i.MX 8M Nano BSP Manual PD23.1.0

PHYTEC Messtechnik GmbH

2025 年 01 月 17 日

Contents

i.MX 8M Nano BSP 手册	
文档标题	i.MX 8M Nano BSP 手册
文档类型	BSP 手册
Yocto 手册	Kirkstone
发布日期	2024/01/10
母文档	i.MX 8M Nano BSP 手册

下表显示了与本手册兼容的 BSP：

Compatible BSPs	BSP **发布类型**	BSP **发布日期**	BSP **状态**
BSP-Yocto-NXP-i.MX8MM-PD23.1.0	大版本	2023/12/12	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.MX8M Nano 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 将为旗下所有产品提供各种硬件和软件文档。包括以下任一以及全部内容：

- **快速上手指南**：简单指导我们如何配置和启动 phyCORE 核心板，以及对构建 BSP、设备树和外设访问进行简要说明。
- **硬件手册**：核心板和配套底板的详细硬件描述。
- **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-8m-mini/nano/#downloads 中找到。

CHAPTER 1

支持的硬件

支持的 i.MX 8M Nano 系统芯片 (SoC) 的 phyBOARD-Polis。

在我们的网页上，您可以查看适用于 BSP 版本 BSP-Yocto-NXP-i.MX8MM-PD23.1.0 的所有 Machine 及其对应的 Article Numbers(产品型号)：网址. 如果您在该网页 **Supported Machines** 一节选择了特定的 **Machine Name** ，您可以查看被选中 machine 下的包含的 **Article Number(产品型号)** 以及简要的硬件描述。如果您只有 **Article Number** ，可以将 **Machine Name** 下拉菜单留空，仅选择您的 **Article Number** 。那么，它会显示您特定硬件所对应的 **Machine Name**

1.1 phyBOARD-Polis 器件

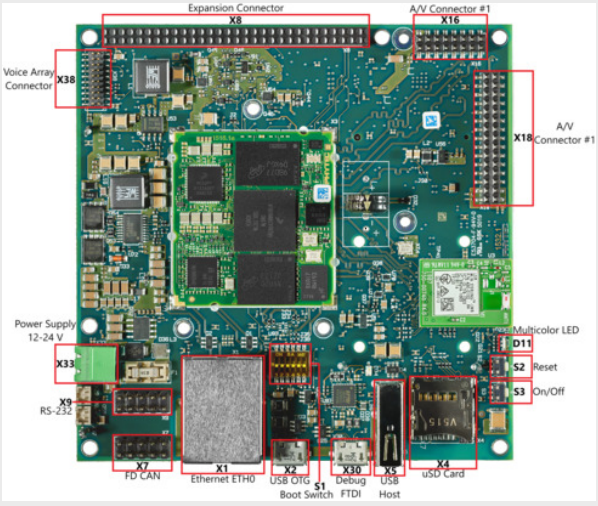


图 1: phyBOARD-Polis 器件（顶部）

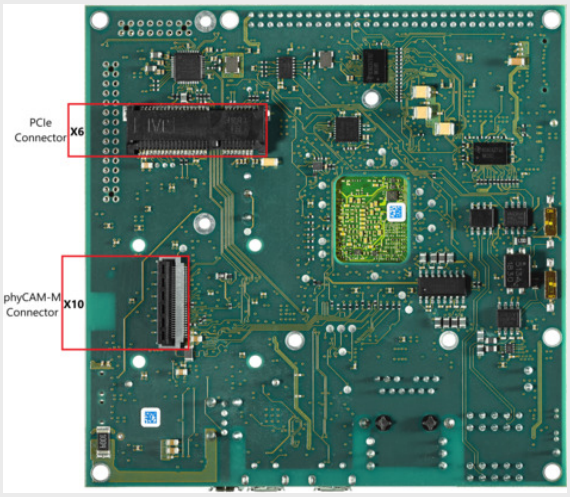


图 2: phyBOARD-Polis 器件（底部）

CHAPTER 2

开始使用

该 **phyCORE-i.MX8M Nano Kit** 包含预先烧写好的 SD 卡。它包含 phytec-headless-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。

2.1 下载镜像

phytec-headless-image 镜像包含完整系统所需的所有必要文件，您需确保镜像中各个分区以及裸数据都会被正确写入启动盘。可以从 PHYTEC 下载服务器 下载 partup 镜像文件或者是可以使用 `dd` 进行烧写的 WIC 镜像。

