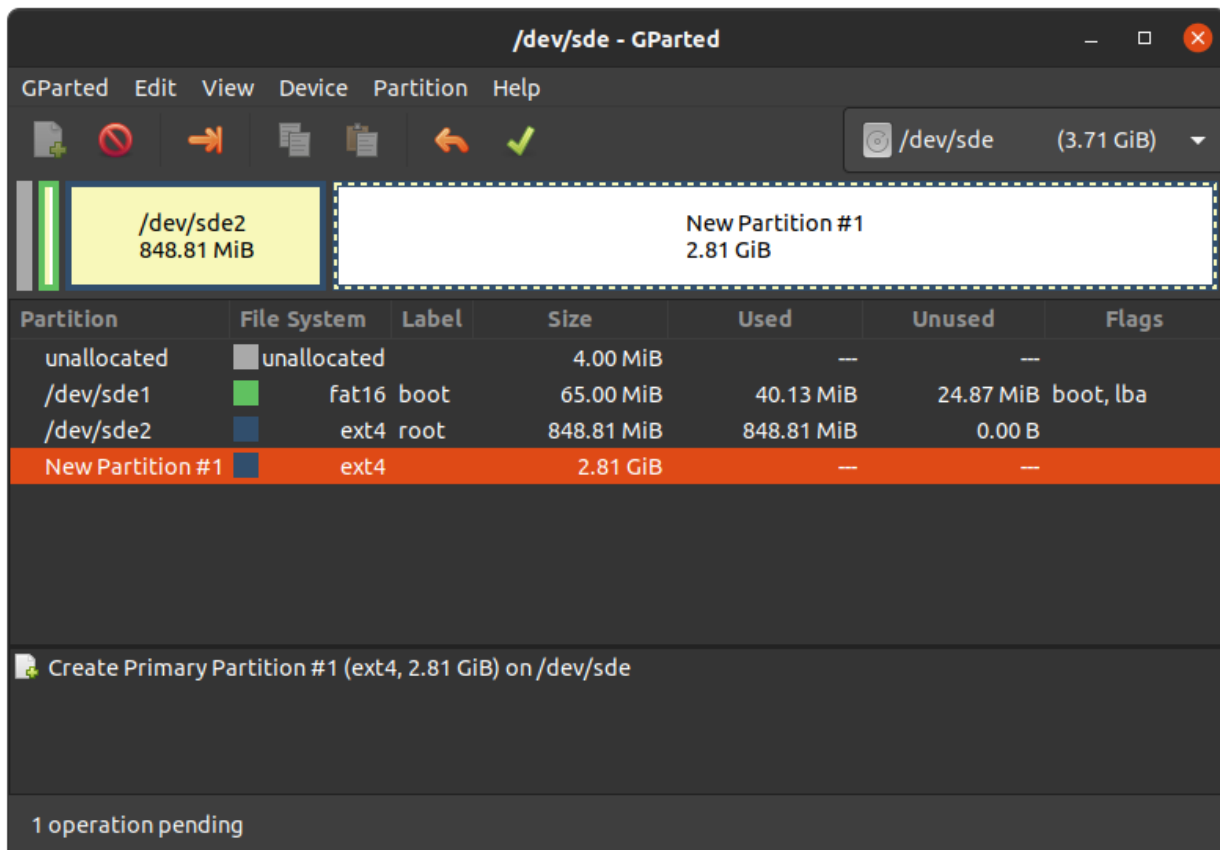
- Choose your SD card device at the drop-down menu on the top right



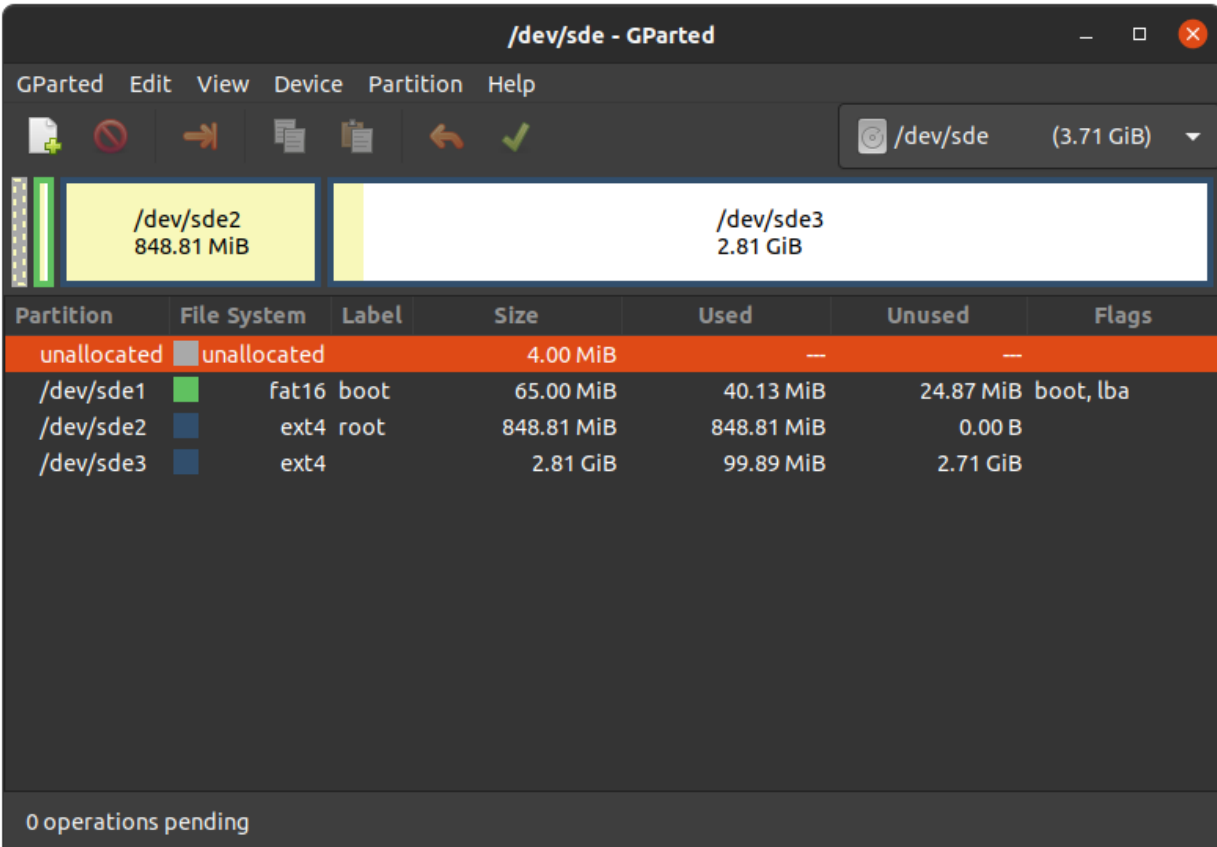
- Choose the bigger unallocated area and press “New”:



- Click “Add”



- Confirm your changes by pressing the green tick.



- Now you can mount the new partition and copy e.g. phytec-qt6demo-image-phyboard-segin-imx93-2.wic image to it. Then unmount it again:

```
host:~$ sudo mount /dev/sde3 /mnt
host:~$ sudo cp phytec-qt6demo-image-phyboard-segin-imx93-2.wic /mnt/ ; sync
host:~$ umount /mnt
```

## 5.9 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 **phyBOARD-Segin i.MX 93** and **phyBOARD-Nash i.MX 93 Kit** supports the following bootscripts:

```
mmc_boot_fit
net_boot_fit
```

### 5.9.1 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 **phyBOARD-Segin i.MX 93** and **phyBOARD-Nash i.MX 93 Kit**, the default values are defined in the U-Boot devicetree (e.g. `arch/arm/dts/imx93-phyboard-segin-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.

