phyCORE-i.MX 95 FPSC BSP Manual ALPHA1

PHYTEC Messtechnik GmbH

2025 年 08 月 13 日

Contents

ALPHA1 i.MX 95 BSP 手册	
文档标题	ALPHA1 i.MX 95 BSP 手册
文档类型	BSP 手册
型号	ALPHA1
Yocto 手册	Scarthgap
发布日期	2025/06/02
母文档	ALPHA1 i.MX 95 BSP 手册

下表显示了与本手册兼容的 BSP：

适用 BSP	BSP 发布类型	BSP 发布日期	BSP 状态
BSP-Yocto-NXP-i.MX95-ALPHA1	Alpha	2025/06/02	已发布

本手册指导您完成 BSP 包的安装、编译和烧写，并描述如何使用 **phyCORE-i.MX 95 FPSC Kit** 的硬件接口。本手册还包括如何从源码编译内核、u-boot 镜像。本手册包含需要在 PC(Linux 操作系统) 上执行的指令。

备注

本文档包含指令示例，描述如何在串口终端上与核心板进行交互。指令示例以 “host:~$” 、“target:~$” 或 “u-boot=>” 开头，开头的这些关键字描述了指令执行的软件环境。如果需要复制这些指令，请仅复制这些关键字之后的内容。

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 指南**：phyCORE 使用的 Yocto 版本的综合指南。本指南包含: Yocto 概述；PHYTEC BSP 介绍、编译和定制化修改；如何使用 Poky 和 Bitbake 等编译框架。
- **BSP 手册**：phyCORE 的 BSP 版本专用手册。可在此处找到如何编译 BSP、启动、更新软件、设备树和外设等信息。
- **开发环境指南**：本指南介绍了如何使用 PHYTEC 虚拟机来搭建多样的开发环境。VM 中包含了 Eclipse 和 Qt Creator 的详细上手指导，还说明了如何将所编译出的 demo 程序放到 phyCORE 核心板上运行。本指南同时也介绍了如何在本地 Linux ubuntu 上搭建完整的开发环境。
- **引脚复用表**：phyCORE 核心板附带一个引脚复用表（Excel 格式）。此表将显示从处理器到底板的信号连接以及默认的设备树复用选项。这为开发人员进行引脚复用和设计提供了必要的信息。

除了这些标准手册和指南之外，PHYTEC 还将提供产品变更通知、应用说明和技术说明。这些文档将根据具体案例进行针对性提供。大部分文档都可以在我们产品的 https://www.phytec.de/produkte/system-on-modules/phycore-imx-95-fpsc/#downloads 中找到。

CHAPTER 1

支持的硬件

在我们的网页上，您可以查看适用于 BSP 版本 BSP-Yocto-NXP-i.MX95-ALPHA1 的所有 Machine 及其对应的 Article Numbers(产品型号)：网页.

如果您在 “Supported Machines” 一栏选择了特定的 **Machine Name** ，您可以查看该 machine 下可用的 **Article Numbers** 以及硬件信息的简短描述。如果您只有硬件的 **Article Numbers** ，您可以将 **Machine Name** 下拉菜单留空，仅选择您的 **Article Numbers** 。现在，它应该会显示您特定硬件所需的 **Machine Name**

1.1 Libra FPSC 器件

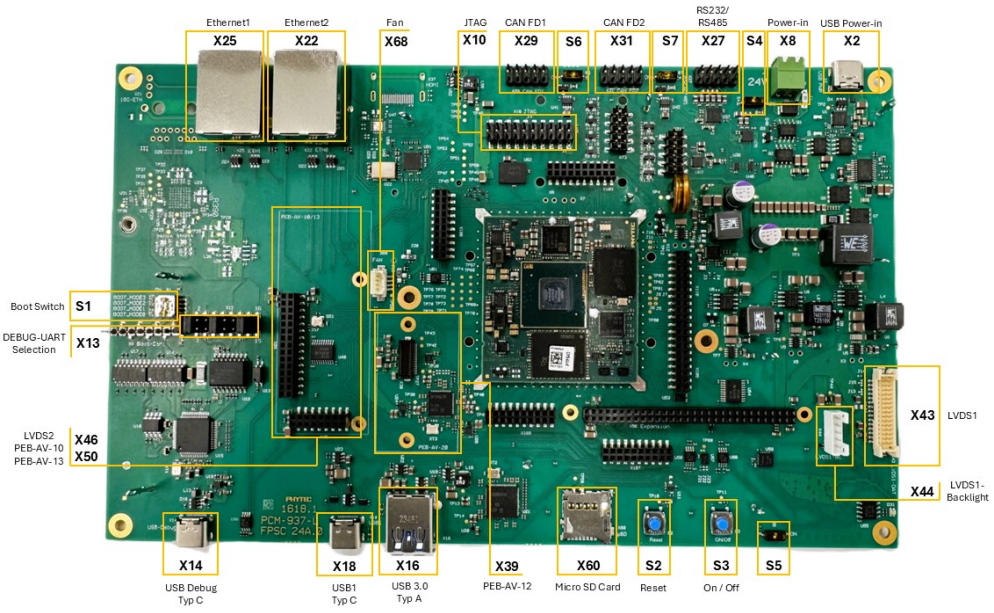


图 1: **Libra FPSC Components (top)**

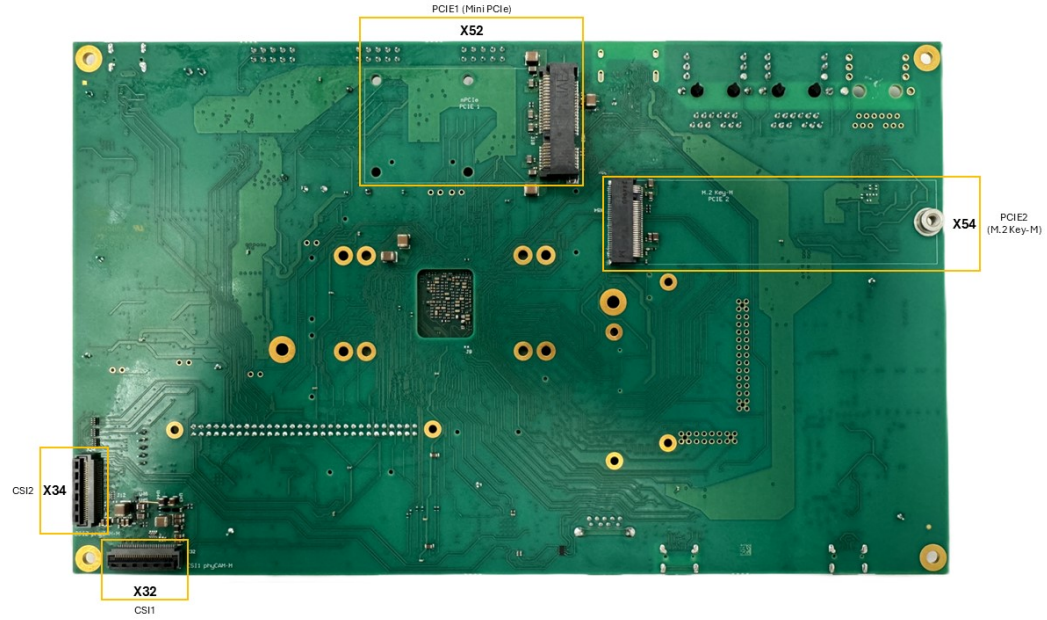


图 2: **Libra FPSC Components (bottom)**

CHAPTER 2

开始使用

该 **phyCORE-i.MX 95 FPSC Kit** 包含预先烧写好的 SD 卡。它包含 phytec-qt6demo-image 镜像，可以直接用作启动盘。默认情况下，核心板上的 eMMC 仅烧写了 U-Boot。您可以从 PHYTEC 下载服务器 获取所有镜像资源。本章将解释如何将 BSP 镜像烧写到 SD 卡以及如何启动开发板。

有几种方法可以将镜像写入 SD 卡或 eMMC。最为人熟知的方式是使用 Linux 命令行工具 `dd` 进行简单的顺序写入。另一种方法是使用 PHYTEC 的自研程序 partup，它可以让格式化复杂系统的过程变得简单。您可以从其发布页面获取 预编译的 Linux partup 二进制文件 。请阅读 partup 的 readme 文件 来获取安装指导。

备注

There is no partup support in imx95 ALPHA1 release yet.

2.1 下载镜像

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

Get the WIC image from the download server:

```
host:~$ wget https://download.phytec.de/Software/Linux/BSP-Yocto-i.MX95/BSP-Yocto-NXP-i.MX95-ALPHA1/images/
↪ampliphy-vendor/imx95-libra-fpsc-1/phytec-qt6demo-image-imx95-libra-fpsc-1.rootfs.wic.xz
```

2.2 将镜像写入 SD 卡

警告

要创建 SD 卡启动盘，必须要拥有 Linux PC 上的 root 权限。在选择烧写设备时请务必小心！所选设备上的所有文件将在命令执行后立即被擦除，而且擦除前不会有任何进一步的确认！

选择错误的设备可能会导致 **数据丢失**，例如，可能会擦除您当前所在 PC 上的系统！

2.2.1 寻找正确的设备

要创建 SD 卡启动盘，首先要找到 PC 上您 SD 卡对应的正确设备名称。在开始将镜像复制到 SD 卡之前，请卸载任何已挂载的分区。

1. 为了获取正确的设备名称，请移除您的 SD 卡并执行：

```
host:~$ lsblk
```

2. 现在插入你的 SD 卡，然后再次执行命令：

```
host:~$ lsblk
```

3. 比较两个输出，以获取第二个输出中的新设备名称。这些是 SD 卡的设备名称（如果 SD 卡已格式化，则包括设备名称和对应的分区）。
4. 为了验证找到的设备名称的最终正确性，请执行命令 `sudo dmesg`。在其输出的最后几行中，您应该也能找到设备名称，例如 `/dev/sde` 或 `/dev/mmcblk0` （具体取决于您的系统）。

或者，您可以使用图形化的程序，例如 GNOME Disks 或 KDE Partition Manager 来找到正确的设备。

现在您已经得到了正确的设备名称，例如 `/dev/sde`，如果 SD 卡曾格式化过，需要确认已取消其分区的挂载，您可以在输出中看到带有附加了数字的设备名称（例如 `/dev/sde1`），它们是 SD 卡的分区。一些 Linux 发行版系统在设备插入时会自动挂载分区。在写入之前，必须卸载这些分区，以避免数据损坏。

卸载所有这些分区，例如：

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

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

2.2.2 使用 bmap-tools

烧写 SD 卡的其中一种方法是使用 bmap-tools 。Yocto 会自动为 WIC 镜像创建一个 block map 文件（`<IMAGENAME>-<MACHINE>.wic.bmap` ），该文件描述了镜像内容并包含数据完整性的校验。*bmaptool* 已被多种 Linux 发行版支持。对于基于 Debian 的系统，可以通过以下命令安装：

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

通过以下命令将 WIC 镜像烧写到 SD 卡：

```
host:~$ bmaptool copy phytec-qt6demo-image-imx95-libra-fpsc-1?(.rootfs).wic?(.xz) /dev/<your_device>
```

将 <your_device> 替换为您之前找到的 SD 卡设备名称，并确保将文件 `<IMAGENAME>-<MACHINE>.wic.bmap` 与 WIC 镜像文件放在一起，以便 bmaptool 知道哪些块需要写入，哪些块需要跳过。

警告

bmaptool 仅擦写 SD 卡上镜像数据所在的区域。这意味着在写入新的镜像后，之前写入的旧 U-Boot 环境变量可能仍然可用。

2.2.3 使用 dd

在卸载所有 SD 卡的挂载分区后，您可以烧写 SD 卡。

一些 PHYTEC BSP 会生成未压缩的镜像（文件名扩展名为 *.wic），而另一些则生成压缩的镜像（文件名扩展名为 *.wic.xz）。

要写入未压缩的镜像（*.wic），请使用以下命令：

```
host:~$ sudo dd if=phytec-qt6demo-image-imx95-libra-fpsc-1?(.rootfs).wic of=/dev/<your_device> bs=1M␣
↪conv=fsync status=progress
```

或者要写入压缩后的镜像（*.wic.xz），请使用以下命令：

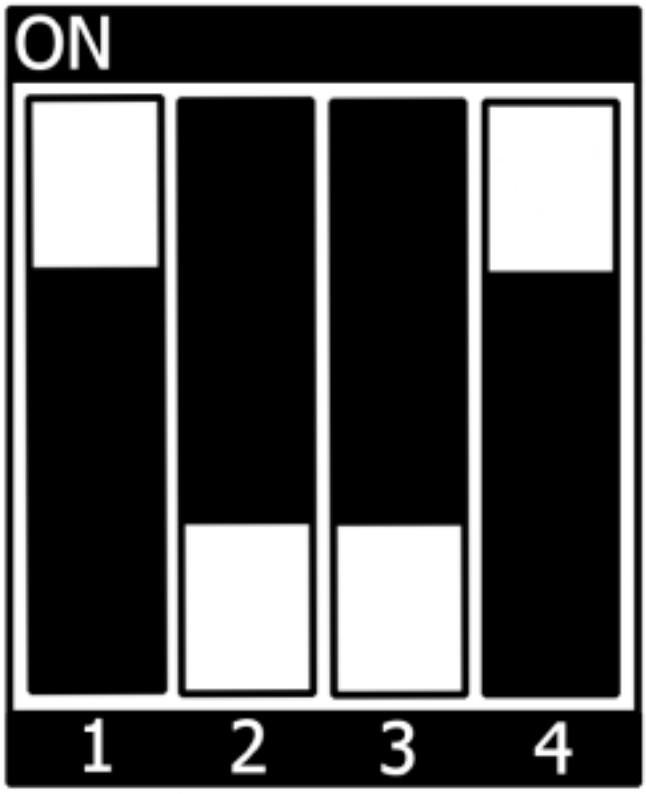
```
host:~$ xzcat phytec-qt6demo-image-imx95-libra-fpsc-1?(.rootfs).wic.xz | sudo dd of=/dev/<your_device>␣
↪bs=1M conv=fsync status=progress
```

再次确保将 <your_device> 替换为之前找到的设备名称。

参数 `conv=fsync` 强制在 `dd` 返回之前对设备进行 sync 操作。这确保所有数据块都已写入 SD 卡，而没有任何数据缓存在内存中。参数 `status=progress` 将打印出进度信息。

2.3 首次启动

- 要从 SD 卡启动，*bootmode switch (S1)* 需要设置为以下位置：



- 插入 SD 卡
- Connect the target and the host with **USB-C** on *(X14)* debug USB
- 给开发板通电

CHAPTER 3

编译 BSP

本节将指导您使用 Yocto 和 phyLinux 脚本进行 i.MX 95 BSP 的编译。更多有关 phytec meta-layer 和 Yocto 的信息，请访问：Yocto Reference Manual (scarthgap) 。

3.1 基本设置

如果您从未在您的主机上使用 Yocto 编译过 Phytec BSP，您应查看 Yocto Reference Manual (scarthgap) 中的 BSP Workspace 安装一节。

3.2 下载 BSP

获取 BSP 有两种方式。您可以从我们的下载页面下载完整的 BSP 镜像：BSP-Yocto-IMX95 ；您也可以使用 Yocto 下载 BSP 工程并编译。如果您想要对 BSP 进行修改，建议使用第二种方式。

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.