从下载服务器获取 partup 镜像文件或 WIC 镜像：

```
host:~$ wget https://download.phytec.de/Software/Linux/BSP-Yocto-i.MX8MM/BSP-Yocto-NXP-i.MX8MM-PD23.1.0/
↪images/ampliphy-vendor/phyboard-polis-imx8mn-2/phytec-headless-image-phyboard-polis-imx8mn-2.partup
host:~$ wget https://download.phytec.de/Software/Linux/BSP-Yocto-i.MX8MM/BSP-Yocto-NXP-i.MX8MM-PD23.1.0/
↪images/ampliphy-vendor/phyboard-polis-imx8mn-2/phytec-headless-image-phyboard-polis-imx8mn-2.wic
```

备注

针对 eMMC，我们建议使用 partup 去烧写比较大的或者是有复杂分区配置的镜像，因为它在写入速度上比 `dd` 更快，并且可以对闪存设备进行更灵活的配置。

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
```

现在，SD 卡已经准备好可以使用 `partup` 、`dd` 或 `bmap-tools` 来写入镜像。

2.2.2 使用 partup

使用 partup 烧写 SD 卡只需一个命令：

```
host:~$ sudo partup install phytec-headless-image-phyboard-polis-imx8mn-2.partup /dev/<your_device>
```

确保将 <your_device> 替换为您之前找到的设备名称。

关于 partup 的进一步使用说明，请参阅其 官方文档 。

备注

partup 的优点在于能够清除 MMC 用户区中的特定区域，这在我们的应用场景中用于擦除 U-Boot 环境。这是一个 *bmaptool* 无法解决的已知问题，如下所述。

partup 相较于其他烧写工具的一个主要优势是，它可以配置 MMC 的特定部分，比如他可以直接写入 eMMCboot 分区，无需调用其他命令。

2.2.3 使用 bmap-tools

准备 SD 卡的另一种方法是使用 bmap-tools。Yocto 会自动为 WIC 镜像创建一个映射文件（`<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-headless-image-phyboard-polis-imx8mn-2.wic /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。

2.2.4 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-headless-image-phyboard-polis-imx8mn-2.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-headless-image-phyboard-polis-imx8mn-2.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：

表 1: 启动模式选择

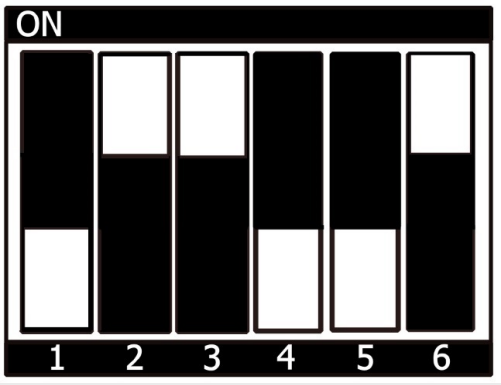


图 1: SD 卡启动

- 插入 SD 卡
- 使用 **micro USB** 线将开发板的*(X30)* 调试 USB 口和主机连接起来
- 给开发板通电

CHAPTER 3

编译 BSP

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

3.1 基本设置

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

3.2 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-IMX8MM ；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。

phyLinux 脚本使用 python 语言编写，是一个用于管理 PHYTEC Yocto BSP 工程的基础工具，帮助用户更快上手 BSP。

- 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=phyboard-polis-imx8mn-2 ./phyLinux init -p imx8mn -r BSP-
↪Yocto-NXP-i.MX8MM-PD23.1.0
```

在执行 init 命令后，phyLinux 将打印一些重要的说明。例如，它将打印您的 git 用户信息、选择的 SOC 和 BSP 版本，以及引导构建过程进行下一步处理的信息。

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-headless-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 phyboard-polis-imx8mn-2 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-imx8mn.bin**: ARM 可信固件二进制文件
- **lpddr4_pmu_train_2d_dmem_202006.bin, lpddr4_pmu_train_2d_imem_202006.bin**: DDR PHY 固件镜像
- **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。
- **Image**: Linux 内核镜像
- **Image.config**: 内核 config 文件
- **imx8mn-phyboard-polis*.dtb**: 内核设备树文件
- **imx8mn-phy*.dtbo**：内核设备树 overlay 文件
- **phytec-headless-image*.tar.gz**: 根文件系统
- **phytec-headless-image*.wic**: SD 卡镜像