## 5.10 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.

### 5.10.1 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}
```

### 5.10.2 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 [i.MX 91 BSP downloads](#) or [i.MX 93 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
```

### 5.10.3 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.



## 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](http://devicetree.org).

### 6.2 PHYTEC i.MX 93 BSP Device Tree Concept

The following sections explain some rules PHYTEC has defined on how to set up device trees for our i.MX 93 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 93 SoC but are just routed down to the carrier board and used there are included in this dts.
- *Overlay.dts* - 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 `phyboard-segin-imx93-2.conf` are:

```
imx93-phyboard-segin-peb-av-02.dtbo
imx93-phyboard-segin-peb-av-18.dtbo
```

(continues on next page)

(continued from previous page)

```
imx93-phyboard-segin-peb-eval-01.dtbo
imx93-phyboard-segin-peb-wlbt-05.dtbo
imx93-phycore-npu.dtbo
imx93-phycore-rpmsg.dtbo
```

Available overlays for phyboard-segin-imx91-1.conf are:

```
imx91-phyboard-segin-peb-av-02.dtbo
imx91-phyboard-segin-peb-av-18.dtbo
imx91-phyboard-segin-peb-eval-01.dtbo
imx91-phyboard-segin-peb-wlbt-05.dtbo
```

Available overlays for phyboard-nash-imx93-1.conf are:

```
imx93-phyboard-nash-jtag.dtbo
imx93-phyboard-nash-peb-av-10.dtbo
imx93-phyboard-nash-peb-av-10-etml1010g3dra.dtbo
imx93-phyboard-nash-peb-av-10-ph128800t006.dtbo
imx93-phyboard-nash-peb-wlbt-07.dtbo
imx93-phyboard-nash-pwm-fan.dtbo
imx93-phyboard-nash-vm016.dtbo
imx93-phyboard-nash-vm020.dtbo
imx93-phycore-npu.dtbo
imx93-phycore-rpmsg.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_DEVICE_TREE` 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-imx93-phyboard-segin-overlay1.dtbo#conf-imx93-phyboard-segin-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-imx93-phyboard-segin-overlay1.dtbo
u-boot=> printenv fit_overlay_conf
fit_overlay_conf=conf-imx93-phyboard-segin-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=imx93-phyboard-segin-overlay1.dtbo imx93-phyboard-segin-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 imx93-phyboard-segin-overlay1.dtbo
u-boot=> printenv overlays
overlays=imx93-phyboard-segin-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.

## ACCESSING PERIPHERALS

To find out which boards and modules are supported by the release of PHYTEC's phyCORE-i.MX 93 BSP described herein, visit our [i.MX 91 BSP](#) or [i.MX 93 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 93 BSP Device Tree Concept to get an understanding of our i.MX 93 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 93 Pin Muxing

The i.MX 93 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 93 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 93 Reference Manual for more information about the specific pins and the muxing capabilities.

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

The following is an example of the pin muxing of the UART1 device in `imx93-phyboard-segin.dts`:

```
pinctrl_uart1: uart1grp {
    fsl,pins = <
        MX93_PAD_UART1_RXD__LPUART1_RX    0x31e
        MX93_PAD_UART1_TXD__LPUART1_TX    0x30e
    >;
};
```

The first part of the string `MX93_PAD_UART1_RXD_LPUART1_RX` names the pad (in this example `UART1_RXD`). The second part of the string (`LPUART1_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 pull up is activated.

The device tree representation for UART1 pinmuxing: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-segin.dts#L292>

## 7.2 Ethernet

phyBOARD-Segin/Nash i.MX 91/93 provides two ethernet interfaces.

- On phyBOARD-Segin i.MX 91/93 we have:
  - a 100 megabit Ethernet provided by phyCORE-i.MX 93
  - and 100 megabit Ethernet provided by phyBOARD.
- On phyBOARD-Nash i.MX 93 we have:
  - a 100 megabit Ethernet provided by phyCORE-i.MX 93
  - and 1 gigabit Ethernet provided by phyBOARD.

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 `end0` is:

```
end0: 192.168.3.11/24
```

To configure `end0` to dynamic IP over DHCP, go to `/lib/systemd/network/\*-end0.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: for [phyCORE i.MX 91](#) or [phyCORE i.MX 93](#).

The device tree set up for EQOS Ethernet IP core where the PHY is populated can be found here: for [phyBOARD-Segin i.MX 91](#) or [phyBOARD-Segin i.MX 93](#) or [phyBOARD-Nash i.MX 93](#)

### 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 tftpboot 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 end0 in the target kernel:

```
target:~$ ip -statistics address show end0
2: end0: <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 end0 configuration:

```
target:~$ ip address add 192.168.3.11/24 dev end0
```

## 7.3 WLAN/Bluetooth

WLAN and Bluetooth connectivity are enabled on the phyBOARD-Segin i.MX 91/93 using the PEB-WLBT-05 expansion card, and on the phyBOARD-Nash i.MX 93 with the PEB-WLBT-07 expansion card. Installation instructions for these WLAN and Bluetooth expansion cards can be found in the “The PEB-WLBT-05 for phyBOARD-Segin i.MX 91/93 Quickstart Guide” and the “The PEB-WLBT-07 for phyBOARD-Nash i.MX 93 Quickstart Guide”, respectively.”

### 7.3.1 PEB-WLBT-05 on phyBOARD-Segin i.MX 91/93

#### Tip

The PEB-WLBT-05 overlay configuration needs to be activated first, otherwise the PEB-WLBT-05 won't be recognized.

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

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

```
fit_overlay_conf=conf-imx93-phyboard-segin-peb-wlbt-05.dtbo
```

The changes will be applied after a reboot:

```
target:~$ reboot
```

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

With PEB-WLBT-05 adapter, we use Sterling-LWB module from LSR for WLAN and Bluetooth support. This module supports 2,4 GHz bandwidth and can be run in several modes, like client mode, Access Point (AP) mode using WEP, WPA, WPA2 encryption, and more. More information about the module can be found at [https://connectivity-staging.s3.us-east-2.amazonaws.com/2019-09/CS-DS-SterlingLWB%20v7\\_2.pdf](https://connectivity-staging.s3.us-east-2.amazonaws.com/2019-09/CS-DS-SterlingLWB%20v7_2.pdf)

#### Note

For proper Bluetooth operation please make sure to follow the “The PEB-WLBT-05 for phyBOARD-Segin i.MX 91/93 Quickstart Guide” to correctly set jumper J9 & J10 configurations.

### 7.3.2 PEB-WLBT-07 on phyBOARD-Nash i.MX 93

#### Tip

The PEB-WLBT-07 overlay configuration needs to be activated first, otherwise the PEB-WLBT-07 won't be recognized.