- 创建一个新的项目文件夹，获取 phyLinux 脚本，并赋予脚本具备可执行权限：

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

> **警告**
>
> 我们需要一个空的项目文件夹，phyLinux 首先会清理当前所在的工作目录。从一个不为空的目录下调用 phyLinux 将会产生告警。

- 运行 phyLinux：

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

备注

在首次初始化时，phyLinux 脚本会要求您在 `/usr/local/bin` 目录中安装 Repo 工具。

- 在执行 init 命令时，您需要选择您的处理器平台（SoC）、PHYTEC 的 BSP 版本号以及您正在使用的硬件。

备注

如果您无法根据菜单中提供的信息识别您的开发板，请查看产品的发票。并查看 our BSP 。

- 也可以通过命令行参数直接传递这些信息：

```
host:~/yocto$ DISTRO=ampliphy-vendor MACHINE=imx95-libra-fpsc-1 ./phyLinux init -p imx95 -r BSP-Yocto-
↪NXP-i.MX95-ALPHA1
```

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 开始构建

- 设置 Shell 环境变量：

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

备注

在每次打开新的用于编译的 shell 时，都需要先执行这一步骤。

- 当前的工作目录会变更为 build/。
- 打开主配置文件，同意并接受 GPU 和 VPU 二进制文件的许可证协议。通过取消注释相应的行来完成此操作，如下所示。

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

- 编译您的镜像：

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

备注

对于第一次编译，我们建议从我们的较小的非图形化镜像 phytec-headless-image 开始，以查看一切是否正常工作。

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

第一次构建过程在现代的 Intel Core i7 处理器上大约需要 40 分钟。后续的构建将使用本次编译产生的缓存，大约需要 3 分钟。

3.2.2 BSP 镜像

所有由 Bitbake 生成的镜像都放在 `~/yocto/build/deploy*/images/<machine>` 。例如以下列表是 imx95-libra-fpsc-1 machine 生成的所有文件：

- **u-boot.bin**: 编译后的 U-boot bootloader 二进制文件。不是最终镜像中的 bootloader!
- **oftree**: 默认内核设备树
- **u-boot-spl.bin**: 二级程序加载器 (SPL)
- **bl31-imx95.bin**: ARM Trusted Firmware binary
- **lpddr5_dmem_qb_v202311.bin, lpddr5_dmem_v202311.bin, lpddr5_imem_qb_v202311.bin, lpddr5_imem_v202311.bin**: DDR PHY firmware images
- **oei-m33-ddr.bin, oei-m33-tcm.bin**: OEI images
- **m33_image-mx95libra.bin**: System Manager image
- **imx-boot**：由 imx-mkimage 编译的 bootloader 镜像，包括 SPL、U-Boot、ARM 可信固件和 DDR 固件。这是最终的可引导 bootloader 镜像。
- **fitImage**: Linux 内核 FIT 镜像
- **fitImage-its*.its**
- **Image**: Linux 内核镜像
- **Image.config**: 内核 config 文件
- **imx95-libra-rdk-fpsc*.dtb**: Kernel device tree file
- **imx95-phycore-fpsc*.dtbo, imx95-libra-rdk-fpsc*.dtbo**: Kernel device tree overlay files
- **phytec-qt6demo-image*.tar.gz**: 根文件系统
- **phytec-qt6demo-image*.rootfs.wic.xz**: 压缩的 SD 卡镜像

CHAPTER 4

安装操作系统

4.1 启动模式开关 (S1)

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

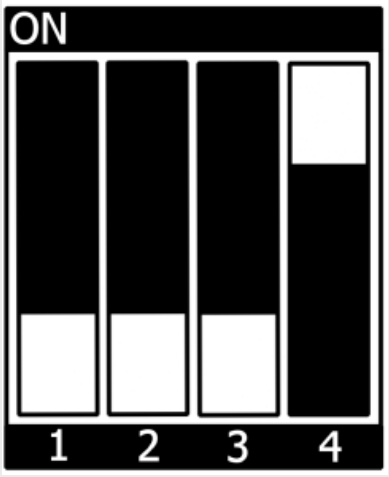


图 1: eMMC

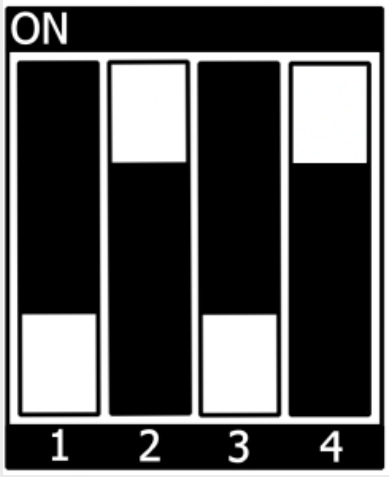


图 2: 内部 fuse

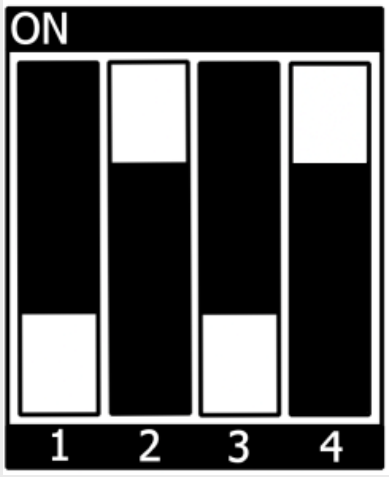


图 3: SPI NOR

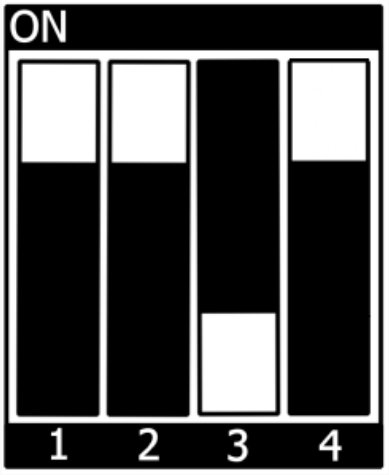


图 4: USB

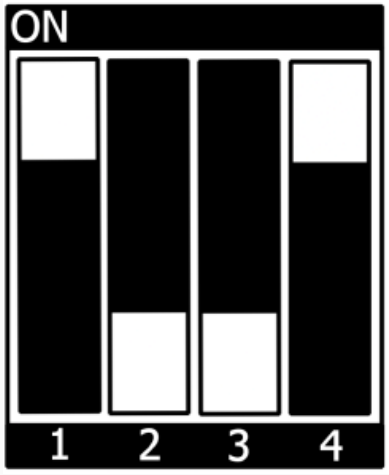


图 5: SD 卡

4.2 Flash e.MMC

为了保持文档的一致性和简洁性，假设已经配置好了 TFTP 服务器；所有生成的镜像（如上所列）都被复制到默认的/srv/tftp 目录。如果您没有进行设置，您需要修改路径到包含镜像的目录。有关如何设置 TFTP 服务器和目录的说明，请参见*Setup Network Host* 。

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

警告

When e.MMC 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
```

可以运行上述命令来检查系统启动在这种条件下是否会到影响。如果 `mmcblk2p1` 和 `mmcblk1p1` 具有相同

的 UUID，则会影响系统正确启动。

4.2.1 Flash e.MMC from Network

i.MX 95 boards have an Ethernet connector and can be updated over a network. Be sure to set up the development host correctly. The IP needs to be set to 192.168.3.10, the netmask to 255.255.255.0, and a TFTP server needs to be available. From a high-level point of view, an e.MMC device is like an SD card. Therefore, it is possible to flash the **WIC image** (`<name>.wic`) from the Yocto build system directly to the e.MMC. The image contains the bootloader, kernel, device tree, device tree overlays, and root file system.

Flash e.MMC via Network in Linux on Host

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

小技巧

需要保证设备和存储镜像的主机之间的网络正常！ *Setup Network Host*

查看主机上可用的镜像文件：

```
host:~$ ls /srv/tftp
phytec-qt6demo-image-imx95-libra-fpsc-1.rootfs.wic.xz
phytec-qt6demo-image-imx95-libra-fpsc-1.rootfs.wic.bmap
```

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

```
host:~$ scp /srv/tftp/phytec-qt6demo-image-imx95-libra-fpsc-1.rootfs.wic.* root@192.168.3.11:/tmp && ssh␣
↪root@192.168.3.11 "bmaptool copy /tmp/phytec-qt6demo-image-imx95-libra-fpsc-1.rootfs.wic.xz /dev/mmcblk2"
```

Flash e.MMC via Network in Linux on Target

You can update the e.MMC from your target.

小技巧

需要保证设备和存储镜像的主机之间的网络正常！ *Setup Network Host*

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

```
target:~$ scp <USER>@192.168.3.10:/srv/tftp/phytec-qt6demo-image-imx95-libra-fpsc-1.rootfs.wic.* /tmp &&␣
↪bmaptool copy /tmp/phytec-qt6demo-image-imx95-libra-fpsc-1.rootfs.wic.xz /dev/mmcblk2
```

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

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

小技巧

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

小技巧

需要保证设备和存储镜像的主机之间的网络正常！ *Setup Network Host*

解压缩您的镜像

```
host:~$ unxz /srv/tftp/phytec-headless-image-imx95-libra-fpsc-1.rootfs.wic.xz
```

通过网络将您的镜像加载到内存中：

- 使用 DHCP

```
u-boot=> dhcp phytec-headless-image-imx95-libra-fpsc-1.rootfs.wic
BOOTP broadcast 1
DHCP client bound to address 192.168.3.1 (1 ms)
Using ethernet@30be0000 device
TFTP from server 192.168.3.10; our IP address is 192.168.3.1
Filename 'phytec-headless-image-imx95-libra-fpsc-1.rootfs.wic'.
Load address: 0x40480000
Loading: #####################################
         #####################################
         #####################################
         ...
         ...
         ...
         #####################################
         #############
         11.2 MiB/s
done
Bytes transferred = 911842304 (36599c00 hex)
```

- 使用静态 IP 地址（必须先设置 serverip 和 ipaddr）。

```
u-boot=> tftp ${loadaddr} phytec-headless-image-imx95-libra-fpsc-1.rootfs.wic
Using ethernet@30be0000 device
TFTP from server 192.168.3.10; our IP address is 192.168.3.11
Filename 'phytec-headless-image-imx95-libra-fpsc-1.rootfs.wic'.
Load address: 0x40480000
Loading: #####################################
         #####################################
         #####################################
         ...
         ...
         ...
         #####################################
         #############
         11.2 MiB/s
done
Bytes transferred = 911842304 (36599c00 hex)
```

Write the image to the e.MMC:

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

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

4.2.2 Flash e.MMC U-Boot image via Network from running U-Boot

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

小技巧

需要保证设备和存储镜像的主机之间的网络正常！ *Setup Network Host*

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

```
u-boot=> tftp ${loadaddr} imx-boot
u-boot=> setexpr nblk ${filesize} / 0x200
u-boot=> mmc dev 2
u-boot=> mmc write ${loadaddr} 0x40 ${nblk}
```

提示

十六进制值表示偏移量，单位为 512 字节块的倍数。请参阅偏移表 以获取相应 SoC 的正确值。

4.2.3 Flash e.MMC from USB stick

Flash e.MMC from USB in Linux

These steps will show how to flash the e.MMC on Linux with a USB stick. You only need a complete image saved on the USB stick and a bootable WIC image. (e.g. phytec-qt6demo-image-imx95-libra-fpsc-1.|yocto-imageext|). Set the *bootmode switch (S1)* to SD card.

- 插入并挂载 U 盘:

```
[   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
```

- 现在查看您在 USB 优盘上保存的镜像文件:

```
target:~$ ls /mnt
phytec-qt6demo-image-imx95-libra-fpsc-1.rootfs.wic.xz
phytec-qt6demo-image-imx95-libra-fpsc-1.rootfs.wic.bmap
```

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

```
target:~$ bmaptool copy /mnt/phytec-qt6demo-image-imx95-libra-fpsc-1.rootfs.wic.xz /dev/mmcblk2
```

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

小技巧

在此之前，您需要将*bootmode switch (S1)* 配置为 **eMMC**。

4.2.4 Flash e.MMC from SD card

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

或者，使用 partup 包烧写 SD 卡，如*Getting Started* 中所述。这样就可使用 SD 卡的全部容量。

备注

There is no partup support in imx95 ALPHA1 release yet.

Flash e.MMC from SD card in Linux on Target

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

- 检查在 SD 卡上保存的 partup 包或 WIC 镜像文件：

```
target:~$ ls
phytec-qt6demo-image-imx95-libra-fpsc-1.rootfs.partup
phytec-qt6demo-image-imx95-libra-fpsc-1.rootfs.wic.xz
phytec-qt6demo-image-imx95-libra-fpsc-1.rootfs.wic.bmap
```

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

```
target:~$ partup install phytec-qt6demo-image-imx95-libra-fpsc-1.rootfs.partup /dev/mmcblk2
```

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

备注

另外，也可以使用 `bmaptool` 工具：