CHAPTER 4

安装操作系统

4.1 启动模式开关 (S1)

该 phyBOARD-Polis 具有一个启动配置开关，带有六个可单独切换的开关，用于选择 phyCORE-i.MX 8M Nano 的默认启动源。

硬件修订主板：1532.2 及更新版本

表 1: 启动模式选择

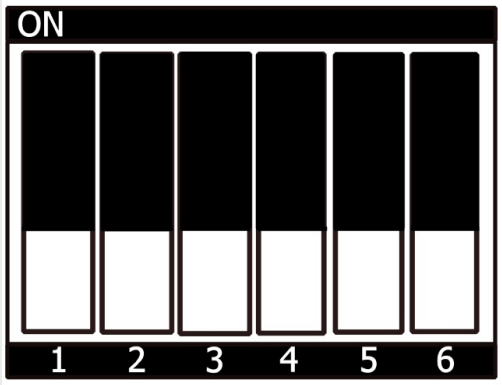


图 1: eMMC（核心板的默认启动方式）

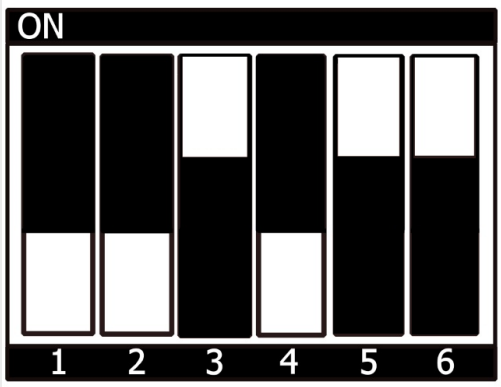


图 2: USB 串行下载器

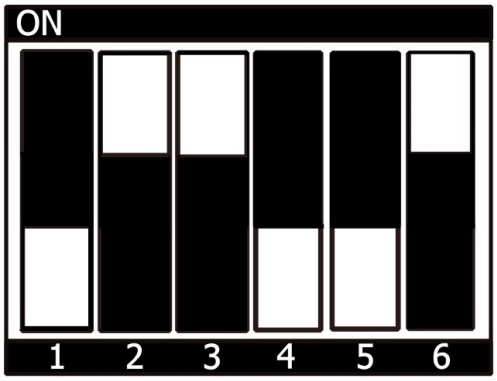


图 3: SD 卡

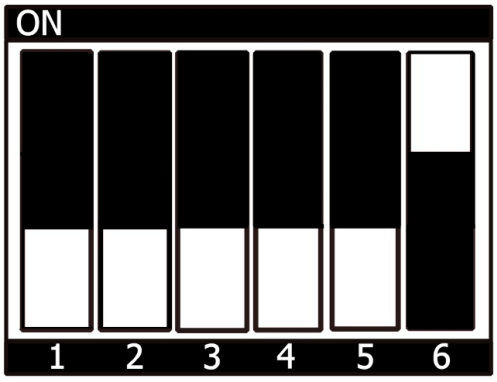


图 4: 内部 fuse

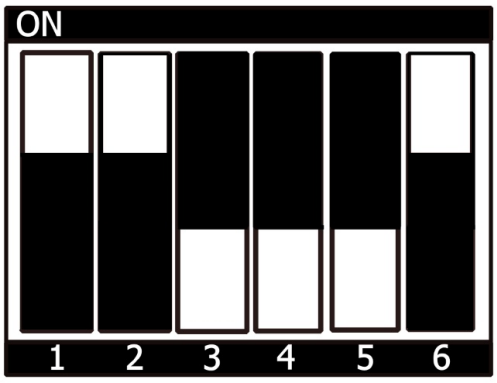


图 5: SPI NOR

表 2: 通过开关 (S1) 的第 5 位在 USB HOST/OTG 之间切换。

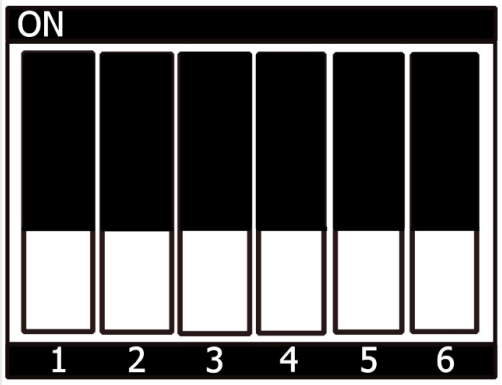


图 6: USB 主机

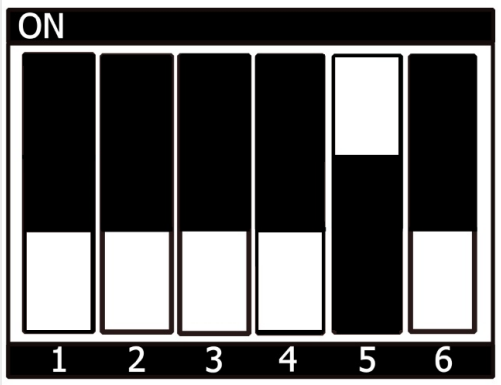


图 7: USB OTG

表 3: 通过开关 (S1) 的第 4 位在 UART1 的 RS485/RS232 之间切换。

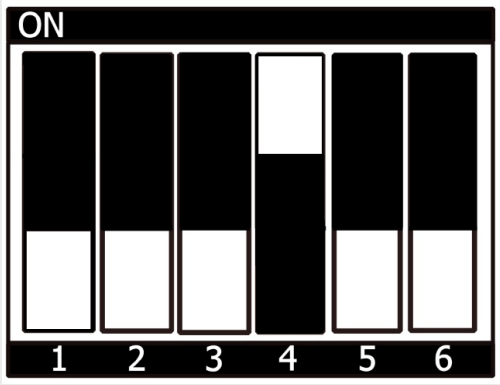


图 8: UART1 RS485

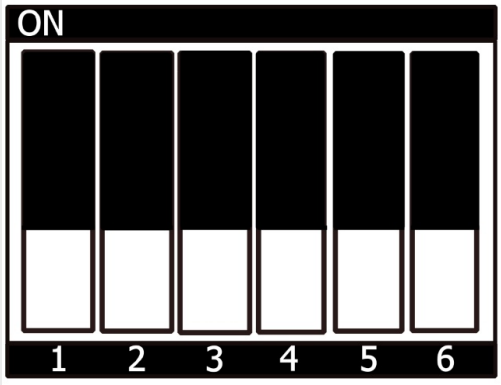


图 9: UART1 RS232

4.2 烧写 eMMC

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* 。

要从 eMMC 启动，请确保 BSP 镜像已正确烧写到 eMMC，并且*bootmode switch (S1)* 设置为 **eMMC**。

警告

当 eMMC 和 SD 卡上烧录了相同（完全一致）的镜像时，他们 boot 分区的 UUID 也是相同的。所以如果从 emmc 启动时，烧录一致镜像的 SD 卡也同时存在，这会导致不确定的后果，因为 Linux 会根据 UUID 来挂载启动分区。