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

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

```
fit_overlay_conf=conf-imx93-phyboard-nash-peg-wlbt-07.dtbo
```

The changes will be applied after a reboot:

```
target:~$ reboot
```

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

With PEB-WLBT-07 adapter, we use MAYA-W2 from u-blox for WLAN and Bluetooth support. This module supports dual-band 2,4 GHz and 5 GHz bandwidth and can be run in several modes, like client mode, Access Point (AP) mode using WEP, WPA, WPA2 encryption, and more. More information about the module can be found at [https://content.u-blox.com/sites/default/files/documents/MAYA-W2\\_DataSheet\\_UBX-22009721.pdf](https://content.u-blox.com/sites/default/files/documents/MAYA-W2_DataSheet_UBX-22009721.pdf)

#### Note

The following WLAN chapter assumes wireless network interface name is wlan0. However with PEB-WLBT-07 adapter the name of the WLAN interface is actually mlan0. Thus when using commands to configure wireless network, substitute wlan0 with mlan0 when using PEB-WLBT-07 on phyBOARD-Nash i.MX 93.

### 7.3.3 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).

### 7.3.4 Bluetooth

Bluetooth is connected to UART5 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>
```

### 7.3.5 Visibility

To activate visibility:

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

To disable visibility:

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

### 7.3.6 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 93 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:

- for phyBOARD-Segin i.MX 91: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx91-phyboard-segin.dts#L214>
- for phyBOARD-Segin i.MX 93: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-segin.dts#L215>
- for phyBOARD-Nash i.MX 93: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-nash.dts#L213>

DT configuration for the eMMC interface can be found here:

- for phyBOARD-Segin i.MX 91: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx91-phycore-som.dtsi#L207>
- for phyBOARD-Segin i.MX 93 or phyBOARD-Nash i.MX 93: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx93-phycore-som.dtsi#L214>

## 7.5 e.MMC Devices

PHYTEC modules like phyCORE-i.MX 93 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 93 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)
=====

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
#0R
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

[...]
```

(continues on next page)

(continued from previous page)

```
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 phyCORE-i.MX 93 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.
  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
```

(continues on next page)

(continued from previous page)

**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](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 93 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 93 SoC:

SoC	Offset User Area	Offset Boot Partition	e.MMC Device
i.MX 93	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 phyBOARD-Segin/Nash i.MX 91/93 doesn't have a set of pins especially dedicated for user I/Os since all GPIOs are used by kernel device drivers or used for a specific purpose. The processor has organized its GPIOs into five banks of 32 GPIOs each (GPIO1 – GPIO4) GPIOs. `gpiochip0`, `gpiochip32`, `gpiochip64` and `gpiochip96` are the sysfs representation of these internal i.MX 93 GPIO banks GPIO1 – GPIO4.

The GPIOs are identified as `GPIO<X>_<Y>` (e.g. `GPIO4_07`). `<X>` identifies the GPIO bank and counts from 1 to 4, 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 [43810000.gpio] (32 lines)
gpiochip1 [43820000.gpio] (32 lines)
gpiochip2 [43830000.gpio] (32 lines)
gpiochip3 [47400000.gpio] (32 lines)
```

**Note**

Order of GPIOchips in i.MX 93 Application Processor Reference Manual and in Linux kernel differ!

GPIOchip address	Linux	Reference Manual
0x43810000	gpiochip0	gpiochip2
0x43820000	gpiochip1	gpiochip3
0x43830000	gpiochip2	gpiochip4
0x47400000	gpiochip3	gpiochip1

- 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 3 from chip0):

```
target:~$ gpioget -c gpiochip0 3
```

- Set the value of GPIO 3 on chip0 to 0 and daemonize:

```
target:~$ gpioset -z -c gpiochip0 3=0
```

**Note**

When demonizing gpioset command please note that the process is still running in the background and you need to kill it afterward to release the GPIO. Otherwise you might get an error when trying to change state of the same GPIO:

```
target:~$ gpioset -z -c gpiochip0 3=1
gpioset: unable to request lines on chip '/dev/gpiochip0': Device or resource busy
```

This is the expected behavior in libgpiod version 2.

As a workaround it is possible to use the `-t 0` switch:

```
target:~$ gpioset -t0 -c gpiochip0 3=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:
```

(continues on next page)

(continued from previous page)

```

--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.  
 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 imx9\_phytec\_defconfig config in the linux kernel sources under arch/arm64/configs

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

## 7.7 ADC

The PHYTEC phyCORE-i.MX 93 includes general purpose Analog-to-Digital Converters (ADC) which can be used for interfacing analog sensors.

Reading the ADC values can be done through sysfs:

```
target:~$ cat /sys/bus/iio/devices/iio:deviceX/in_voltageY_raw
```

On phyBOARD-Nash i.MX 93 the ADC lines are accessible on X16 expansion connector:

ADC input	X16 pin
ADC_IN0	47
ADC_IN2	49

Additionally, there is a current sensing circuitry available on phyBOARD-Nash i.MX 93. It is capable of measuring input current consumption of the phyCORE-i.MX 93 SoM @ 3.3V. The circuitry consists of a MAX4372 current-sensing amplifier (50 V/V) with two 70 mOhm shunts resistors in parallel configuration (effective R = 35 mOhm) connected to the ADC input channel ADC\_IN1 (Vref = 1.8V) via voltage divider with ratio of 1/2. This results in a current scaling factor of 0.502232142 mA/LSB.

Current scaling factor is available as a sysfs parameter /sys/bus/iio/devices/iio:device1/in\_current0\_scale.

The phyCORE-i.MX 93 SoM consumption can thus be measured with this simple script:

```
#!/bin/sh
RAW=$(cat /sys/bus/iio/devices/iio:device1/in_current0_raw)
SCALE=$(cat /sys/bus/iio/devices/iio:device1/in_current0_scale)
CURRENT=$(echo "$RAW * $SCALE" | bc -l)
printf "Current: %.3f mA\n" "$CURRENT"
```

### Note

Current scaling factor calculation depends on CONFIG\_IIO\_RESCALE=y kernel configuration.

In our BSP we also enable config option CONFIG\_SENSORS\_IIO\_HWMON=y which provides hwmon interface to conveniently measure phyCORE-i.MX 93 SoM consumption directly via the sysfs parameter:

```
root@phyboard-nash-imx93-1:~# cat /sys/class/hwmon/hwmon1/curr1_input
415
```

In the example output provided above, the current consumption measured was 415 mA @ 3.3V.

## 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
green:heartbeat@ mmc0::@ mmc1::@ yellow:@
```

Here the LEDs `green:heartbeat` is on the phyCORE-i.MX 93. If you are using phyBOARD-Segin there is also `yellow` LED which is populated on the PEB-EVAL-01.

- To toggle the LEDs ON:

```
target:~$ echo 1 > /sys/class/leds/green\:heartbeat/brightness
```

- To toggle OFF:

```
target:~$ echo 0 > /sys/class/leds/green\:heartbeat/brightness
```

Device tree configuration for the User I/O configuration can be found here: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-segin-peb-eval-01.dts#L28> or <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx91-phyboard-segin-peb-eval-01.dts#L28>

## 7.9 I<sup>2</sup>C Bus

The i.MX 93 contains several Multimaster fast-mode I<sup>2</sup>C modules. PHYTEC boards provide plenty of different I<sup>2</sup>C devices connected to the I<sup>2</sup>C modules of the i.MX 93. This section describes the basic device usage and its DT representation of some I<sup>2</sup>C devices integrated into our phyBOARD-Segin/Nash i.MX 91/93.

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 I<sup>2</sup>C3 bus configuration (e.g. `imx93-phycore-som.dtsi`): <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx93-phycore-som.dtsi#L111> or <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx91-phycore-som.dtsi#L104>

General I<sup>2</sup>C2 bus configuration for `imx93-phyboard-segin.dts`:

- for phyBOARD-Segin i.MX 91: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx91-phyboard-segin.dts#L143>
- for phyBOARD-Segin i.MX 93: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-segin.dts#L144>
- for phyBOARD-Nash i.MX 93: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-nash.dts#L128>

## 7.10 EEPROM

There are two different I2C EEPROM flashes populated on phyCORE-i.MX 93 SoM and on the phyBOARD-Segin/Nash i.MX 91/93. For now only the one on the phyCORE-i.MX 93 is enabled, and it is used for board detection.

### 7.10.1 I2C EEPROM on phyCORE-i.MX 93

The I2C EEPROM on the phyCORE-i.MX 93 SoM has its memory divided into two parts.

- normal area (size: 4096 bytes, bus: I2C-2, addr: 0x50)
- ID page (size: 32 bytes, bus: I2C-2, addr: 0x58)

It is possible to read and write from the device populated:

```
target:~$ hexdump -C /sys/class/i2c-dev/i2c-2/device/2-0058/eeprom
```

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

```
target:~$ dd if=/sys/class/i2c-dev/i2c-2/device/2-0050/eeprom bs=1 count=1024 | od -x
```

To fill the whole EEPROM (ID page) with zeros we first need to disable the EEPROM write protection, use:

```
target:~$ gpioset -t0 -c 2 21=0
```

Then the EEPROM can be written to:

```
target:~$ dd if=/dev/zero of=/sys/class/i2c-dev/i2c-2/device/2-0058/eeprom bs=32 count=1
```

To re-enable EEPROM wire protection, use:

```
target:~$ gpioset -t0 -c 2 21=1
```

#### Warning

The first 256 bytes of the normal EEPROM area (bus: I2C-2 addr: 0x50) are reserved and should not be overwritten! (See below)

### 7.10.2 EEPROM SoM Detection

PHYTEC uses first 256 bytes in EEPROM normal area to store information about the SoM. This includes PCB revision and mounting options.

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

If the first 256 bytes of the normal area are deleted, the bootloader will fall back to the phyCORE-i.MX 93 Kit RAM setup, which is 1GiB RAM.

#### Warning

Data in the first 256 bytes of the normal EEPROM area (bus: I2C-2 addr: 0x50) shouldn't be erased or corrupted! This might influence the behavior of the bootloader. The board might not boot correctly anymore.

### 7.10.3 Rescue EEPROM Data

The hardware introspection data is pre-written on the EEPROM data spaces. If you have accidentally deleted or overwritten the HW data, you could contact our support!

DT representation, e.g. in phyCORE-i.MX 93 file can be found in our PHYTEC git: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx93-phycore-som.dtsi#L172> or <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx91-phycore-som.dtsi#L170>

## 7.11 RTC

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

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

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

- You will get, for example:

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

### Tip

This will list all RTCs including the non-I<sup>2</sup>C RTCs. Linux assigns RTC device IDs based on the device tree/aliases entries if present.

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

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

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

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

### Note

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

Using the `date` command does not change the time and date of the RTC, so if we were to restart the target those changes would be discarded. To write to the RTC we need to use the `hwclock` command. Write the

current date and time (set with the date command) to the RTC using the hwclock tool and reboot the target to check if the changes were applied to the RTC:

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

To set the time and date from the RTC use:

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

### 7.11.1 RTC Wakealarm

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

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

#### Note

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

### 7.11.2 RTC Parameters

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

- We can check RTC supported features with:

```
target:~$ hwclock --param-get features
The RTC parameter 0x0 is set to 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 I<sup>2</sup>C RTCs:

- for phyBOARD-Segin i.MX 91: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx91-phyboard-segin.dts#L164>
- for phyBOARD-Segin i.MX 93: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-segin.dts#L165>
- for phyBOARD-Nash i.MX 93: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-nash.dts#L138>

## 7.12 USB Host Controller

The USB controller of the i.MX 93 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 480 Mbps (HighSpeed ‘HS’). The USB subsystem has two independent USB controller cores. Both cores are capable of acting as a USB peripheral device or a USB host, but on the phyBOARD-Segin/Nash i.MX 91/93 one of them is used as a host-only port (USB-A connector).

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.

The OTG port provides an additional pin for over-current protection, which is not used on the phyBOARD-Segin/Nash i.MX 91/93. Since it’s not used, the driver part is also disabled from within the device tree. In case this pin is used, activate this over-current in the device tree and set the correct polarity (active high/low) according to the device specification. For the correct setup, please refer to the Kernel documentation under `Linux/Documentation/devicetree/bindings/usb/ci-hdrc-usb2.txt`.

DT representation for USB Host:

- for phyBOARD-Segin i.MX 91: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx91-phyboard-segin.dts#L193>
- for phyBOARD-Segin i.MX 93: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-segin.dts#L194>
- for phyBOARD-Nash i.MX 93: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-nash.dts#L192>

## 7.13 USB OTG

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

### 7.13.1 USB Device

In order to connect the board's USB device to a USB host port (for example a PC), you need to configure the appropriate USB gadget. With USB configs 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  
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 RS232/RS485

The phyBOARD-Nash i.MX 93 i.MX 93 SoC provides one RS232/RS485 serial port.

**Warning**

RS232 with HW flow control and RS485 are not working due to HW bug on the phyBOARD-Nash i.MX 93 PCB revision 1616.0

### 7.14.1 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/ttyLP6 115200 crtscts raw -echo
```

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

```
target:~$ echo 123 > /dev/ttyLP6
```

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

The host should print out “123”.

### 7.14.2 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/ttyLP6 -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/ttyLP6
```

You can show all options with:

```
target:~$ rs485conf /dev/ttyLP6 -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>

The device tree representation for RS232 and RS485: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-nash.dts#L185>

## 7.15 CAN FD

The phyBOARD-Segin/Nash i.MX 91/93 has one flexCAN interface 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 can0, such as bit rate and error counters, type:

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

can0	Interface Name
NOARP	CAN cannot use ARP protocol
MTU	Maximum Transfer Unit
RX packets	Number of Received Packets
TX packets	Number of Transmitted Packets
RX bytes	Number of Received Bytes
TX bytes	Number of Transmitted Bytes
errors...	Bus Error Statistics

The CAN configuration is done in the systemd configuration file `/lib/systemd/network/11-can.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-can.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-can.network` file.

To disable flexible datarate manually, one can use:

```
target:~$ ip link set can0 down
target:~$ ip link set can0 type can bitrate 500000 dbitrate 0 fd off
target:~$ ip link set can0 up
```

The bitrate can also be changed manually, for example:

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

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

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

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

```
target:~$ cangen
```

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

Device Tree CAN configuration of `imx93-phyboard-segin.dts`:

- for phyBOARD-Segin i.MX 91: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx91-phyboard-segin.dts#L135>
- for phyBOARD-Segin i.MX 93: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-segin.dts#L136>
- for phyBOARD-Nash i.MX 93: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-nash.dts#L120>

## 7.16 Audio on phyBOARD-Segin i.MX 91/93

On phyBOARD-Segin i.MX 91/93 the TI TLV320AIC3007 audio codec is used. It uses I2S for data transmission and I2C for codec control. The audio signals available are:

- Stereo LINE IN,
- Stereo LINE OUT,
- Output where D-Class 1W speaker can be connected

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

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

Or type for recording devices:

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

### 7.16.1 Alsamixer

To inspect the capabilities of your soundcard, call:

```
target:~$ alsamixer
```

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

### 7.16.2 Restore default volumes

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

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

You can restore saved alsa settings with:

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

### 7.16.3 Playback

Run `speaker-test` to check playback availability:

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

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

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

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

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

If Speaker volume is too low you can increase its volume with (values 0-3):

```
target:~$ amixer -c 0 sset Class-D 3
```

#### Hint

Speaker output is only mono so when stereo track is played only left channel will be played by speaker.

### 7.16.4 Capture

`arecord` is a command-line tool for capturing audio streams which use Line In as the default input source.

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

#### Hint

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

Device Tree Audio configuration:

- for phyBOARD-Segin i.MX 91: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx91-phyboard-segin.dts#L79>
- for phyBOARD-Segin i.MX 93: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-segin.dts#L80>

## 7.17 Audio on phyBOARD-Nash i.MX 93

#### Warning

Due to HW bug Audio is broken on phyBOARD-Nash i.MX 93 PCB revision: 1616.0

To use audio with phyBOARD-Nash i.MX 93 an additional adapter for the Audio/Video connector is needed. The PEB-AV-10 (1531.1 revision) can be bought separately to the Kit. PEB-AV-10 is populated with a TI TLV320AIC3007 audio codec. Audio support is done via the I2S interface and controlled via I2C.

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

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

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

Or type for recording devices:

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

### 7.17.1 Alsamixer

To inspect the capabilities of your soundcard, call:

```
target:~$ alsamixer
```

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

### 7.17.2 Restore default volumes

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

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

You can restore saved alsa settings with:

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

### 7.17.3 ALSA configuration

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

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

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

To set PEB-AV-10 as output, set `playback.pcm` from “dummy” to “pebav10”:

```
[...]

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

[...]
```

If the sound is not audible change playback devices to the software volume control playback devices, set `playback.pcm` to the respective softvol playback device e.g. “softvol\_ pebav10”. Use alsamixer controls to vary the volume levels.

```
[...]
```

(continues on next page)

(continued from previous page)

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

[...]
```

### 7.17.4 Pulseaudio Configuration

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

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

To select the output device, type:

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

### 7.17.5 Playback

Run `speaker-test` to check playback availability:

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

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

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

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

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

### 7.17.6 Capture

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

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

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

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

#### Hint

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

Device Tree Audio configuration: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-nash-peb-av-10.dtsi#L60>

## 7.18 Video

### 7.18.1 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
```

## 7.19 Display

The **phyBOARD-Segin** i.MX 91/93 supports PEB-AV-02 with 7" edt, etm0700g0edh6 parallel display with capacitive touchscreen. Device-tree overlay for the aforementioned display is enabled in `/boot/overlays.txt` by default!

Additionally, since release PD26.1.0 **phyBOARD-Segin** i.MX 91/93 supports PEB-AV-18 with 7" powertip, ph800480t032-zhc19 parallel display with capacitive touchscreen. Device-tree overlay for the aforementioned display can be enabled in `/boot/overlays.txt`:

```
fit_overlay_conf=conf-imx93-phyboard-segin-peb-av-18.dtbo
```

The **phyBOARD-Nash** i.MX 93 needs additional adapter to support 10" edt, etm11010g3dra or powertip, ph128800t006-zhc01 LVDS displays with capacitive touchscreen. The PEB-AV-10 (1531.1 revision) can be bought separately to the Kit. Device-tree overlay for the aforementioned panel `powertip,ph128800t006-zhc01` is enabled in `/boot/overlays.txt` by default!

### 7.19.1 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 qtpy
```

- Weston can be stopped with:

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

### Note

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

## 7.19.2 Backlight Control

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

### Note

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

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

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

Valid brightness values are 0 to `<max_brightness>`.

- To obtain the current brightness level, type:

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

- Write to the file `brightness` to change the brightness:

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

turns the backlight off for example.

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

The device tree of PEB-AV-02/PEB-AV-18 can be found here:

- for phyBOARD-Segin i.MX 91: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx91-phyboard-segin-peb-av-02.dtso> <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx91-phyboard-segin-peb-av-18.dtso>
- for phyBOARD-Segin i.MX 93: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-segin-peb-av-02.dtso> <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-segin-peb-av-18.dtso>

The device tree of PEB-AV-10 for phyBOARD-Nash i.MX 93 can be found here: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-nash-peb-av-10.dtso>

## 7.20 Power Management

### 7.20.1 CPU Core Frequency Scaling

The CPU in the i.MX 93 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. Unlike i.MX8 M family the i.MX 93 doesn't support *Dynamic Voltage and Frequency Scaling (DVFS)*, but has the support of basic **Voltage and Frequency Scaling (VFS)**. The board can be put into these modes:

- nominal (ND),
- overdrive (OD),
- Low Drive (LD) and
- Low Drive (LD) with Software Fast Frequency Change (SWFFC).

Mode	CPU freq	DDR data rate	VDD_SOC
OverDrive (OD)	1.7 GHz	3733 MT/s	900mV
NominalDrive (ND)	1.4 GHz	1866 MT/s	850mV
LowDrive (LD)	900 MHz	1866 MT/s	800mV
LowDrive (LD) with SWFFC	900 MHz	625 MT/s	800mV

The i.MX 93 BSP supports the VFS feature. The Linux kernel provides a LPM driver that allows setting VDD\_SOC, CPU freq and DDR speed.

#### Note

Low-cost i.MX 93 SoC variants such as parts numbers NXP IMX9301/IMX9302 do not support VFS features. Those SoCs always run in LowDrive (LD) mode. Hence, the Linux LPM driver is disabled automatically for SoMs with such SoCs.

- To put the device in **OverDrive (OD)** mode type:

```
target:~$ echo 0 > /sys/devices/platform/imx93-lpm/mode
```

- To put the device in **NominalDrive (ND)** mode type:

```
target:~$ echo 1 > /sys/devices/platform/imx93-lpm/mode
```

- To put the device in **LowDrive (LD)** mode type:

```
target:~$ echo 2 > /sys/devices/platform/imx93-lpm/mode
```

- To put the device in **LowDrive (LD)** mode with the lowest DDR speed with SWFFC type:

```
target:~$ echo 3 > /sys/devices/platform/imx93-lpm/mode
```

- To check the current CPU frequency type:

```
target:~$ mhz
```

- To check the current mode and DDR frequency type:

```
target:~$ cat /sys/devices/platform/imx93-lpm/mode
```

- To check the current VDD\_SOC type:

```
target:~$ cat /sys/kernel/debug/regulator/regulator_summary
```

For more detailed information about the LPM driver and modes, refer to the NXP's documentation: [https://docs.nxp.com/bundle/AN13917/page/topics/low\\_power\\_mode\\_use\\_cases.html](https://docs.nxp.com/bundle/AN13917/page/topics/low_power_mode_use_cases.html)

## 7.20.2 CPU Core Management

The i.MX 93 SoC can have multiple processor cores on the die. The i.MX 93, for example, has 2 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 -l
```

- This will show, for example:

```
cpu0/  
cpu1/  
cpufreq/  
[...]
```

Here the system has two 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/cpu1/online
```

As confirmation, you will see:

```
[ 110.505012] psci: CPU1 killed (polled 0 ms)
```

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/cpu1/online
```

## 7.20.3 Suspend to RAM

The phyCORE-i.MX 93 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/ttyLP0/power/wakeup
target:~$ echo mem > /sys/power/state
```

Device can be put into suspend and waken-up with PEB-EVAL-01 S2 button

To wake up with RTC alarm check: *RTC Wakealarm*

## 7.21 Thermal Management

### 7.21.1 U-Boot

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

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

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

### 7.21.2 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 93 SoC platform. The i.MX 9 has internal temperature sensors for the SoC.