```
target:~$ bmaptool copy phytec-qt6demo-image-imx95-libra-fpsc-1.rootfs.wic.xz /dev/mmcblk2
```

请注意，在使用 `bmaptool` 烧写时，根文件系统分区并不会使用 eMMC 的最大容量。

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

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

Flash e.MMC from SD card in U-Boot on Target

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

- 将一个可用的镜像烧写到 SD 卡，并创建一个 EXT4 格式的第三分区。将 WIC 镜像（例如 phytec-qt6demo-image.rootfs.wic）复制到该分区。
- Configure the *bootmode switch (S1)* to SD card and insert the SD card.
- 打开电源并进入 U-Boot。
- 加载镜像：

```
u-boot=> ext4load mmc 1:3 ${loadaddr} phytec-headless-image-imx95-libra-fpsc-1.rootfs.wic
reading
911842304 bytes read in 39253 ms (22.2 MiB/s)
```

- Switch the mmc dev to e.MMC:

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

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

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

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

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

CHAPTER 5

开发

5.1 独立编译准备

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 ‘Git server at https://github.com/phytec.

5.1.1 Git 仓库

- 使用的 U-Boot 仓库：

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

- 我们的 U-Boot 基于 u-boot-phytec-imx 并添加了一些硬件相关的补丁。
- 使用的 Linux 内核仓库：

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

- 我们的 i.MX 95 内核是基于 linux-phytec-imx 内核。

要找出核心板应使用的 u-boot 和 kernel 版本对应的 git 仓库 tag 标签，请查看您的 BSP 源文件夹：

```
meta-phytec/recipes-kernel/linux/linux-phytec-imx_*.bb
meta-phytec/recipes-bsp/u-boot/u-boot-phytec-imx_*.bb
```

5.1.2 编译 SDK

您可以使用 Yocto 自行编译 SDK：

- 移动到 Yocto 的 build 目录：

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

5.1.3 安装 SDK

- 设置正确的权限并安装 SDK：

```
host:~$ chmod +x phytec-ampliphy-vendor-glibc-x86_64-phytec-qt6demo-image-cortexa55-crypto-toolchain-
↪5.0.x.sh
host:~$ ./phytec-ampliphy-vendor-glibc-x86_64-phytec-qt6demo-image-cortexa55-crypto-toolchain-5.0.x.sh
=================================================================================================================
Enter target directory for SDK (default: /opt/ampliphy-vendor/5.0.x):
You are about to install the SDK to "/opt/ampliphy-vendor/5.0.x". 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 使用 SDK

通过在工具链目录中 source *environment-setup* 文件来初始化您的 shell 交叉编译环境：

```
host:~$ source /opt/ampliphy-vendor/5.0.x/environment-setup-cortexa55-crypto-phytec-linux
```

5.1.5 安装所需工具

独立编译 Linux kernel 和 U-Boot 需要主机安装一些额外的工具。对于 Ubuntu，您可以使用以下命令安装它们：

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

5.2 单独编译 U-Boot

5.2.1 获取源代码

- 获取 U-Boot 源代码：

```
host:~$ git clone https://github.com/phytec/u-boot-phytec-imx
```

- 要获取正确的 *U-Boot* **tag**，您需要查看我们的 release notes，可以在这里找到：release notes
- 此版本中使用的 **tag** 称为 v2024.04_2.0.0-phy14
- 查看所需的 *U-Boot* **tag**：

```
host:~$ cd ~/u-boot-phytec-imx/
host:~/u-boot-phytec-imx$ git fetch --all --tags
host:~/u-boot-phytec-imx$ git checkout tags/v2024.04_2.0.0-phy14
```

- 设置编译环境：

```
host:~/u-boot-phytec-imx$ source /opt/ampliphy-vendor/5.0.x/environment-setup-cortexa55-crypto-phytec-
↪linux
```

5.2.2 编译 bootloader

- 编译 flash.bin (imx-boot)：

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

5.2.3 Build the imx-boot binary with imx-mkimage

Clone the repository

First clone the imx-mkimage git repository from NXP.

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

获取所需的二进制文件

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

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

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

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

Build the flash.bin binary

Go to the imx-mkimage folder and execute:

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

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

5.2.4 将 bootloader 烧写到块设备上

flash.bin 文件可以在 u-boot-phytec-imx/ 目录下找到，现在可以进行烧写。需要指定芯片特定的偏移量：

SoC	User 分区偏移量	Boot 分区偏移量	eMMC 设备
i.MX 95	32 kiB	0 kiB	/dev/mmcblk0

例如，烧写 SD 卡：

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

提示

如果您有我们的 BSP Yocto 工程代码，具体的偏移值也会在 Yocto 变量”BOOTLOADER_SEEK” 和”BOOTLOADER_SEEK_EMMC” 中声明。

5.3 单独编译内核

内核与设备树一起打包在 FIT 镜像中。U-Boot 已被配置为能够加载 FIT 镜像并引导其中包含的内核。因此，内核镜像必须打包在 FIT 镜像中。

5.3.1 配置源代码

- 使用的 linux-phytec-imx 分支可以在 release notes 中找到
- 此版本所需的标签称为 v6.6.52-2.2.0-phy13
- Check out 所需的 linux-phytec-imx 标签：

```
host:~$ git clone https://github.com/phytec/linux-phytec-imx
host:~$ cd ~/linux-phytec-imx/
host:~/linux-phytec-imx$ git fetch --all --tags
host:~/linux-phytec-imx$ git checkout tags/v6.6.52-2.2.0-phy13
```

- 为了提交更改，强烈建议切换到一个新分支：

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

- 设置编译环境：

```
host:~/linux-phytec-imx$ source /opt/ampliphy-vendor/5.0.x/environment-setup-cortexa55-crypto-phytec-
↪linux
```

5.3.2 编译内核

- 编译 Linux 内核：

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

- 安装内核模块，比如安装到 NFS 目录：

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

- 镜像可以在 ~/linux-phytec-imx/arch/arm64/boot/Image.gz 找到
- dtb 文件可以在 ~/linux-phytec-imx/arch/arm64/boot/dts/freescale/imx95-libra-rdk-fpsc.dtb 找到
- 要（重新）编译设备树和 -overlay 文件，只需运行

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

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

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

备注

如果您遇到以下编译问题：

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

确保您在主机系统上安装了 *"libyaml-dev"* 包：

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

5.3.3 将内核打包成 FIT 镜像

要简单地替换内核，您需要一个 `image tree source` (.its) 文件。如果您已经使用 Yocto 编译了我们的 BSP，可以从此处提到的目录获取 its 文件: *BSP Images* 或者您可以在这里下载该文件: https://download.phytec.de/Software/Linux/BSP-Yocto-i.MX95/BSP-Yocto-NXP-i.MX95-ALPHA1/images/ampliphy-vendor/imx95-libra-fpsc-1/

将.its 文件复制到当前工作目录，创建一个指向内核镜像的链接，并使用 mkimage 创建最终的 fitImage。

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

5.3.4 Copy FIT image and kernel modules to SD card

FIT 镜像以及内核 module 可以用以下方式复制到已挂载的 SD 卡上。

```
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

- 请按照 https://github.com/nxp-imx/mfgtools#linux 上的说明进行操作。
- 如果您要从源代码编译 UUU，请将其添加到 `PATH` 中:

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

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

- 设置 udev 规则（在 `uuu -udev` 中有详细说明）:

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

5.4.2 获取镜像

Download imx-boot from our server or get it from your Yocto build directory at build/deploy-ampliphy-vendor/images/imx95-libra-fpsc-1/. For flashing a wic image to e.MMC, you will also need phytec-qt6demo-image-imx95-libra-fpsc-1.rootfs.wic.

5.4.3 开发板准备

将*bootmode switch (S1)* 设置为 **USB 串行下载**。同时，将 USB 端口*X18 (upper connector)* 连接到主机。

5.4.4 通过 UUU 工具启动 bootloader

执行并给开发板上电：

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

您可以像往常一样通过*(X14)* 在终端上查看启动日志。

备注

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.

5.5 主机网络准备

为了在 bootloader 中执行涉及网络的各种任务，需要配置一些主机服务。在开发主机上，必须安装和配置 TFTP、NFS 和 DHCP 服务。启动以太网所需的工具如下：

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

5.5.1 TFTP 服务设置

- 首先，创建一个目录来存储 TFTP 文件：

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

- 然后将您的 BSP 镜像文件复制到此目录，并确保 other 用户也对 tftp 目录中的所有文件具有读取权限，否则将无法从开发板访问这些文件。

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

- 您还需要为相应的接口配置一个静态 IP 地址。PHYTEC 开发板的默认 IP 地址是 192.168.3.11。可以将主机地址设置为 192.168.3.10，子网掩码为 255.255.255.0

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

  将 <network-interface> 替换为连接到开发板的网络接口。您可以通过不指定网络接口来显示所有可选网络接口。

- 返回的结果应包含以下内容：

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

- 创建或编辑 `/etc/default/tftpd-hpa` 文件：

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

  TFTP_USERNAME="tftp"
  ```

(续下页)

(接上页)

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

- 将 TFTP_DIRECTORY 设置为您的 TFTP 服务器根目录
- 将 TFTP_ADDRESS 设置为 TFTP 服务监听的主机地址（设置为 0.0.0.0:69 以监听 69 端口上所有 IP）。
- 设置 TFTP_OPTIONS，以下命令显示可配置的选项：

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

- 重新启动服务以应用配置更改：

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

现在将开发板的以太网端口连接到您的主机。我们还需要在开发板和运行 TFTP 服务的主机之间建立网络连接。TFTP 服务器的 IP 地址应设置为 192.168.3.10，子网掩码为 255.255.255.0。

NFS 服务器设置

- 创建一个 NFS 目录：

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

- NFS 服务对文件共享的路径没有限制，因此在大多数 linux 发行版中，我们只需修改文件 `/etc/exports`，并将我们的根文件系统共享到网络。在这个示例文件中，整个目录被共享在主机地址为 192.168.3.10 的 IP 地址上。注意这个地址需要根据本地情况进行调整：

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

- 现在 NFS 服务器需要再次读取 `/etc/exportfs` 文件：

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

DHCP 服务器设置

- 创建或编辑 `/etc/kea/kea-dhcp4.conf` 文件；以内部子网为例，将 <network-interface> 替换为物理网络接口的名称：

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

(续下页)

(接上页)

```
        "pools": [
          { "pool": "192.168.3.1 - 192.168.3.255" }
        ]
    }]
  }
}
```

警告

在创建子网时请小心，因为这可能会扰乱公司网络政策。为了安全起见，请使用不同的子网，并通过 `interfaces` 配置选项指定该网络。

- 现在 DHCP 服务需要重新读取 `/etc/kea/kea-dhcp4.conf` 文件：

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

当您启动/重启主机时，如果 kea-dhcp4 配置中指定的网络接口未处于活动状态，kea-dhcp4-server 将无法启动。因此请确保在连接接口后启动或者重启该 systemd 服务。

备注

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`.

警告

Using netboot with standardboot and static IPs does not work yet in this release. Standardboot will always use dhcp.

5.6 从网络启动内核

从网络启动意味着通过 TFTP 加载内核和设备树，并通过 NFS 加载根文件系统。但 bootloader 需要从另外的的启动设备加载。

5.6.1 在主机上放置网络启动的镜像

- 将内核 fitimage 复制到您的 tftp 目录中：

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

- 将启动脚本复制到您的 tftp 目录中：

```
host:~$ cp boot.scr.uimg /srv/tftp
```

- 确保 other 用户对 tftp 目录中的所有文件具有读取权限，否则将无法从开发板访问它们：

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

- 将根文件系统解压到您的 NFS 目录中：

```
host:~$ sudo tar -xvzf phytec-qt6demo-image-imx95-libra-fpsc-1.rootfs.tar.gz -C /srv/nfs
```

备注

请确保使用 sudo 执行命令，以保留根文件系统中文件的所属权限。

5.6.2 设置网络启动的 bootenv.txt 文件

在您的 tftp 目录中创建一个 bootenv.txt 文件，并将以下变量写入其中。

```
nfsroot=/srv/nfs
overlays=<overlayconfignames>
```

<overlayconfignames> 替换为您想要使用的设备树 overlay 配置名称。用 # 号分隔配置名称。例如：

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

小技巧

所有支持的设备树 overlay 文件都在*device tree* 章节中。或者可以通过以下命令打印：

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

5.6.3 开发板上的网络设置

如果要自定义开发板上的以太网配置，请按照此处的说明进行操作：*Network Environment Customization*

5.6.4 从开发板启动

将开发板启动到 U-boot，按任意键暂停。

- 要从网络启动，请运行：

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

5.7 获取 BSP 开发中版本

5.7.1 当前 release 的开发中版本

这些 release manifest 文件是为了让您访问 *Yocto* BSP 的开发版本。它们不会在 phyLinux 选择菜单中显示，需要手动选择。可以使用以下命令行来完成此操作：

```
host:~$ ./phyLinux init -p imx95 -r BSP-Yocto-NXP-i.MX95-PD25.1.y
```

这将初始化一个 BSP，用于跟踪当前版本（BSP-Yocto-NXP-i.MX95-ALPHA1 ）的最新开发版本。从现在开始，在此文件夹中执行 *repo sync* 将从我们的 Git 仓库中拉取所有最新的更改：

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

5.7.2 即将发布版本的开发中版本

即将发布版本的开发中版本可以通过这种方式访问。请执行以下命令，并查找一个比最新版本（BSP-Yocto-NXP-i.MX95-ALPHA1 ）的 PDXX.Y 数字更高的版本，并且以 .y 结尾：

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

5.8 Format SD card

使用单一的 SD 卡启动盘对存储介质进行烧写是开发过程中的常见任务。本章节针对此场景提供基础说明。大多数镜像的大小超过了默认的 root 分区剩余容量。要使用 SD 卡进行烧写，根文件系统需要扩展或创建一个单独的分区。有几种不同的方法可以格式化 SD 卡。最简单的方法是使用 Gparted。