```
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 从网络烧写 eMMC

i.MX 8M Nano 开发板具有以太网连接器，可以通过网络进行更新。确保正确设置主机，主机的 IP 需要设置为 192.168.3.10，子网掩码为 255.255.255.0，并且需要在主机开启 TFTP 服务。抽象来看，eMMC 设备和 SD 卡十分类似。因此，可以直接将 Yocto 生成的 **WIC 镜像**（`<name>.wic` ）直接烧写到 eMMC。该镜像包含 bootloader、内核、设备树、设备树 overlay 和根文件系统。

在开发板的 U-Boot 中通过网络烧写 eMMC

这些步骤将展示如何通过网络更新 eMMC。

小技巧

此步骤仅在镜像文件小于 1GB 的情况下会被执行成功，因为在启用 OPTEE 后，Bootloader 中可用的 RAM 大小有限，不足以加载超过 1GB 的镜像

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*

通过网络将您的镜像加载到内存中：

- 使用 DHCP

```
u-boot=> dhcp phytec-headless-image-phyboard-polis-imx8mn-2.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-phyboard-polis-imx8mn-2.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-phyboard-polis-imx8mn-2.wic
Using ethernet@30be0000 device
TFTP from server 192.168.3.10; our IP address is 192.168.3.11
Filename 'phytec-headless-image-phyboard-polis-imx8mn-2.wic'.
Load address: 0x40480000
```

(续下页)

(接上页)