- The current temperature can be read in millicelsius with:

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

- You will get, for example:

```
49000
```

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

	Commercial	Industrial	Extended Industrial
passive (warning)	85°C	95°C	115°C
critical (shutdown)	90°C	100°C	120°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*, the board immediately shuts down to avoid any heat damage. If this doesn't meet your expectations, an external supervisor circuit

that starts the module again with X\_ONOFF signal when the temperature drops below a selected trip point can be implemented

#### Note

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

### 7.21.3 PWM Fan

A PWM fan can be connected to the phyBOARD-Nash i.MX 93 connector X48 (label FAN).

Afterwards, a PWM fan overlay needs to be activated, otherwise PWM fan won't be recognized.

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

The overlays.txt file should look like this (it can also contain other devicetree overlay configuration!):

```
fit_overlay_conf=conf-imx93-phyboard-nash-pwm-fan.dtbo
```

The changes will be applied after a reboot:

```
target:~$ reboot
```

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

The SoC only contains one temperature sensor which is already used by the thermal frequency scaling. The fan thus can not be controlled by the kernel. We use lmsensors with hwmon for this instead. lmsensors reads the temperature periodically and adjusts output PWM duty-cycle accordingly. By default, temperature threshold for PWM fan to activate is set to 60°C.

The settings can be configured in the configuration file:

```
/etc/fancontrol
```

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

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

## 7.22 Watchdog

The PHYTEC i.MX 93 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.

### 7.22.1 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
RebootWatchdogSec=10min
```

*RuntimeWatchdogSec* defines the timeout value of the watchdog, while *RebootWatchdogSec* 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
```

## 7.23 Power Key

The X\_ONOFF pin connected to the ON/OFF button can be pressed long (for 5 seconds) to trigger Power OFF without SW intervention or used to wake up the system out of suspend. With the bbnsn\_pwrkey driver, the KEY\_POWER event is also reported to userspace when the button is pressed. On default, systemd is configured to ignore such events. The function of Power OFF without SW intervention is not configured. Triggering a power off with systemd when pushing the ON/OFF button can be configured under `/usr/lib/systemd/logind.conf.d/00-systemd-conf-imx.conf` and set using:

```
HandlePowerKey=poweroff
```

By default, ON/OFF button is configured to emit the KEY\_POWER event code when short-pressed. This behavior can be verified using the `evtest` tool, which is included in our BSP:

```
target:~$ evtest /dev/input/by-path/platform-44440000.bbnsn:pwrkey-event
Input driver version is 1.0.1
Input device ID: bus 0x19 vendor 0x0 product 0x0 version 0x0
Input device name: "44440000.bbnsn:pwrkey"
Supported events:
Event type 0 (EV_SYN)
Event type 1 (EV_KEY)
    Event code 116 (KEY_POWER)
Properties:
Testing ... (interrupt to exit)
Event: time 1762862749.414004, type 1 (EV_KEY), code 116 (KEY_POWER), value 1
Event: time 1762862749.414004, ----- SYN_REPORT -----
Event: time 1762862749.542006, type 1 (EV_KEY), code 116 (KEY_POWER), value 0
Event: time 1762862749.542006, ----- SYN_REPORT -----
```

However, it is possible to reassign the short-press event by modifying the device-tree and changing the default `linux,code` property of the `bbnsn_pwrkey` node:

```
&bbnsn_pwrkey {
    linux,code = <KEY_SLEEP>;
};
```

In this example, the ON/OFF button is assigned KEY\_SLEEP code, and pressing the button will now put the board into sleep mode by default.

All valid key codes are defined in the `include/uapi/linux/input-event-codes.h` file part of the Linux kernel.

## 7.24 PXP

The i.MX 93 SoC contains an PiXel Pipeline (PXP). The PXP combines the following into a single processing engine:

- Scaling
- Color Space Conversion (CSC)
- Secondary Color Space Conversion (CSC2)
- Rotation

and thus minimizes the memory footprint required for the display pipeline. How to use the PXP with GStreamer and Wayland check the *How to Use PXP in GStreamer and Wayland* (AN13829) Application note from NXP.

## 7.25 On-Chip OTP Controller (OCOTP\_CTRL) - eFuses

The i.MX 93 provides one-time programmable fuses to store information such as the MAC address, boot configuration, and other permanent settings (“On-Chip OTP Controller (OCOTP\_CTRL)” in the i.MX 93 Reference Manual). The following list is an abstract from the i.MX 93 Reference Manual and includes some useful registers in the OCOTP\_CTRL (at base address 0x47510000):

Name	Bank	Word	Memory offset at 0x47510000	Description
BOOT_CF0	3	0	0x60	boot fuse settings
BOOT_CF1	3	1	0x64	boot fuse settings
BOOT_CF2	3	2	0x68	boot fuse settings
BOOT_CF3	3	3	0x6c	boot fuse settings
MAC1_ADDR	39	3	0x4ec	contains lower 32 bits of ENET0 MAC address
MAC1/2_ADDR	39	4	0x4f0	contains upper 16 bits of ENET0 MAC address and the lower 16 bits of ENET1 MAC address
MAC2_ADDR	39	5	0x4f4	contains upper 32 bits of ENET1 MAC address

A complete list and a detailed mapping between the fuses in the OCOTP\_CTRL and the boot/mac/... configuration are available in the section “Fuse Map” of the i.MX 93 Security Reference Manual.

### 7.25.1 Reading Fuse Values in uBoot

MAC1\_ADDR:

```
u-boot=> fuse read 39 3
```

### 7.25.2 Reading Fuse Values in Linux

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

```
target:~$ hexdump /sys/devices/platform/soc\@0/47510000.efuse/fsb_s400_fuse0/nvmem
```

### 7.25.3 Burning MAC addresses

Let's say we want to burn the following MAC addresses:

MAC1	12:34:56:78:90:Aa
MAC2	Bb:Cc:Dd:Ee:Ff:D0

We would execute this in u-boot:

```
u-boot=> fuse prog 39 3 0x567890Aa
u-boot=> fuse prog 39 4 0xFfD01234
u-boot=> fuse prog 39 5 0xBbCcDdEe
```

### 7.25.4 Burning Boot Fuses

#### Warning

Fuses can only be written once! You can brick your board easily by burning the wrong boot configuration. It cannot be reversed!

Which fuse bank/word should be used to program the BOOT\_CFGX can be checked in *i.MX 93 Applications Processor Reference Manual* attached spreadsheet named **i.MX93\_Fusemap.xlsx**.

These values should be written to the BOOT\_CFG0, which can be read/written from fuses on Bank 3, Word 0.

Boot Device	BOOT_CFG0
eMMC	0x20020002
SD Card	0x20000103

To set internal fuses to boot from eMMC one can program them with:

```
u-boot=> fuse prog 3 0 0x20020002
```

In this example we:

- set the Boot\_Mode to 0b0010 (eMMC) with BOOT\_CFG0[3:0],
- set the eMMC Bus width to 0b01 (8 bit) with BOOT\_CFG0[18:17]
- set the BT\_FUSE\_SEL (Boot fuses already programmed) bit with BOOT\_CFG0[29]

Make sure you set the right bits by reading the **Boot Fusemap** chapter in *i.MX 93 Applications Processor Reference Manual*.