5.8.1 Gparted

- 获取 GParted：

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

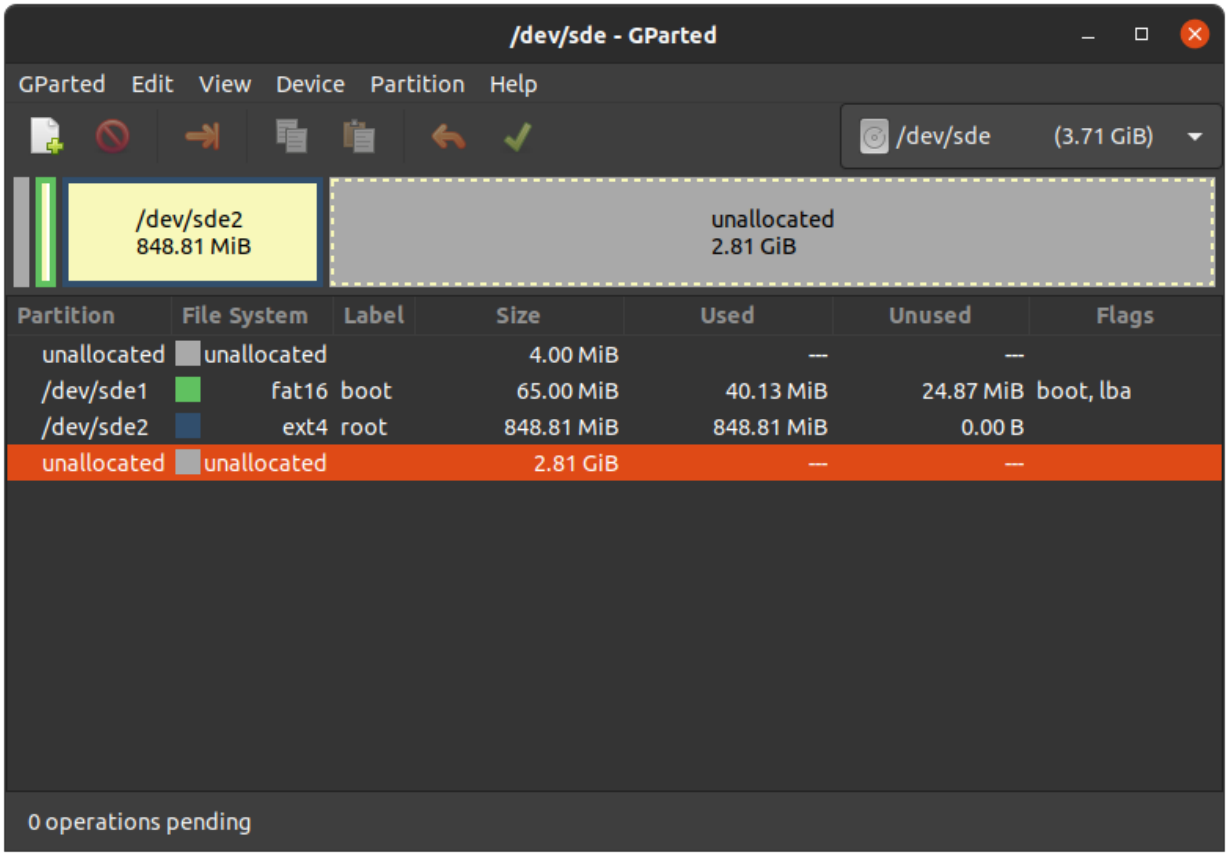
- Insert the SD card into your host and get the device name:

  ```
  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.

- 启动 GParted：

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

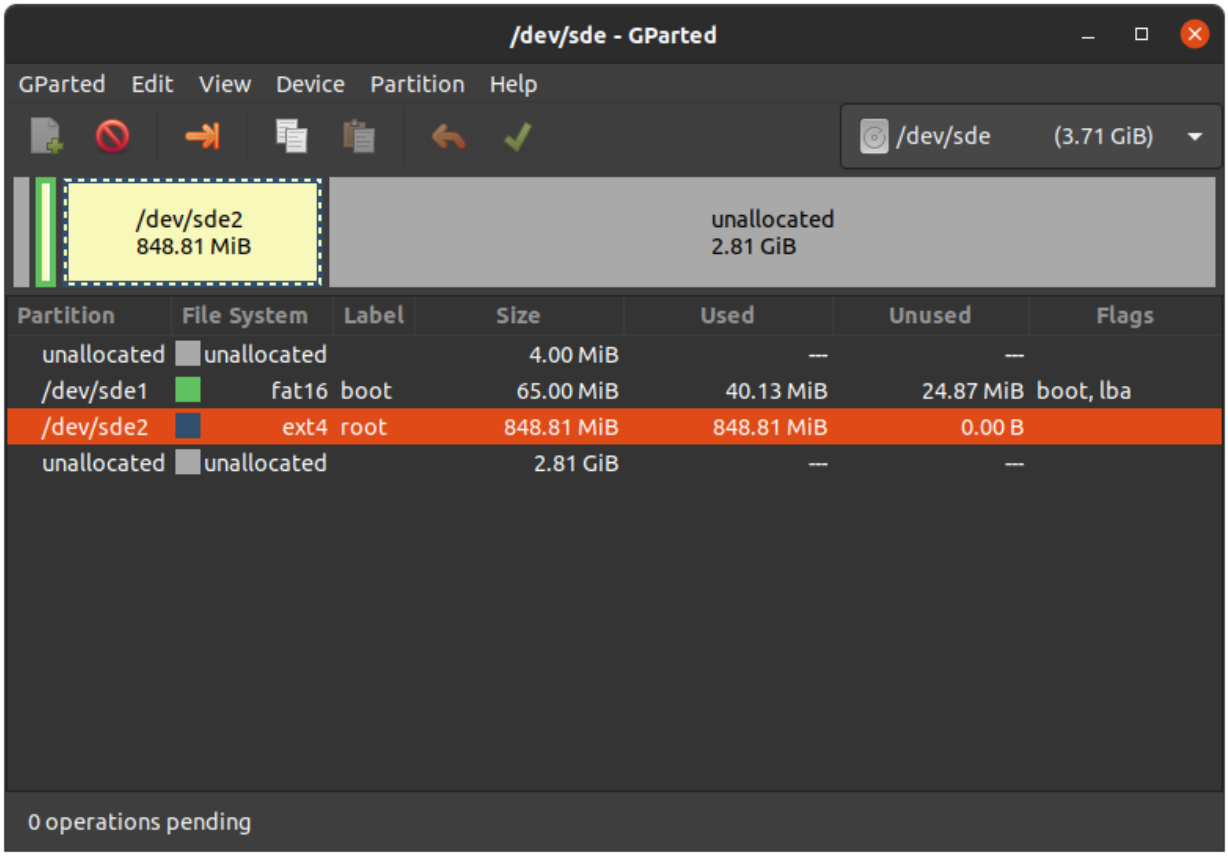


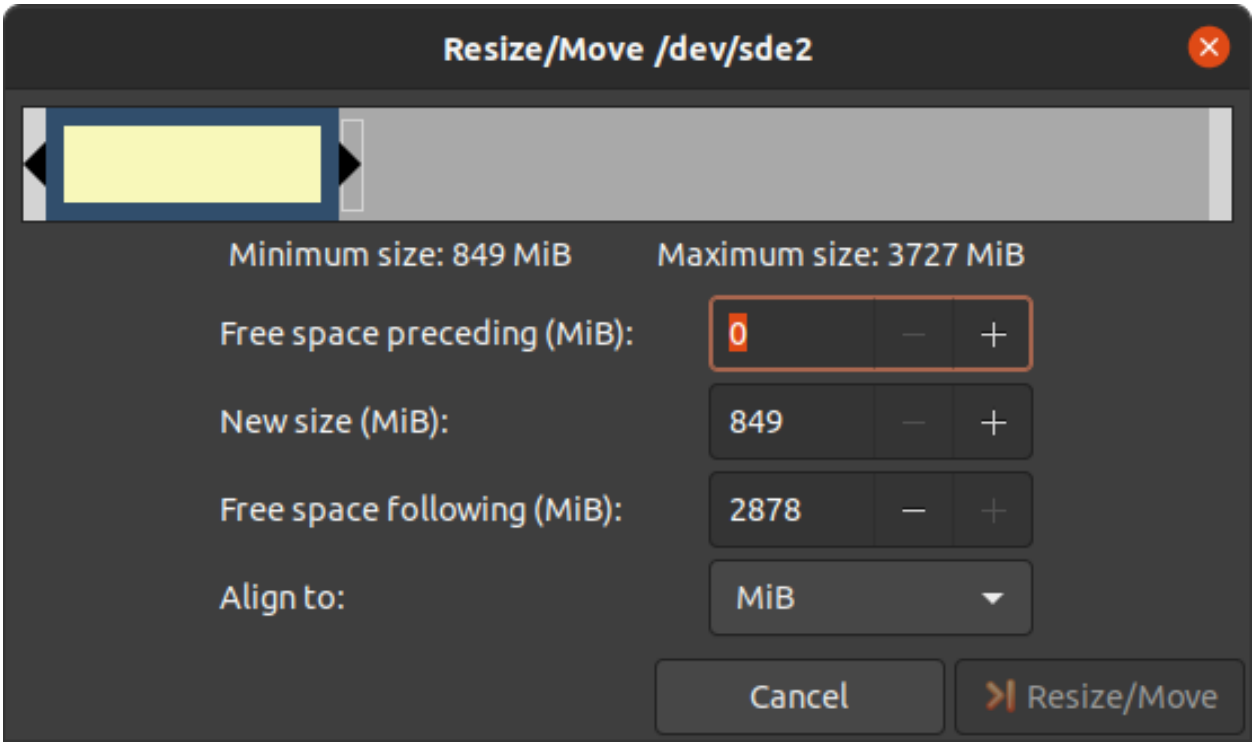
扩展根文件系统

警告

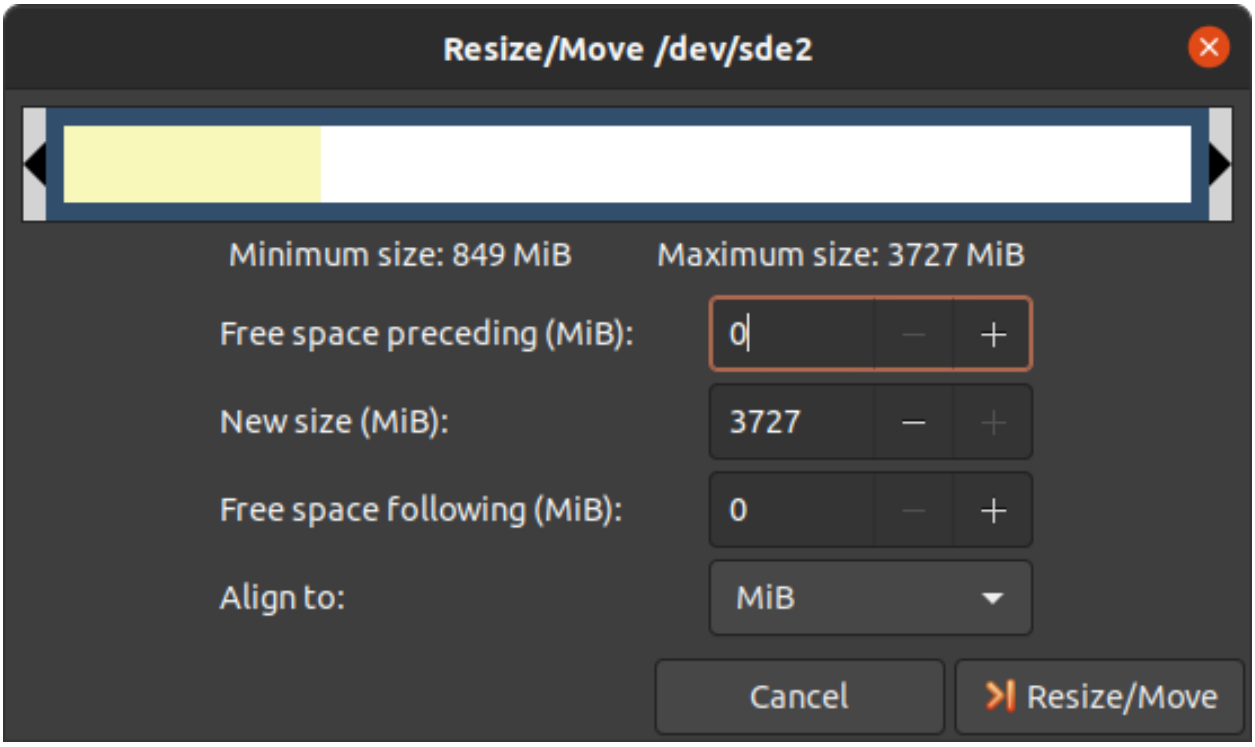
使用 resize2fs 版本 1.46.6 及更早版本的 PC 系统（例如 Ubuntu 22.04）无法烧写在 Mickledore 以及更新的 yocto 版本上创建的 partup 软件包。这个是因为 resize2fs 新增了默认选项而导致的兼容性问题。有关详细信息，请参阅 发布说明 。

- Choose your SD card device at the drop-down menu on the top right