```
Loading: ###################################
         ###################################
         ###################################
         ...
         ...
         ...
         ###################################
         #############
         11.2 MiB/s
done
Bytes transferred = 911842304 (36599c00 hex)
```

将镜像写入 eMMC：

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

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

在开发板的 Linux 系统中通过网络烧写 eMMC

您可以在开发板系统中更新 eMMC。

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*

使用一条命令，通过网络 ssh 协议将带有块映射的压缩或未压缩的镜像发送到开发板的 eMMC 上使用，执行：

```
target:~$ scp <USER>@192.168.3.10:/srv/tftp/phytec-headless-image-phyboard-polis-imx8mn-2.wic /tmp &&␣
↪bmaptool copy /tmp/phytec-headless-image-phyboard-polis-imx8mn-2.wic /dev/mmcblk2
```

在 Linux 主机上通过网络烧写 eMMC

可以在您的 Linux 主机上将镜像烧写到 eMMC。和之前一样，您需要在主机上准备一个完整的镜像。

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-headless-image-phyboard-polis-imx8mn-2.wic
phytec-headless-image-phyboard-polis-imx8mn-2.wic.bmap
```

通过网络 ssh 协议使用 `bmaptool` 命令将镜像发送到开发板的 eMMC：

```
host:~$ scp /srv/tftp/phytec-headless-image-phyboard-polis-imx8mn-2.wic root@192.168.3.11:/tmp && ssh␣
↪root@192.168.3.11 "bmaptool copy /tmp/phytec-headless-image-phyboard-polis-imx8mn-2.wic /dev/mmcblk2"
```

4.2.2 在运行的 U-Boot 中通过网络烧写 eMMC U-Boot 镜像

可以在 U-Boot 中更新 U-Boot 镜像 imx-boot，eMMC 上的 U-Boot 需要位于 eMMC 的 user 区域。

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*

通过 tftp 将镜像加载到 RAM 中，然后写入 eMMC：

```
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 从 USB 大容量存储设备烧写 eMMC

在开发板上通过 U-Boot 从 USB 烧写 eMMC

小技巧

此步骤仅在镜像文件小于 1GB 的情况下会被执行成功，因为在启用 OPTEE 后，Bootloader 中可用的 RAM 大小有限，不足以加载超过 1GB 的镜像

下面这些步骤展示如何通过 USB 设备更新 eMMC。将 |ref-bootswitch| 配置为 SD 卡启动，并插入 SD 卡。给开发板上电并进入 U-Boot 环境。将已存储了未压缩 WIC 镜像的优盘插入开发板 USB 接口。

将镜像从 USB 设备加载到 RAM 中：

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

将镜像写入 eMMC：

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

(续下页)

(接上页)

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

在运行的 Linux 系统中从 USB 烧写 eMMC

下面这些步骤展示如何在 Linux 系统上使用 USB 大容量存储设备烧写 eMMC。您需要一个保存了完整镜像的 U 盘和一个可从 SD 卡启动的核心板（例如 phytec-headless-image-phyboard-polis-imx8mn-2.wic）。将*bootmode switch (S1)* 设置为 SD 卡启动。

- 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-headless-image-phyboard-polis-imx8mn-2.wic
phytec-headless-image-phyboard-polis-imx8mn-2.wic.bmap
```

- 将镜像写入 phyCORE-i.MX 8M Nano eMMC（无分区的 MMC 设备 2）：

```
target:~$ bmaptool copy /mnt/phytec-headless-image-phyboard-polis-imx8mn-2.wic /dev/mmcblk2
```

- 在完成写入后，您的开发板可以从 eMMC 启动。

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

4.2.4 从 SD 卡烧写 eMMC

即使没有可用的网络，您也可以更新 eMMC。为此，您需要一个位于 SD 卡上的镜像文件（`*.wic` ）。由于镜像文件相当大，您需要在 SD 卡创建第三个分区。要创建新分区或扩展您的 SD 卡，请参见*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。