## 7.26 TPM

The phyBOARD-Segin/Nash i.MX 91/93 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-----
MIIBVQIBADANBgkqhkiG9w0BAQEFAASCAT8wggE7AgEAAkEAxsvmbxjwuKnYeuZ
2AVBmuLvYyqF/LpY0D3IB/v+YvEoLxdGGmjiFLECU6xZ1j3+dIt4Y1zbcKS10cWT
I8mbSwIDAQABAKBoy8wrYNhmP/1kzUJIcLznPYJckGoZlFI1M7xjGSA9H1xDK6if
5g5CYCHPrbBp8e0mEokPRZoihxxzGTxGPiAhAiEA/70YM0pVZ5SD3YcRsWcQlkWI
M0SPUYg6vxvGG9xp4FcCIQDHB01RoHr+qXJwxIu3/3oQAUBI4ACJ4JRp0KelwhC0
LQIHANJzSvg/dak5l8pU55/99q3nbn7nPnnZSJiP0F6P62gjAiEAjf7qrfMF7Uyt
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-----
MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAMB5nG8Y8Lip2HrmdgFQZri72MqhfY6
WDg9yAf7/mLxKJcXRhpo4hSxAl0sWdY9/nSLeGNC23CktTnFkyPJm0sCAwEAAQ==
-----END PUBLIC KEY-----
```

Device tree TPM configuration can be found here: <https://github.com/phytec/linux-phytec-imx/blob/v6.12.34-2.1.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-nash.dts#L166>

## 7.27 JTAG

A JTAG debugger can be connected to phyBOARD-Segin/Nash i.MX 91/93. 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\\_93](https://kb.segger.com/NXP_i.MX_93)

### 7.27.1 Enabling JTAG Debug Interface on phyBOARD Nash

The i.MX 93 has a JTAG debug interface which can be used to debug software running on the SoC. The JTAG interface is routed through the pinmux and is not enabled by default. To enable the JTAG interface, please add this overlay configuration to your `overlays.txt` file:

```
#conf-imx93-phyboard-nash-jtag.dtbo
```

#### Note

JTAG and the MIPI CSI interface cannot be used at the same time, since they share the same pins. Jumper JP15 switches signals between JTAG and MIPI CSI. It needs to be set to use JTAG.

### 7.27.2 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 phyBOARD-Segin/Nash i.MX 91/93 is oriented correctly. The first pin is usually marked on the connector and the cable.

### 7.27.3 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 Pop-up 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](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>
...
J-Link>
```

## 7.27.4 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
```

## I.MX 93 M33 CORE

In addition to the Cortex-A55 cores, there is a Cortex-M33 Core as MCU integrated into the i.MX 93 SoC. Our Yocto-Linux-BSP runs on the A55-Cores and the M33 Core can be used as a secondary core for additional tasks using bare-metal firmware. Both cores have access to the same peripherals and thus peripheral usage needs to be limited either in the M33 Core’s firmware or the devicetree for the Linux operating system.

Our Yocto-BSP contains pre-built firmware examples for M33 Core from NXP.

This section describes how to run pre-built M33 Core firmware examples on phyBOARD-Segin/Nash i.MX 91/93.

### 8.1 Running M33 Core Examples

There are two ways to run the M33 Core firmware examples, from U-Boot bootloader and from Remoteproc subsystem within a running Linux.

To receive debug messages start your favorite terminal software (e.g. Minicom, Tio, or Tera Term) on your host PC and configure it for 115200 baud, 8 data bits, no parity, and 1 stop bit (8n1) with no handshake.

On phyBOARD-Segin/Nash i.MX 91/93 an external “USB TTL to serial adapter” is required. Adapter’s I/O pins should be able to operate at 3.3V voltage levels.

Connect external “USB TTL to serial adapter” signals to the *X16* connector on the board according to the following table:

USB-TTL adapter pins	X16 signal	X16 pin
RXD	X_UART2_TX	5
TXD	X_UART2_RX	8
GND	GND	4

#### 8.1.1 Running Examples from U-Boot

To load firmware examples using the U-Boot bootloader, the `bootaux` command can be used:

1. Prepare an SD Card with our Yocto-BSP
2. Stop the autoboot by pressing any key
3. List available M33 Core firmware examples on the first partition of SD Card:

```
u-boot=> ls mmc 1
```

**Note**

Available firmware examples start with `imx93-11x11-evk_m33_TCM_*` and end with `.bin`. Examples come from NXP's Yocto layer meta-imx and are selected based on compatibility with phyBOARD-Segin/Nash i.MX 91/93 hardware.

1. Load desired firmware example:

```
u-boot=> fatload mmc 1 ${loadaddr} <firmware.bin>
u-boot=> cp.b ${loadaddr} 0x201e0000 ${filesize}
u-boot=> run prepare_mcore
u-boot=> bootaux 0x1ffe0000 0
## Starting auxiliary core addr = 0x1FFE0000...
```

The program's output should appear on the M33 Core's debug UART.

### 8.1.2 Running Examples from Linux using Remoteproc

Remoteproc is a module that allows you to control the M33 Core from Linux during runtime. Firmware examples for M33 Core can be loaded and the execution started or stopped within Linux. To use Remoteproc a devicetree overlay needs to be set:

Edit the `overlays.txt` file located in the `/boot` directory on the target by adding `imx93-phycore-rpmsg.dtbo` overlay configuration:

```
fit_overlay_conf=conf-imx93-phyboard-segin-peb-av-02.dtbo#conf-imx93-phycore-rpmsg.dtbo
```

Restart the target and execute in U-Boot:

```
u-boot=> run prepare_mcore
```

Firmware examples `.elf` files for the M33 Core can be found under `/lib/firmware`. List available firmware examples:

```
target:~$ ls /lib/firmware/*.elf
```

To load the firmware, type:

```
target:~$ echo /lib/firmware/<firmware>.elf > /sys/class/remoteproc/remoteproc0/firmware
target:~$ echo start > /sys/class/remoteproc/remoteproc0/state
```

To load a different firmware, the M33 Core needs to be stopped:

```
target:~$ echo stop > /sys/class/remoteproc/remoteproc0/state
```

**Note**

The samples found in `/lib/firmware` on the target come from NXP's Yocto layer meta-imx and are selected based on compatibility with phyBOARD-Segin/Nash i.MX 91/93 hardware.

Some firmware examples from NXP require additional Linux kernel modules to be loaded.

For example, when loading `imx93-11x11-evk_m33_TCM_rpmsg_lite_str_echo_rtos.elf` firmware, one requires corresponding `imx_rpmsg_tty` module to be loaded:

```
target:~$ modprobe imx_rpmmsg_tty
```

This exposes an RPMsg endpoint as a virtual TTY at `/dev/ttyRPMMSG30`. Now it is possible to send messages from A55 Core to M33 Core by typing:

```
target:~$ echo "PHYTEC" > /dev/ttyRPMMSG30
```

Observing M33 Core debug UART should result in the following output:

```
RPMMSG String Echo FreeRTOS RTOS API Demo...
Nameservice sent, ready for incoming messages...
Get Message From Master Side : "PHYTEC" [len : 6]
```

## 8.2 ARM Ethos-U NPU

The NXP i.MX 93 family of SoCs optionally integrates the ARM Ethos-U65 NPU (neural processing unit). The Ethos-U65 microNPU is designed for efficient machine learning acceleration. It helps developers build more capable, cost-effective, and energy-efficient AI/ML applications.

On i.MX 93 the Ethos-U65 NPU is tied to M33 Core and works with Cortex-M33 Core. ML workloads are offloaded from M33 Core to the NPU. The Cortex-A55 cores handle general-purpose and OS-level tasks (like running Linux), while the M33 Core manages real-time and low-power ML tasks using the NPU.

The Cortex-A55 sends ML tasks to the Cortex-M33 Core via inter-processor communication. The M33 Core then controls the NPU, loads firmware, and runs inference.

Our Yocto BSP (since PD24.2.2) includes pre-built NXP examples for NPU delegation.

This section shows how to run NXP NPU examples on phyBOARD-Segin/Nash i.MX 91/93.

### 8.2.1 Running NPU examples

From Linux, you can delegate NPU tasks using the remoteproc subsystem. First, load the device tree overlays for remoteproc and NPU. Edit `overlays.txt` in the `/boot` directory and add `imx93-phycore-rpmmsg.dtbo` and `imx93-phycore-npu.dtbo` overlay configurations:

```
fit_overlay_conf=conf-imx93-phyboard-segin-peb-av-02.dtbo#conf-imx93-phycore-rpmmsg.dtbo#conf-
↪imx93-phycore-npu.dtbo
```

Reboot the target afterwards.

After reboot, check if `/dev/ethosu0` exists in `devfs`. Then navigate to `/usr/bin/tensorflow-lite-*/examples`, the folder which contains NXP's NPU examples.

Before running a TensorFlow Lite model on i.MX93, you must first convert it to a Vela model. The Vela model is a TensorFlow Lite model optimized for Ethos-U NPUs. It rewrites operators, tweaks quantization, and adjusts memory layout for efficient execution. Compared to a regular TensorFlow Lite model, it is smaller and tailored for the ARM Ethos-U NPU's architecture.

To convert a TensorFlow Lite model to a Vela model, use the BSP-provided `vela` compiler:

```
target:~$ cd /usr/bin/tensorflow-lite-*/examples
target:examples$ vela mobilenet_v1_1.0_224_quant.tflite
```

```
Network summary for mobilenet_v1_1.0_224_quant
```

(continues on next page)

(continued from previous page)

Accelerator configuration	Ethos_U65_256
System configuration	internal-default
Memory mode	internal-default
Accelerator clock	1000 MHz
Design peak SRAM bandwidth	14.90 GB/s
Design peak DRAM bandwidth	3.49 GB/s
Total SRAM used	370.91 KiB
Total DRAM used	3719.84 KiB
CPU operators = 0 (0.0%)	
NPU operators = 60 (100.0%)	
Average SRAM bandwidth	5.53 GB/s
Input SRAM bandwidth	11.40 MB/batch
Weight SRAM bandwidth	9.25 MB/batch
Output SRAM bandwidth	4.10 MB/batch
Total SRAM bandwidth	24.86 MB/batch
Total SRAM bandwidth	per input 24.86 MB/inference (batch size 1)
Average DRAM bandwidth	1.17 GB/s
Input DRAM bandwidth	1.45 MB/batch
Weight DRAM bandwidth	3.08 MB/batch
Output DRAM bandwidth	0.72 MB/batch
Total DRAM bandwidth	5.26 MB/batch
Total DRAM bandwidth	per input 5.26 MB/inference (batch size 1)
Neural network macs	572406226 MACs/batch
Network Tops/s	0.25 Tops/s
NPU cycles	3732560 cycles/batch
SRAM Access cycles	1016537 cycles/batch
DRAM Access cycles	1676662 cycles/batch
On-chip Flash Access cycles	0 cycles/batch
Off-chip Flash Access cycles	0 cycles/batch
Total cycles	4491867 cycles/batch
Batch Inference time	4.49 ms, 222.62 inferences/s (batch size 1)

This step creates the Vela model at `output/mobilenet_v1_1.0_224_quant_vela.tflite`.

#### Note

Converting a TensorFlow Lite model to a Vela model is CPU intensive (~20 seconds). You only need to do this once per model.

Now you can run the NPU examples. This guide uses the NXP example `label_image`. It performs image classification by loading a pre-trained network, processing the input image, and printing the top predicted labels with confidence scores.

```
target:examples$ ./label_image -m output/mobilenet_v1_1.0_224_quant_vela.tflite |
```

(continues on next page)

(continued from previous page)

```

-i grace_hopper.bmp -l labels.txt \
--external_delegate_path=/usr/lib/libethosu_delegate.so
INFO: Loaded model output/mobilenet_v1_1.0_224_quant_vela.tflite
INFO: resolved reporter
INFO: Ethosu delegate: device_name set to /dev/ethosu0.
remoteproc remoteproc0: powering up imx-rproc
remoteproc remoteproc0: Booting fw image ethosu_firmware, size 242844
INFO: Ethosu delegate: cache_file_path set to .
INFO: Ethosu delegate: timeout set to 600000000000.
INFO: Ethosu delegate: enable_cycle_counter set to 0.
INFO: Ethosu delegate: enable_profiling set to 0.
INFO: Ethosu delegate: profiling_buffer_size set to 2048.
INFO: Ethosu delegate: pmu_event0 set to 0.
INFO: Ethosu delegate: pmu_event1 set to 0.
INFO: Ethosu delegate: pmu_event2 set to 0.
INFO: Ethosu delegate: pmu_event3 set to 0.
INFO: EXTERNAL delegate created.
rproc-virtio rproc-virtio.1.auto: assigned reserved memory node vdevbuffer@a4020000
virtio_rpmsg_bus virtio0: rpmsg host is online
rproc-virtio rproc-virtio.1.auto: registered virtio0 (type 7)
rproc-virtio rproc-virtio.2.auto: assigned reserved memory node vdevbuffer@a4020000
virtio_rpmsg_bus virtio1: rpmsg host is online
virtio_rpmsg_bus virtio1: creating channel rpmsg-ethosu-channel addr 0x1e
rproc-virtio rproc-virtio.2.auto: registered virtio1 (type 7)
remoteproc remoteproc0: remote processor imx-rproc is now up
INFO: EthosuDelegate: 1 nodes delegated out of 1 nodes with 1 partitions.
INFO: Applied EXTERNAL delegate.
INFO: invoked
INFO: average time: 3.778 ms
INFO: 0.780392: 653 military uniform
INFO: 0.105882: 907 Windsor tie
INFO: 0.0156863: 458 bow tie
INFO: 0.0117647: 466 bulletproof vest
INFO: 0.00784314: 835 suit

```

Running on the NPU took 3.778 ms. You can now compare this to CPU-only execution. To run without delegating work to the NPU, omit `--external_delegate_path` and use the original TensorFlow Lite model:

```

target:examples$ ./label_image -m mobilenet_v1_1.0_224_quant.tflite \
-i grace_hopper.bmp -l labels.txt
INFO: Loaded model mobilenet_v1_1.0_224_quant.tflite
INFO: resolved reporter
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
INFO: invoked
INFO: average time: 164.281 ms
INFO: 0.768627: 653 military uniform
INFO: 0.105882: 907 Windsor tie
INFO: 0.0196078: 458 bow tie
INFO: 0.0117647: 466 bulletproof vest
INFO: 0.00784314: 835 suit

```

Running on the CPU took 164.281 ms. This makes inference about 40x slower compared to running on the NPU. This highlights why using the NPU is preferred over the CPU for AI/ML tasks.

**Note**

For more details and resources, see NXP's application note: [AN13854 - Hardware acceleration with Ethos-U on i.MX93](#)