- 选择 ext4 根分区并点击调整大小:

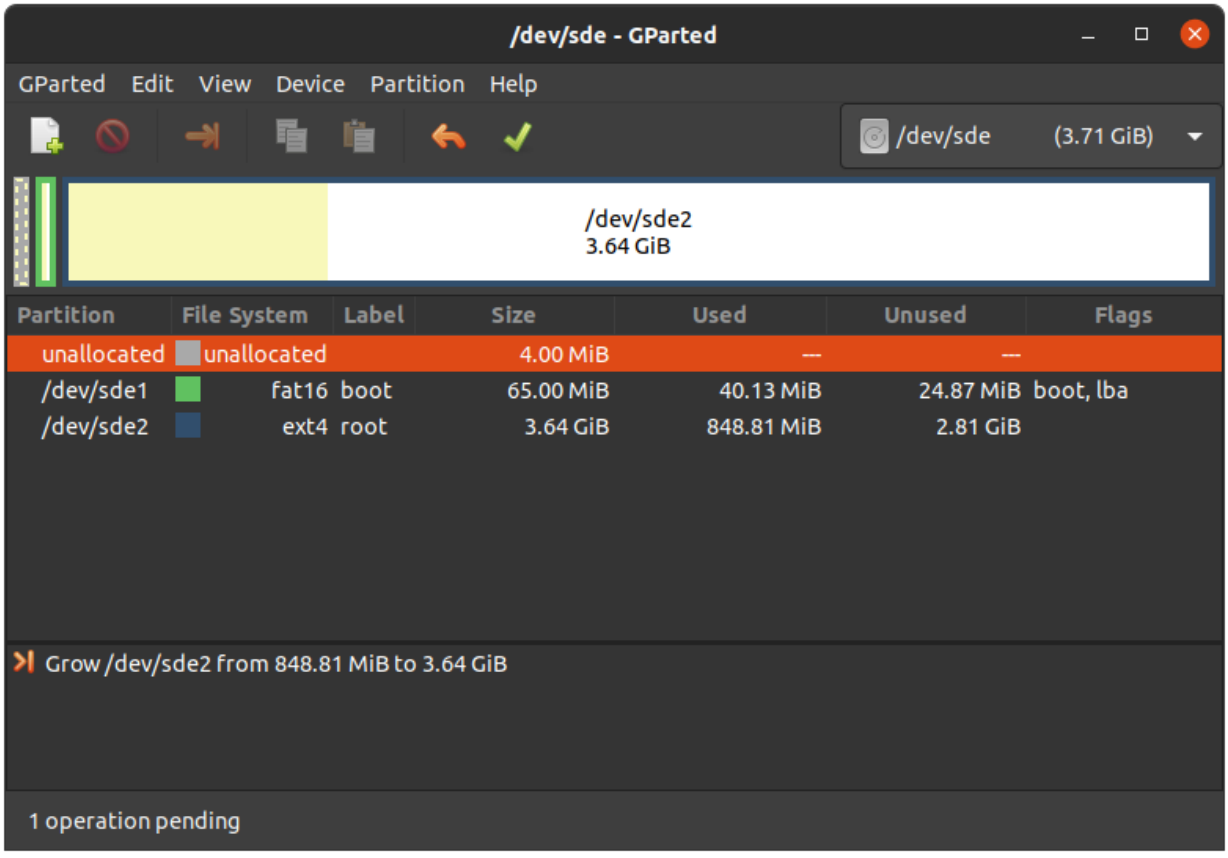




- 您可以根据需要拖动滑块或手动输入大小。



- 通过点击“Change Size”按钮确认您的输入。



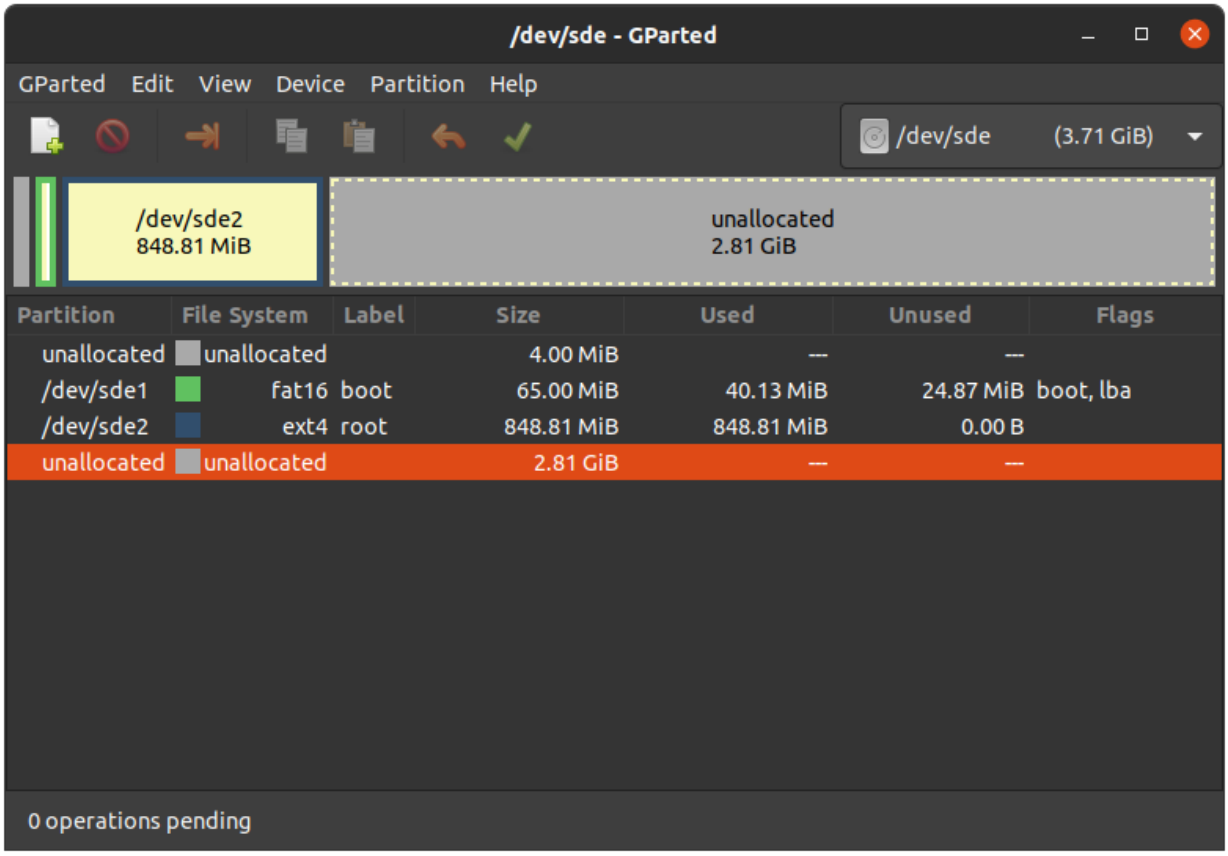
- 要应用您的更改，请按绿色勾号。

- 现在您可以挂载根分区并将 phytec-qt6demo-image-imx95-libra-fpsc-1.wic 镜像复制到其中。然后再卸载它：

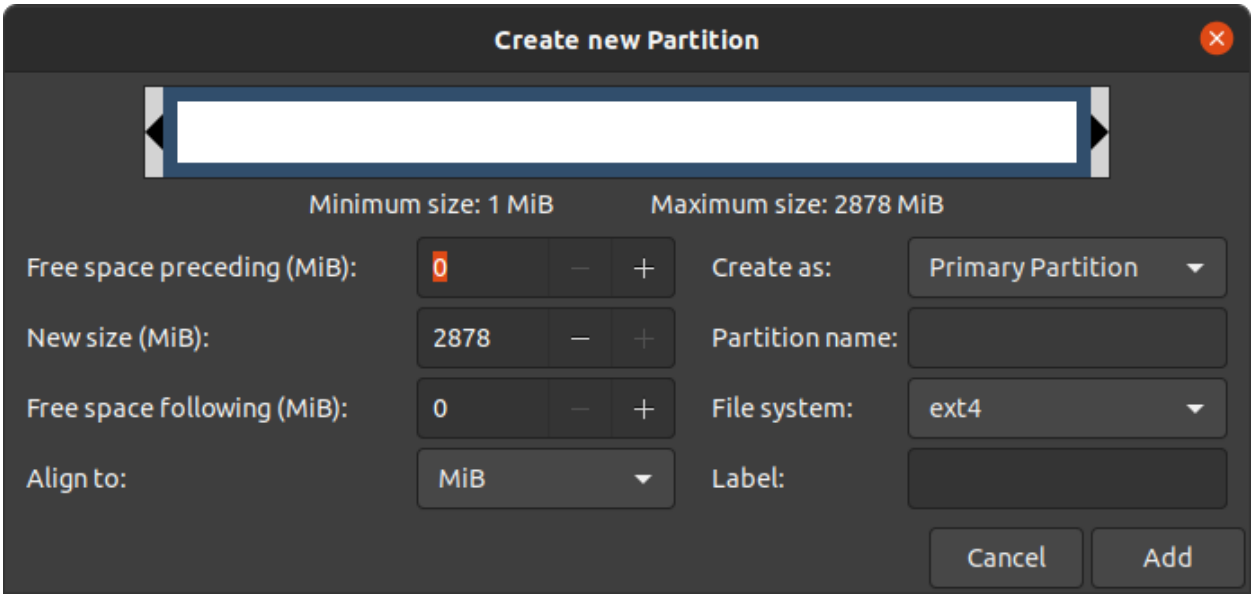
```
host:~$ sudo cp phytec-qt6demo-image-imx95-libra-fpsc-1.wic /mnt/ ; sync
host:~$ umount /mnt
```

创建第三个分区

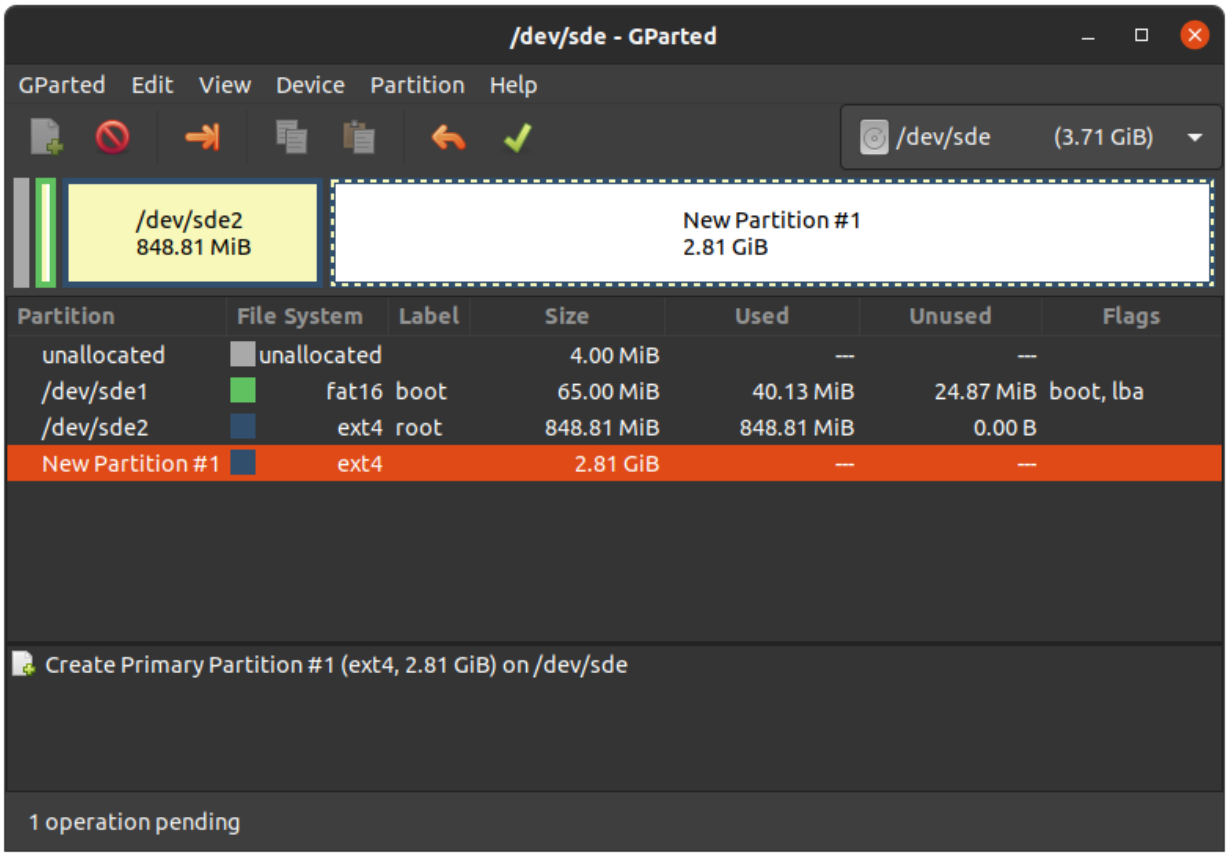
- Choose your SD card device at the drop-down menu on the top right



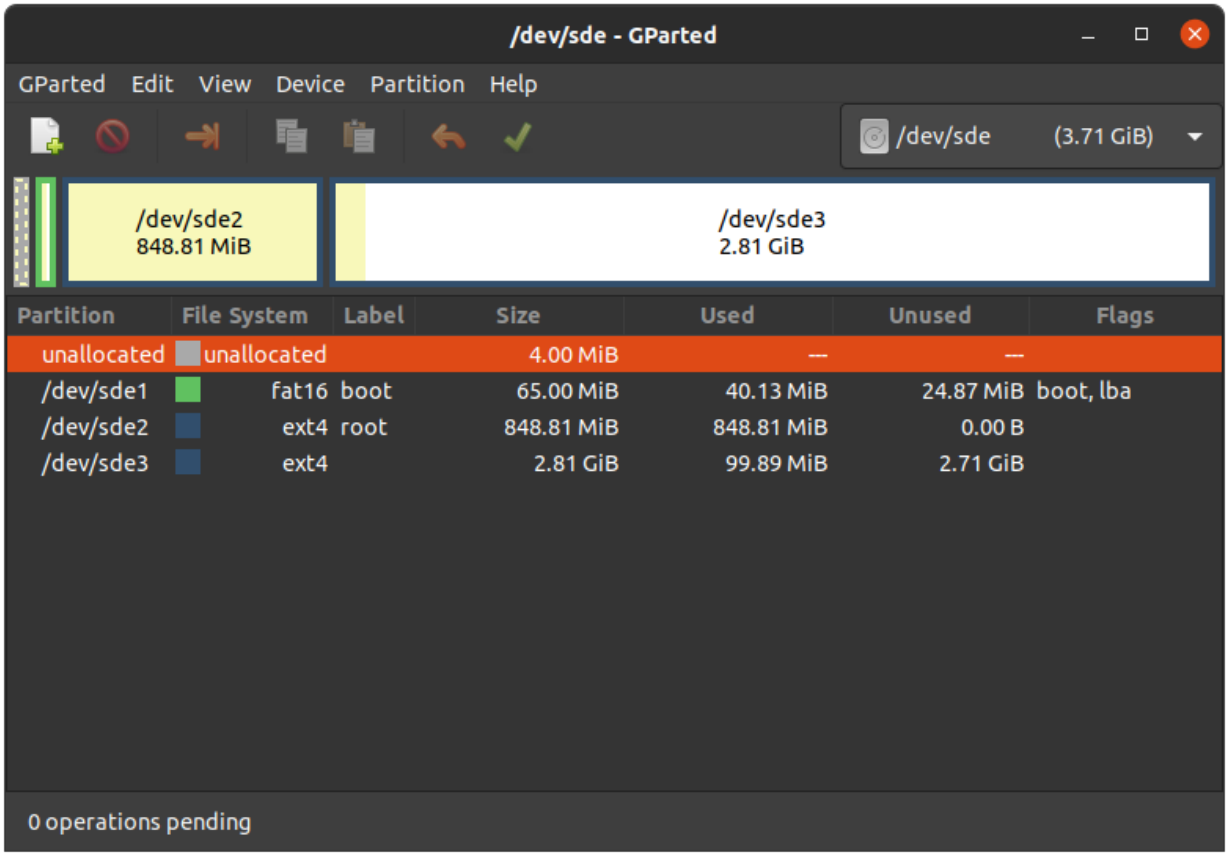
- 选择更大的未分配区域，然后点击”New”：



- 点击”Add”



- 按绿色勾确认更改。



- 现在您可以挂载新的分区并将 phytec-qt6demo-image-imx95-libra-fpsc-1.wic 镜像复制到其中。然后卸载它：