在开发板的 uboot 环境中通过 SD 卡烧写 eMMC

> **小技巧**
>
> 此步骤仅在镜像文件大小小于 1GB 的情况下有效，因为在启用 OPTEE 后，Bootloader 中可用的 RAM 大小有限。如果镜像文件过大，请阅读 *在开发板上通过 SD 卡更新 eMMC* 一节

- 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-headless-image.wic）复制到该分区。
- 将*bootmode switch (S1)* 配置为 SD 卡并插入 SD 卡。
- 打开电源并进入 U-Boot。
- 加载镜像：

```
u-boot=> ext4load mmc 1:3 ${loadaddr} phytec-headless-image-phyboard-polis-imx8mn-2.wic
reading
911842304 bytes read in 39253 ms (22.2 MiB/s)
```

- 将当前 mmc 设备切换到 eMMC：

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

- 将您的 WIC 镜像（例如 phytec-headless-image.wic）从 SD 卡烧写到 eMMC。这将对卡进行分区，并将 imx-boot、Image、dtb、dtbo 和根文件系统复制到 eMMC。

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

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

- 关闭电源并将*bootmode switch (S1)* 更改为 eMMC。

在开发板的 linux 环境中通过 SD 卡烧写 eMMC

您也可以在 Linux 系统中烧写 eMMC。您只需要一个 partup 包或保存在 SD 卡上的 WIC 镜像。

- 检查在 SD 卡上保存的 partup 包或 WIC 镜像文件：

```
target:~$ ls
phytec-headless-image-phyboard-polis-imx8mn-2.partup
phytec-headless-image-phyboard-polis-imx8mn-2.wic
phytec-headless-image-phyboard-polis-imx8mn-2.wic.bmap
```

- 使用 partup 将镜像写入 phyCORE-i.MX 8M Nano 的 eMMC（MMC 设备 2 **不带**分区字样）：

```
target:~$ partup install phytec-headless-image-phyboard-polis-imx8mn-2.partup /dev/mmcblk2
```

使用 partup 烧写的优点是可以充分利用 eMMC 设备的全部容量，会相应自动调整分区大小。

备注

另外，也可以使用 `bmaptool` 工具：

```
target:~$ bmaptool copy phytec-headless-image-phyboard-polis-imx8mn-2.wic /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。

- 在完成写入后，您的开发板可以从 eMMC 启动。

警告

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

4.3 烧写 SPI NOR Flash

phyCORE-i.MX8MN 模块可选配 SPI NOR Flash。要从 SPI Flash 启动，请将*bootmode switch (S1)* 设置为 **SPI NOR** 。SPI Flash 通常比较小。phyBOARD-Pollux-i.MX8MP 开发套件仅配备 32MB 的 SPI NOR Flash。只能存储 bootloader 及其环境变量。默认情况下，内核、设备树和文件系统会从 eMMC 加载。

SPI NOR Flash 分区表在 U-Boot 环境变量中定义。可以通过以下命令打印：

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

4.3.1 通过网络烧写 SPI NOR Flash

SPI NOR 可以包含 bootloader 及其环境变量。arm64 的 linux 内核无法自行解压缩，内核镜像大小超出了 phyCORE-i.MX 8M Nano 上的 SPI NOR Flash 的容量。

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*

在开发板的 U-Boot 环境中通过网络烧写 SPI NOR

类似于通过网络更新 eMMC，请确保正确设置主机 PC。IP 地址需要设置为 192.168.3.10，子网掩码设置为 255.255.255.0，并且需要有一个可用的 TFTP 服务。在进行读写之前，需要对 SPI NOR Flash 进行枚举：

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

- SPI NOR Flash 需要使用特殊格式的 U-Boot 镜像。确保您使用了正确的镜像文件。通过 tftp 加载镜像，然后将 bootloader 写入 Flash：

  ```
  u-boot=> tftp ${loadaddr} imx-boot-phyboard-polis-imx8mn-2-fspi.bin-flash_evk_flexspi
  u-boot=> sf update ${loadaddr} 0 ${filesize}
  device 0 offset 0x0, size 0x1c0b20
  1641248 bytes written, 196608 bytes skipped in 4.768s, speed 394459 B/s
  ```

- 同时需要擦除环境分区。这样，环境变量可以在从 SPI NOR Flash 启动后写入：

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

在开发板 linux 环境中通过网络烧写 SPI NOR Flash

- 将镜像从主机复制到开发板：