```
host:~$ sudo mount /dev/sde3 /mnt
host:~$ sudo cp phytec-qt6demo-image-imx95-libra-fpsc-1.wic /mnt/ ; sync
host:~$ umount /mnt
```

CHAPTER 6

设备树 (DT)

6.1 介绍

以下文本简要描述了设备树，关于设备树的相关文档可以在 Linux kernel 文档中找到（https://docs.kernel.org/devicetree/usage-model.html）。

“Open Firmware Device Tree” 或简称设备树（DT）是一种用于描述硬件的数据结构和语言。更具体地说，它是一个可由操作系统读取的硬件描述，以便操作系统不需要对 machine 的细节进行硬编码

内核文档是学习设备树的一个非常好的资源。关于设备树数据格式的概述可以在 devicetree.org 的设备树使用页面找到。

6.2 PHYTEC i.MX 95 BSP 设备树概念

以下部分说明了 PHYTEC 配置基于 i.MX 95 的核心板设备树的一些规则。

6.2.1 设备树结构

- *Module.dtsi* - 文件包括所有安装在核心板上的设备，例如 PMIC 和 RAM。
- *Board.dts* - 包含核心板 dtsi 文件。从 SoC i.MX 95 引出并在底板使用的设备也包含在此 dts 中。
- *Overlay.dtso* - 根据核心板或底板上可选硬件（例如 SPI 闪存或 PEB-AV-10）的情况来启用/禁用一些功能。

在 Linux 内核的根目录下，我们的 i.MX 9 平台的设备树文件可以在 `arch/arm64/boot/dts/freescale/` 找到。

6.2.2 设备树 Overlay

设备树 Overlay 是可以在启动时合并到设备树中的设备树片段。下面是扩展板的硬件描述。对比源码中的 include，overlay 通过覆盖的方式来生效。overlay 也可以根据实际开发板的硬件配置来设置设备树节点状态。设备树 Overlay 与我们 Linux 内核仓库中的其他设备树文件一起放在子文件夹 `arch/arm64/boot/dts/freescale/` 中。

imx95-libra-fpsc-1.conf 可用的 overlay 文件有：

```
imx95-libra-rdk-fpsc-lvds-ph128800t006-zhc01.dtbo
imx95-libra-rdk-fpsc-neoisp.dtbo
imx95-libra-rdk-fpsc-vm016-csi1.dtbo
imx95-libra-rdk-fpsc-vm016-fpdlink-port0-csi1.dtbo
imx95-libra-rdk-fpsc-vm016-fpdlink-port1-csi1.dtbo
imx95-libra-rdk-fpsc-vm016-csi2.dtbo
imx95-libra-rdk-fpsc-vm016-fpdlink-port0-csi2.dtbo
imx95-libra-rdk-fpsc-vm016-fpdlink-port1-csi2.dtbo
imx95-libra-rdk-fpsc-vm017-csi1.dtbo
imx95-libra-rdk-fpsc-vm017-fpdlink-port0-csi1.dtbo
imx95-libra-rdk-fpsc-vm017-fpdlink-port1-csi1.dtbo
imx95-libra-rdk-fpsc-vm017-csi2.dtbo
imx95-libra-rdk-fpsc-vm017-fpdlink-port0-csi2.dtbo
imx95-libra-rdk-fpsc-vm017-fpdlink-port1-csi2.dtbo
imx95-libra-rdk-fpsc-vm020-csi1.dtbo
imx95-libra-rdk-fpsc-vm020-fpdlink-port0-csi1.dtbo
imx95-libra-rdk-fpsc-vm020-fpdlink-port1-csi1.dtbo
imx95-libra-rdk-fpsc-vm020-csi2.dtbo
imx95-libra-rdk-fpsc-vm020-fpdlink-port0-csi2.dtbo
imx95-libra-rdk-fpsc-vm020-fpdlink-port1-csi2.dtbo
```

可以在 linux 中使用以下命令来列出 FIT 镜像中所有的 overlay 配置

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

或者在 u-boot：

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

可以在 Linux 或 U-Boot 环境下配置 overlay。overlay 是在引导命令调用后、内核加载之前生效。接下来的部分将更详细地解释配置方法。

设置 ${overlays} 变量

${overlays} U-Boot 环境变量包含一个以井号 # 分隔的 overlay 文件列表，这些 overlay 文件将在启动过程中应用。overlays 变量中的 overlay 列表会包含在 FIT 镜像中。在 $KERNEL_DEVICETREE 这个 Yocto machine 变量中设置的 overlay 文件将自动添加到 FIT 镜像中。

${overlays} 变量可以直接在 U-Boot 环境中设置，也可以作为外部 bootenv.txt 环境文件的一部分。当希望使用在 U-Boot 环境中手工配置的 overlay 变量，可以配置 env set no_bootenv 1 ，因为 overlay 变量可能在运行启动脚本时被覆盖。默认情况下，${overlays} 变量来自位于启动分区的 bootenv.txt 文件。您可以在已启动的开发板上从 Linux 读取和写入该文件：

```
target:~$ cat /boot/bootenv.txt
overlays=conf-imx95-libra-rdk-fpsc-peb-eval-01.dtbo#conf-|dtbo-peb-av-10|
```

更改将在下次重启后生效。如果没有可用的 bootenv.txt 文件，可以直接在 U-Boot 环境中设置 overlay 变量。

```
u-boot=> setenv overlays conf-|dtbo-peb-av-10|
u-boot=> printenv overlays
overlays=conf-|dtbo-peb-av-10|
u-boot=> boot
```

如果用户定义了 ${overlays} 变量，同时存在 bootenv.txt 文件，则需要设置 ${no_bootenv} 变量：

```
u-boot=> setenv no_bootenv 1
u-boot=> setenv overlays conf-|dtbo-peb-av-10|
u-boot=> boot
```

有关环境的更多信息，请参见 U-boot External Environment subsection of the *device tree overlay section*。

我们使用 `${overlays}` 变量来描述在运行时无法自动检测的扩展板和摄像头。如果想禁用 `${overlays}` 变量中列出的 overlay，可以在 U-Boot 的环境中将 overlay 变量 unset，并且将 `${no_overlays}` 变量设置为 *1*。

```
u-boot=> env delete overlays
u-boot=> env set no_bootenv 1
```

如果希望通过 bootenv.txt 文件设置除 overlay 之外的 U-Boot 变量并且禁用 overlay，可以从 bootenv.txt 文件中删除 overlay 定义行，而不是设置 no_bootenv。

不同的 SoM 配置

核心板会自动加载额外的 overlay，以禁用核心板未贴装的组件。通过读取出厂 EEPROM 数据（EEPROM SoM Detection）来实现自动加载对应的 overlay。

核心板型号会决定是否应用设备树 overlay。要在 U-Boot 环境中查询是否会应用某个 overlay，请运行：

```
u-boot=> env print fit_extensions
```

如果没有可用的 EEPROM 数据，则不加载任何设备树 overlay。

要禁止启动时自动加载不同核心板的 overlay，可以在 bootloader 环境中将 `${no_extensions}` 变量设置为 *1*：

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

6.2.3 U-boot 外部环境

在 Linux 内核启动时，外部环境 `bootenv.txt` 文本文件将从 MMC 设备的 boot 分区或通过 TFTP 加载。该文件的主要目的是存储 `${overlays}` 变量。这可以针对不同的 machine 在 Yocto 中预定义不同的 overlay 配置。文件的内容在 meta-phytec 中的 Yocto recipe 中的 bootenv 中定义：https://git.phytec.de/meta-phytec/tree/recipes-bsp/bootenv?h=scarthgap

该文件中也可以设置其他变量。这些变量将覆盖环境中现有的设置。但只有对 boot 命令后进行计算的变量生效，例如 `${nfsroot}` 或 `${mmcargs}`。在文件中更改其他变量将不会有作用。以网络启动的用法作为示例。

如果无法加载外部环境，启动过程将继续进行，并使用自带的环境变量值。

6.2.4 在 Linux 环境下更改开发板上的 U-boot 环境变量

Libubootenv 是我们镜像中包含的一个工具，用于在开发板 linux 上修改 U-Boot 环境。

使用以下命令打印 U-Boot 环境：

```
target:~$ fw_printenv
```

使用以下命令修改 U-Boot 环境：

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

小心

Libubootenv 会读取配置文件中配置的环境变量。要修改的环境变量会被插入到该文件中，默认情况下使用 eMMC 中存储环境变量。

如果 eMMC 没有被烧写过或者 eMMC 环境被擦除，libubootenv 将无法工作。您应该修改 `/etc/fw_env.config` 文件，以匹配您想要使用的环境源。

CHAPTER 7

访问外设

To find out which boards and modules are supported by the release of PHYTEC's phyCORE-i.MX 95 FPSC BSP described herein, visit our BSP web page and click the corresponding BSP release in the download section. Here you can find all hardware supported in the columns "Hardware Article Number" and the correct machine name in the corresponding cell under "Machine Name".

为了最大化软件的可复用性，Linux 内核提供了一个巧妙的软件架构，软件会根据不同硬件组件来分层。BSP（板级支持包）尽可能地对套件的功能进行模块化。当定制开发板或自定义核心板时，大部分软件配置可以简单的复制粘贴。与具体的开发板相关的内核代码可以在内核代码仓库中的设备树（DT）中找到，路径为 `arch/arm64/boot/dts/freescale/*.dts` 。

实际上，软件复用是 Linux 内核最重要的特性之一，尤其是在 ARM 架构中，它必须应对大量复杂且不同的系统级芯片（SoC）。整个开发板的硬件在设备树（DT）中描述，独立于内核镜像。硬件描述在一个单独的二进制文件中，称为设备树二进制文件（Device Tree Blob，DTB）（参见*device tree*）。

请阅读 PHYTEC i.MX 95 BSP 设备树概念部分，以了解我们的 i.MX 9 BSP 设备树模型。

以下部分概述了 i.MX 9 平台上支持的硬件组件及其对应操作系统驱动程序。客户可以根据自身的需求进行更改。

7.1 i.MX 95 引脚复用

该 i.MX 95 Soc 包含许多外设接口。为了在保持最大功能性的同时减少封装尺寸和降低整体系统成本，许多 i.MX 95 引脚可以多路复用为多达八种信号功能。尽管存在许多可能的引脚多路复用组合，但由于时序限制，只有一定数量的组合被称为有效的 IO 集合。这些有效的 IO 集合经过精心挑选，以为用户提供尽可能多的应用场景。

请参考我们的硬件手册或 NXP i.MX 95 参考手册，以获取有关特定引脚和复用能力的更多信息。

IO 集合的配置，也称为复用（muxing），是在设备树中完成的。驱动程序 pinctrl-single 读取设备树的节点 fsl,pins，并进行引脚复用配置。

The following is an example of the pin muxing of the lpuart7 device in imx95-phycore-fpsc.dtsi:

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

The first part of the string IMX95_PAD_GPIO_IO37___LPUART7_RX names the pad (in this example IMX95_PAD_GPIO_IO37). The second part of the string (LPUART7_RX) is the desired muxing option for this pad. The pad setting value (hex value on the right) defines different modes of the pad, for example, if internal pull resistors are activated or not. In this case, the internal resistors are enabled.

The device tree representation for UART1 pinmuxing: https://github.com/phytec/linux-phytec-imx/tree/v6.6.52-2.2.0-phy13/arch/arm64/boot/dts/freescale/imx95-phycore-fpsc.dtsi#L438

7.2 网络

Libra FPSC-i.MX 95 provides three ethernet interfaces. A gigabit Ethernet is provided by our module and board. Additionally there is a 10Gbit Ethernet. Currently only the one Gigabit Ethernet ports are supported (ETH0 and ETH1).

所有接口都提供一个标准的 Linux 网络端口，可以使用 BSD 套接字接口进行编程。整个网络配置由 systemd-networkd 守护进程管理。相关的配置文件可以在开发板的 `/lib/systemd/network/` 目录中找到，以及在 BSP 中的 `meta-ampliphy/recipes-core/systemd/systemd-conf` 目录中。

IP addresses can be configured within *.network files. The interfaces are configured to static IP as default. The default IP address and netmask for eth0 is:

```
eth0: 192.168.3.11/24
```

To configure eth0 to dynamic IP over DHCP, go to `/lib/systemd/network/\*-eth0.network` and delete the line:

```
Address=192.168.3.11/24
```

The DT Ethernet setup might be split into two files depending on your hardware configuration: the module DT and the board-specific DT. The device tree set up for the ethernet where the PHY is populated on the SoM can be found here: https://github.com/phytec/linux-phytec-imx/tree/v6.6.52-2.2.0-phy13/arch/arm64/boot/dts/freescale/imx95-phycore-fpsc.dtsi#L229.

The device tree set up for the ethernet where the PHY is populated on the Libra FPSC can be found here: https://github.com/phytec/linux-phytec-imx/tree/v6.6.52-2.2.0-phy13/arch/arm64/boot/dts/freescale/imx95-libra-rdk-fpsc.dts#L109.

7.2.1 网络配置

U-boot 网络环境

- 要在 bootloader 中查找以太网设置：

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

- 在将主机设置为 IP 192.168.3.10 和子网掩码 255.255.255.0 的情况下，开发板应该返回：

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

(续下页)

(接上页)

```
serverip=192.168.3.10
netmask=255.225.255.0
```

- 如果您需要进行任何更改：

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

<parameter> 应该是 ipaddr、netmask、gatewayip 或 serverip 中的一个。<value> 将是所选参数的设定值。

- 您所做的更改目前是临时的。要保存这些更改：

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

在这里，您也可以将 IP 地址更改为 DHCP，而不是使用静态 IP 地址。

- 配置：

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

- 设置 TFTP 和 NFS 的路径。修改可以如下所示：

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

请注意，这些修改只会影响 bootloader 的设置。

小技巧

像 nfsroot 和 netargs 这样的变量可以被 U-boot 外部环境重新赋值。对于网络启动，外部环境将通过 tftp 加载。例如，要在 `bootenv.txt` 文件中设置 nfsroot 变量，请在 tftproot 目录修改：

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

无需在开发板上存储这些信息。请注意，U-boot 外部环境对于像 ipaddr 或 serveraddr 这样的变量不起作用，因为它们在加载外部环境之前已经被设置完成。

内核网络环境

- 在开发板中查找 eth0 的以太网设置：

```
target:~$ ip -statistics address show eth0
2: eth0: <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
```

- 临时调整 eth0 的配置：

```
target:~$ ip address add 192.168.3.11/24 dev eth0
```

7.3 SD card

The i.MX 95 supports a slot for Secure Digital cards to be used as general-purpose block devices. These devices can be used in the same way as any other block device.

警告

这些设备是热插拔的。然而，您必须确保在设备仍然挂载时不要拔掉它。这可能会导致数据丢失！

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> 作为分区编号，从 1 开始计数，直到该设备的最大分区数量。分区可以使用任何类型的文件系统进行格式化，并且可以以标准方式进行处理，例如，可以使用 mount 和 umount 命令进行分区挂载和卸载。

小技巧

这些分区设备节点要求 SD 卡包含有效的分区表（类似于“硬盘”）。如果没有分区表，则整个设备作为一个文件系统使用（类似于“软盘”）。在这种情况下，必须使用 /dev/mmcblk1 进行格式化和挂载。卡始终以可写方式挂载。

DT configuration for the MMC (SD card slot) interface can be found here: https://github.com/phytec/linux-phytec-imx/tree/v6.6.52-2.2.0-phy13/arch/arm64/boot/dts/freescale/imx95-phycore-fpsc.dtsi#L632

DT configuration for the eMMC interface can be found here: https://github.com/phytec/linux-phytec-imx/tree/v6.6.52-2.2.0-phy13/arch/arm64/boot/dts/freescale/imx95-phycore-fpsc.dtsi#L619

7.4 e.MMC Devices

PHYTEC modules like phyCORE-i.MX 95 FPSC are populated with an e.MMC memory chip as the main storage. e.MMC devices contain raw Multi-Level Cells (MLC) or Triple-Level Cells (TLC) combined with a memory controller that handles ECC and wear leveling. They are connected via an SD/MMC interface to the i.MX 95 and are represented as block devices in the Linux kernel like SD cards, flash drives, or hard disks.

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

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

7.4.1 扩展 CSD 寄存器

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.

在 Linux 用户空间中，您可以查询寄存器：

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

你将会看到：

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

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

7.4.2 使能后台操作 (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)**.

默认情况下（取决于芯片），后台操作可能会定期执行，也可能不会，他影响读写的最大延迟时间。

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.

寄存器 BKOPS_EN（寄存器：163）的位 MANUAL_EN（位 0）的含义：

- 值 0：主机不支持手动触发 BKOPS。设备写入性能会受到影响。
- 值 1：主机支持手动触发 BKOPS。当主机不进行设备读写时，它会不时触发 BKOPS。

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).

JEDEC 标准 v5.1 引入了一种新的自动 BKOPS 功能。它使主机能够定期触发后台操作，因为设备在空闲时会自动启动 BKOPS（请参见寄存器 BKOPS_EN（寄存器：163）中位 AUTO_EN 的描述）。

- 要检查 *BKOPS_EN* 是否已设置，请执行：

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

  输出将会是，例如：

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

  值 0x00 表示 BKOPS_EN 被禁用，设备的写入性能受到影响。值 0x01 表示 BKOPS_EN 被启用，主机将不时发起后台操作。

- 通过以下命令使能 BKOPS_EN：

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

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

- 要设置 BKOPS_EN 位，请执行：

```
target:~$ mmc bkops_en manual /dev/mmcblk0
```

- 为了确保新设置生效并且内核能够自动触发 BKOPS，请先关闭系统：

```
target:~$ poweroff
```

小技巧

BKOPS_EN 位是一次性可编程的，无法恢复。

7.4.3 可靠写入

有两种不同的可靠写入选项：

1. Reliable Write option for a whole e.MMC device/partition.
2. 单次写的可靠写入方式。

小技巧

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 95 FPSC SoMs. To check this on the running target:

```
target:~$ mmc extcsd read /dev/mmcblk0 | grep -A 5 WR_REL_SET
Write reliability setting register [WR_REL_SET]: 0x1f
 user area: the device protects existing data if a power failure occurs during a write o
peration
 partition 1: the device protects existing data if a power failure occurs during a write
 operation
 partition 2: the device protects existing data if a power failure occurs during a write
 operation
 partition 3: the device protects existing data if a power failure occurs during a write
 operation
 partition 4: the device protects existing data if a power failure occurs during a write
 operation
--
 Device supports writing EXT_CSD_WR_REL_SET
 Device supports the enhanced def. of reliable write
```

如果默认没有启用，可以使用 mmc 工具启用它：

```
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.
```

第二个可靠写入方式是命令 CMD23 中的配置位 Reliable Write Request parameter（可靠写入请求参数）（位 31）。自内核版本 v3.0 起，文件系统（例如 ext4 的日志）和用户空间应用程序（如 fdisk 的分区表）会通过内核使用该可靠写功能。在 Linux 内核源代码中，它通过标志 REQ_META 进行处理。

结论：使用挂载选项 data=journal 的 ext4 文件系统在断电情况下是安全的。文件系统检查可以在断电后恢复文件系统，但在断电前刚写入的数据可能会丢失。在各种情况下，都可以恢复文件系统的正常状态。为了确保应用程序文件的正常保存，应用程序中应使用系统函数 fdatasync 或 fsync。

7.4.4 调整 ext4 根文件系统的大小

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.

- 获取当前设备大小：

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

- 输出如下：

```
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
```

- 使用 parted 将文件系统分区调整为设备的最大大小：

```
target:~$ parted /dev/mmcblk0 resizepart 2 100%
Information: You may need to update /etc/fstab.

target:~$ parted /dev/mmcblk0 print
Model: MMC Q2J55L (sd/mmc)
Disk /dev/mmcblk0: 7617MB
Sector size (logical/physical): 512[ 1974.191657]  mmcblk0: p1 p2
B/512B
Partition Table: msdos
Disk Flags:

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

- 将文件系统调整为新的分区大小：

```
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.

7.4.5 启用伪 SLC 模式

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

如果您更喜欢可靠性而不是存储容量，可以启用伪 SLC 模式或 SLC 模式。这个方法采用了增强属性，该属性在 JEDEC 标准中有所描述，可以对设备的一个连续区域设置。JEDEC 标准并未规定增强属性的实现细节和保证，这由芯片制造商自行决定。对于美光（Micron）芯片，增强属性提高了可靠性，但也将容量减半。

警告

在设备上启用增强属性时，所有数据将被丢失。

以下步骤展示了如何启用增强属性。

- First obtain the current size of the e.MMC device with:

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

  你将收到：

  ```
  BYT;
  /dev/mmcblk0:63652757504B:sd/mmc:512:512:unknown:MMC S0J58X:;
  ```

  如您所见，该设备的容量为 63652757504 字节 = 60704 MiB。

- 要获取启用伪 SLC 模式后的设备的大小，请使用：

  ```
  target:~$ mmc extcsd read /dev/mmcblk0 | grep ENH_SIZE_MULT -A 1
  ```

  例如：

  ```
  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
  ```

  这里的最大大小是 3719168 KiB = 3632 MiB。

- 现在，您可以通过输入以下命令为整个设备设置增强属性，例如 3719168 KiB：

  ```
  target:~$ mmc enh_area set -y 0 3719168 /dev/mmcblk0
  ```

  你将获得：

  ```
  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
  ```

- 为了确保新设置已生效，请关闭系统：

```
target:~$ poweroff
```

并进行上下电。建议您现在确认设置是否正确。

- 首先，检查 ENH_SIZE_MULT 的值，它必须是 3719168 KiB：

```
target:~$ mmc extcsd read /dev/mmcblk0 | grep ENH_SIZE_MULT  -A 1
```

您应该看到：

```
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
```

- 最后，检查设备的大小：

```
target:~$ parted -m /dev/mmcblk0 unit B print
BYT;
/dev/mmcblk0:31742492672B:sd/mmc:512:512:unknown:MMC S0J58X:;
```

7.4.6 擦除设备

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
```

选项 --secure 确保命令在 eMMC 设备擦除所有块之前会等待。-f (强制) 选项强制擦写，当 eMMC 设备包含有效数据分区时需要使用-f 选项。

小技巧

```
target:~$ dd if=/dev/zero of=/dev/mmcblk0 conv=fsync
```

该命令也会擦除设备上的所有信息，但这个命令不利于设备的磨损均衡，并且需要花费更长的时间！

7.4.7 e.MMC Boot Partitions

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

The user partition is called the User Data Area in the JEDEC standard and is the main storage partition. The partitions boot1 and boot2 can be used to host the bootloader and are more reliable. Which partition the i.MX 95 uses to load the bootloader is controlled by the boot configuration of the e.MMC device. The partition rpmb is a small partition and can only be accessed via a trusted mechanism.

此外，User 分区可以分为四个自定义的一般用途分区。对此功能的解释不在本文件涵盖的范围。有关更多信息，请参阅 JEDEC 标准第 7.2 章分区管理。

小技巧

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.

通过用户空间命令

在主机上运行：

```
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
```

下表是 i.MX 95 SoC 的偏移量：

SoC	User 分区偏移量	Boot 分区偏移量	e.MMC Device
i.MX 95	32 kiB	0 kiB	/dev/mmcblk0

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

(对于'boot0')：

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

(对于'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
```

7.5 GPIOs

Libra FPSC 具有一组专门用于 GPIO 的引脚。这些引脚直接连接到 i.MX 95 引脚，并被复用为 GPIO。它们可以直接在 Linux 用户空间中使用。处理器将其 GPIO 组织为 5 个 GPIO 组（GPIO1 –GPIO5），每个组包含 32 个 GPIO。gpiochip0、gpiochip32、gpiochip64、gpiochip96 和 gpiochip128 是这些内部 i.MX 95 GPIO 组 GPIO1 –GPIO5 的 sysfs 表示。

GPIO 被标识为 GPIO<X>_<Y>（例如：GPIO5_07）。<X> 表示 GPIO Bank，从 1 计数到 5，而 <Y> 表示该 Bank 内的 GPIO。<Y> 从 0 计数到 31（每个 bank 有 32 个 GPIO）。

相比之下，Linux 内核使用一个单一的整数来枚举系统中所有可用的 GPIO。计算正确数字的公式是：

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

从用户空间访问 GPIO 将使用 libgpiod。它提供了一个库和工具，用于与 Linux GPIO 字符设备进行交互。以下是一些工具的用法示例：

- 检测芯片上的 gpiochips：

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

- 显示关于 gpiochips 的详细信息，包括它们的名称、consumer、方向、活动状态和附加 flag：

```
target:~$ gpioinfo -c gpiochip0
```

- 读取 GPIO 的值（例如从 gpiochip0 的 GPIO 20）：

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

- 将 gpiochip0 上的 GPIO 20 的值设置为 0 并退出工具：

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

- gpioset 的帮助文本显示了可能的选项：

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

Set values of GPIO lines.

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

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

Options:
      --banner            display a banner on successful startup
```

(续下页)

(接上页)

```
  -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.
```

警告

某些 GPIO 用于特殊功能。在使用某个 GPIO 之前，请参考您板子的原理图或硬件手册，以确保该 IO 未被其他功能占用。

7.5.1 通过 sysfs 访问 GPIO

警告

通过 sysfs 访问 GPIO 已经过时了，我们建议使用 libgpiod。

默认情况下不再支持通过 sysfs 访问 GPIO。只有手动在内核配置中启用 `CONFIG_GPIO_SYSFS` 后才能支持。要在 menuconfig 中使 `CONFIG_GPIO_SYSFS` 可见，必须首先启用选项 `CONFIG_EXPERT` 。

You can also add this option for example to the defconfig you use in `arch/arm64/configs/` in the linux kernel sources. For our NXP based releases, this could be for example imx9_phytec_defconfig:

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

您也可以创建一个新的 config 片段。有关详细信息，请参阅我们的 Yocto Reference Manual。

7.6 I²C 总线

该 i.MX 95 包含多个多主支持快速模式的 I²C 模块。PHYTEC 板提供了许多不同的 I²C 设备，这些设备连接到 i.MX 95 的 I²C 模块。本节描述了我们 Libra FPSC 中集成的一些 I²C 设备的基本设备使用及其设备树（DT）表示。

i2c 的设备树节点包含一些设置，例如时钟频率，用于设置总线频率，以及引脚控制设置，包括 scl-gpios 和 sda-gpios，这些是用于总线恢复的备用引脚配置。

General I²C bus configuration from SoM (e.g. imx95-phycore-fpsc.dtsi): https://github.com/phytec/linux-phytec-imx/tree/v6.6.52-2.2.0-phy13/arch/arm64/boot/dts/freescale/imx95-phycore-fpsc.dtsi#L117

General I²C bus configuration from carrierboard (e.g. imx95-libra-rdk-fpsc.dts) https://github.com/phytec/linux-phytec-imx/tree/v6.6.52-2.2.0-phy13/arch/arm64/boot/dts/freescale/imx95-libra-rdk-fpsc.dts#L166

7.7 EEPROM

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

On the phyCORE-i.MX 95 FPSC SoM:

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

Device Tree Reference for SoM EEPROMs: https://github.com/phytec/linux-phytec-imx/tree/v6.6.52-2.2.0-phy13/arch/arm64/boot/dts/freescale/imx95-phycore-fpsc.dtsi#L125

And on the Libra FPSC carrier board:

- Board Detection EEPROM * Bus: I2C-4 * Address: 0x51 * Purpose: Reserved for carrier board identification

Device Tree Reference for Carrier Board: https://github.com/phytec/linux-phytec-imx/tree/v6.6.52-2.2.0-phy13/arch/arm64/boot/dts/freescale/imx95-libra-rdk-fpsc.dts#L231

7.8 RTC

RTC 可以通过 `/dev/rtc*` 访问。由于 PHYTEC 板通常有多个 RTC，因此可能会有多个 RTC 设备文件。

- 要找到 RTC 设备的名称，可以通过以下方式读取其 sysfs 条目：

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

- 例如，你将得到：

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

小技巧

这将列出所有实时时钟（RTC），包括非 I²C 接口的 RTC。如果存在设备树/aliases 条目，Linux 会根据这些条目分配 RTC 设备 ID。

日期和时间可以通过 `hwclock` 工具和 date 命令进行操作。要显示目标上设置的当前日期和时间：

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

使用日期命令更改日期和时间。日期命令以以下语法设置时间："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
```

备注

您的时区（在此示例中为 +0100）可能会有所不同。

使用 date 命令并不会改变实时时钟（RTC）的时间和日期，因此如果我们重启开发板，这些更改将会被丢弃。要写入 RTC，我们需要使用 `hwclock` 命令。使用 `hwclock` 工具将当前的日期和时间（通过 date 命令设置）写入 RTC，然后重启开发板以检查更改是否已应用到 RTC 上：

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

要从实时时钟（RTC）设置系统时间和日期，请使用：

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

7.8.1 RTC 唤醒 alarm

可以从实时时钟（RTC）发出中断以唤醒系统。该格式使用 Unix 纪元时间，即自 1970 年 1 月 1 日 UTC 午夜以来的秒数。要在从挂起到 RAM 状态后的 4 分钟唤醒系统，请输入：

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

备注

内部唤醒 alarm 时间将被向上舍入到下一个分钟，因为 alarm 功能不支持秒。

7.8.2 RTC 参数

实时时钟（RTC）具有一些功能，可以通过 `hwclock` 工具进行读取和设置。

- 我们可以通过以下方式检查 RTC 支持的功能：

```
target:~$ hwclock --param-get features
The RTC parameter 0x0 is set to 0x71.
```

这个值的含义在内核中进行了编码，每个位的定义为：

```
#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
```

- 我们可以通过以下方式检查 RTC BSM（Backup Switchover Mode 备份切换模式）：

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

- 我们可以通过以下方式设置 RTC BSM：

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

BSM 位的定义为：

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

小技巧

您应该将 BSM 模式设置为 DSM 或 LSM，以便在初始电源不可用时，RTC 可以切换到备用电源。请查看 **RV-3028** RTC 的 Datasheet，以了解 LSM（电平切换模式）和 DSM（直接切换模式）这两个定义的工作模式。

DT representation for I²C RTCs: https://github.com/phytec/linux-phytec-imx/tree/v6.6.52-2.2.0-phy13/arch/arm64/boot/dts/freescale/imx95-phycore-fpsc.dtsi#L139

And the addions on the carrierboard: https://github.com/phytec/linux-phytec-imx/tree/v6.6.52-2.2.0-phy13/arch/arm64/boot/dts/freescale/imx95-libra-rdk-fpsc.dts#L292

7.9 USB 主控制器

i.MX 95 SoC 的 USB 控制器为众多消费类便携设备提供了一种低成本的连接解决方案，实现 USB 设备之间的数据传输，传输速度可达 4 Gbit/s（超高速'SS'）。USB 子系统具有两个独立的 USB 控制器。这两个控制器都能够作为 USBDevice 或 USB Host 使用。每个核心都连接到一个 USB 3.0 物理层（PHY）。

BSP 支持大容量存储设备（优盘）和键盘。其他与 USB 相关的设备驱动程序必须根据需要在内核配置中启用。由于 udev，所有连接的存储设备都会获得唯一的 ID，并可以在 `/dev/disk/by-id` 中找到。这些 ID 可以在 `/etc/fstab` 中用于以不同的方式挂载不同的 USB 存储设备。

DT representation for USB Host: https://github.com/phytec/linux-phytec-imx/tree/v6.6.52-2.2.0-phy13/arch/arm64/boot/dts/freescale/imx95-libra-rdk-fpsc.dts#L383

7.10 视频

7.10.1 视频与 Gstreamer

默认情况下，BSP 安装了一个示例视频，路径为 */usr/share/qtphy/videos/* 。可以使用以下命令之一开始视频播放：

```
target:~$ gst-launch-1.0 playbin uri=file:///usr/share/qtphy/videos/caminandes_3_llamigos_720p_vp9.webm
```

- 或者：

```
target:~$ gst-launch-1.0 -v filesrc location=/usr/share/qtphy/videos/caminandes_3_llamigos_720p_vp9.webm !␣
↪decodebin name=decoder decoder. ! videoconvert ! waylandsink
```

- 或者：

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

7.11 显示

The Libra FPSC supports up to 3 different display outputs. The following table shows the required extensions and devicetree overlays for the different interfaces. For the alpha release, we have included overlays for two different LVDS displays. These displays are `edt,etml1010g3dra` or `powertip,ph128800t006-zhc01`. The name can be found on the back of the display.

备注

Currently only LVDS0 (onboard LVDS) is supported

接口	扩展板	设备树 overlay	
LVDS0	Libra FPSC	imx95-libra-rdk-fpsc-lvds-etml1010g3dra.dtbo	imx95-libra-rdk-fpsc-lvds-ph128800t006-zhc01.dtbo

备注

- 在更改 Weston 输出时，请确保音频输出也相匹配。
- LVDS0 (使用 PEB-AV-10 扩展) 和 LVDS1 (板载) 不能同时使用。

The default interface is LVDS0 (onboard LVDS).

备注

当前的显示驱动程序将连接到 LVDS 的 LCD 的像素时钟限制为 74.25MHz（或其分频）。如果这满足不了您的需求，请联系支持团队以获得进一步的帮助。

7.11.1 Weston 配置

为了让 Weston 正确的显示，需要进行正确的配置。这将在/etc/xdg/weston/weston.ini 中完成。

单一显示器

In our BSP, the default Weston output is set to LVDS-1 (onboard LVDS).

```
[output]
name=LVDS-1
mode=preferred
```

7.11.2 Qt Demo

使用 `phytec-qt6demo-image` 时，Weston 会在启动时启动。我们的 Qt6 DEMO 应用程序名为 “qtphy”，可以通过以下方式停止：

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

- 要重新开始 Demo，请运行：

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

- 要禁用 Demo 的自动启动，请运行：

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

- 要启用 Demo 的自动启动，请运行：

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

- Weston 可以通过以下方式停止：

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

备注

在关闭 Weston 之前，必须先关闭 Qt Demo。

7.11.3 背光控制

如果 LCD 连接到 PHYTEC 开发板，可以通过 Linux 内核的 sysfs 接口控制其背光。系统中所有可用的背光设备可以在文件夹/sys/class/backlight 中找到。读取相应的文件并向其写入数据可以控制背光。

备注

一些具有多显示的开发板在 /sys/class/backlight 有多个背光控制。比如：backlight0 和 backlight1

- 例如，要获取最大亮度级别（max_brightness），请执行：

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

  有效的亮度值范围是 0 到 <max_brightness>。

- 要获取当前亮度级别，请输入：

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

- 写入文件 brightness 以更改亮度：

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

  例如，关闭背光。

  有关所有文件的文档，请参见 https://www.kernel.org/doc/Documentation/ABI/stable/sysfs-class-backlight。

Device tree description of LVDS-0 can be found here: https://github.com/phytec/linux-phytec-imx/tree/v6.6.52-2.2.0-phy13/arch/arm64/boot/dts/freescale/imx95-libra-rdk-fpsc-lvds.dtsi#L35

7.12 电源管理

7.12.1 CPU 核心频率调节

i.MX 95 SoC 中的 CPU 能够调整时钟频率和电压。这用于在不需要 CPU 的全部性能时节省电力。调整频率和电压被称为“动态电压和频率调整”（DVFS）。i.MX 95 BSP 支持 DVFS 功能。Linux 内核提供了一个 DVFS 框架，允许每个 CPU 核心设置最小或最大频率和一个管理其运行的 governor。根据使用的 i.MX 9 型号，支持几种不同的频率。

小技巧

尽管 DVFS 框架为每个 CPU 核心提供了频率设置，但一个 CPU 核心的频率更改会影响其他 CPU 核心。因此，所有 CPU 核心始终共享相同的 DVFS 设置。每个核心的单独 DVFS 设置是不可能的。

- 要获取完整列表，请输入：

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

  例如 i.MX 8MPlus CPU，最高可达约 1.6 GHz，则结果将是：

```
1200000 1600000
```

- 要查询当前的频率输入：

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

governor 会根据它们的目标自动选择这些频率中的一个。

- 列出所有可用的 governor，使用以下命令：

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

  结果是：

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

- **conservative** governor 与 ondemand governor 非常相似。只是它的行为有所不同，它会更保守地增减 CPU 速度，而不是在 CPU 有任何负载的瞬间就跳到最大速度。
- **ondemand** （默认）根据当前系统负载在可能的 CPU 核心频率之间切换。当系统负载超过特定值时，它会立即提高 CPU 核心频率。
- **powersave** 始终选择最低的 CPU 核心频率。
- **performance** 始终选择最高的 CPU 核心频率。
- **userspace** 允许以 root 身份运行的用户或用户空间程序设置特定频率（例如，设置为 1600000）。输入：
- 要查询当前的 governor，请输入：

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

  您通常会得到：

```
ondemand
```

- 切换到另一个 governor（例如，userspace）可以通过以下方式完成：

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

- 现在你可以设置频率：

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

有关 governor 的更详细信息，请参阅 Linux 内核代码库中的 Linux 内核文档，路径为 `Documentation/admin-guide/pm/cpufreq.rst` 。

7.12.2 CPU 核心管理

该 i.MX 95 SoC 芯片上可以有多个处理器核心。例如，该 i.MX 95 具有 4 个 ARM 核心，可以在运行时单独开启和关闭。

- 要查看系统中所有可用的核心，请执行：

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

- 这将显示，例如：

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

  这里系统有四个处理器核心。默认情况下，系统中所有可用的核心都被启用，以获得最佳性能。

- 要关闭某个核，请执行：

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

  作为确认，您将看到：

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

  现在核心已关闭电源，并且该核心上不再安排任何进程。

- 您可以使用 top 命令查看核心和进程的图形概览：

```
target:~$ htop
```

- 要重新启用核心，请执行：

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

7.12.3 挂起到 RAM

phyCORE-i.MX 95 FPSC 支持基本的挂起和恢复。可以使用不同的唤醒源。挂起/恢复可以通过以下方式实现：

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

要通过串行控制台唤醒，请运行

```
target:~$ echo enabled > /sys/class/tty/ttyLP6/power/wakeup
target:~$ echo mem > /sys/power/state
```

7.13 热管理

7.13.1 U-Boot

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

7.13.2 内核

Linux 内核集成了热管理功能，能够监测芯片（SoC）温度，降低 CPU 频率，控制风扇，通知其他驱动程序减少功耗，并在最坏的情况下关闭系统（https://www.kernel.org/doc/Documentation/thermal/sysfs-api.txt）。

This section describes how the thermal management kernel API is used for the i.MX 95 SoC platform.

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

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

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

- 例如，你将得到：

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

备注

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

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

trip point	扩展工业级
被动（警告）	105°C
严重（关机）	125°C

（请查看内核 sysfs 文件夹 `/sys/class/thermal/thermal_zone0/`）

内核热管理使用这些触发点来触发事件并改变温控行为。内核中可用的热政策（也称为 thermal governor）包括：Step Wise 和 Power Allocator。BSP 中使用的默认政策是 step_wise。

小技巧

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

7.14 GPU

i.MX95 FPSC has MALI GPU G310. As recommended by NXP in the i.MX Graphics User's Guide, when running benchmarks, set the /etc/environment on the target with the variable "WESTON_FORCE_RENDERER=1" and restart Weston with "systemctl".

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

7.15 看门狗

PHYTEC i.MX 95 模块包含一个硬件看门狗，当系统挂起时能够重置开发板。看门狗在 U-Boot 中默认启动，超时时间为 60 秒。因此，即使在早期内核启动过程中，看门狗也已经开始运行。Linux 内核驱动程序控制看门狗，并确保它有被踢到。本节将解释如何使用 systemd 在 Linux 中配置看门狗，以避免系统挂起和重启期间的情况。

7.15.1 Systemd 中的看门狗支持

Systemd 从版本 183 开始支持硬件看门狗。

- 要启用看门狗支持，需要通过启用选项来配置 `/etc/systemd/` 中的文件 system.conf 文件：

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

RuntimeWatchdogSec 定义了看门狗的超时时间，而 *ShutdownWatchdogSec* 定义了系统重启时的超时时间。有关 systemd 下硬件看门狗的更多详细信息，请访问 http://0pointer.de/blog/projects/watchdog.html。更改将在重启后生效，或者运行：

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

CHAPTER 8

NXP GoPoint demo suite

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

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

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

8.1 ML Benchmark

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

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

8.1.1 Prerequisites

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

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

Yocto Project

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

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

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

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

and flash the Image to the board.

Ethernet

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

显示

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

备注

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

Console

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

8.1.2 Running the demo

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

```
gopoint tui
```

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