  ```
  host:~$ scp imx-boot-phyboard-polis-imx8mn-2-fspi.bin-flash_evk_flexspi root@192.168.3.11:/root
  ```

- 查找要擦除的 U-boot 分区的块数：

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

- 擦除 U-Boot 分区并烧写：

```
target:~$ flash_erase /dev/mtd0 0x0 60
target:~$ flashcp imx-boot-phyboard-polis-imx8mn-2-fspi.bin-flash_evk_flexspi /dev/mtd0
```

4.3.2 从 SD 卡烧写 SPI NOR Flash

SPI NOR Flash 上的 bootloader 也可以通过 SD 卡进行烧写。

在开发板的 U-Boot 环境中从 SD 卡烧写 SPI NOR

- 将 SPI NOR Flash 的 U-boot 镜像 imx-boot-phyboard-polis-imx8mn-2-fspi.bin-flash_evk_flexspi 复制到 SD 卡的第一个分区。
- 在进行读写操作之前，需要对 SPI-NOR Flash 进行枚举：

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

- SPI NOR Flash 需要使用特殊格式的 U-Boot 镜像，请确保使用正确的镜像文件。从 SD 卡加载镜像，擦除并将 bootloader 写入 flash：

```
u-boot=> mmc dev 1
u-boot=> fatload mmc 1:1 ${loadaddr} imx-boot-phyboard-polis-imx8mn-2-fspi.bin-flash_evk_flexspi
u-boot=> sf update ${loadaddr} 0 ${filesize}
```

- 同时需要擦除环境分区。这样，环境变量可以在从 SPI NOR Flash 启动后写入：

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

在开发板的 linux 环境中从 SD 卡烧写 SPI NOR

- 将 SPI NOR Flash 的 U-boot 镜像 imx-boot-phyboard-polis-imx8mn-2-fspi.bin-flash_evk_flexspi 复制到 SD 卡的第一个分区。
- 挂载 SD 卡：

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

- 查找要擦除的 U-Boot 分区的块数：

```
target:~$ mtdinfo /dev/mtd0
mtd0
Name:                           u-boot
```

(续下页)

(接上页)

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

- 擦除 u-boot 分区并烧写：

```
target:~$ flash_erase /dev/mtd0 0x0 60
target:~$ flashcp /mnt/imx-boot-phyboard-polis-imx8mn-2-fspi.bin-flash_evk_flexspi /dev/mtd0
```

4.4 RAUC

BSP 支持 RAUC（Robust Auto-Update Controller）。它管理设备固件更新的过程。这包括更新 Linux 内核、设备树和根文件系统。PHYTEC 已撰写了一份在线手册，介绍如何在我们的 BSP 中集成 RAUC: L-1006e.A5 RAUC Update & Device Management Manual 。

CHAPTER 5

开发

5.1 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
```

5.1.1 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"
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,
```

(续下页)

(接上页)

```
        "next-server": "192.168.3.10",
        "subnet": "192.168.3.0/24",
        "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。

5.2 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。

5.2.1 在主机上放置网络启动的镜像

- 将内核镜像复制到您的 tftp 目录中：

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

- 将设备树复制到您的 tftp 目录：

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

- 将您想要使用的所有 overlay 文件复制到您的 tftp 目录中：

```
host:~$ cp *.dtbo /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-headless-image-phyboard-polis-imx8mn-2.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。

5.2.2 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。

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

<overlayfilenames> 必须替换为您想要使用的 overlay 设备树文件名。用空格分隔文件名。例如：

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

小技巧

所有支持的设备树 overlay 在*device tree* 章节中。

5.2.3 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*

5.2.4 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=> run netboot
  ```

5.3 使用 UUU 工具

NXP 的镜像更新工具（UUU-Tool）是一款在主机上运行的软件，用于通过 SDP（串行下载协议）在开发板上下载并运行 bootloader。有关详细信息，请访问 https://github.com/nxp-imx/mfgtools 或下载 官方 UUU 工具文档。

5.3.1 使用 UUU 工具的准备

- 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：

  这个 BASH 命令只是暂时将 UUU 添加到 `PATH` 中。要永久添加，请将此行添加到 `~/.bashrc` 中。

  ```
  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
  ```

5.3.2 获取镜像

可以从我们的服务器下载 imx-boot，或者从 Yocto 编译目录中的 build/deploy/images/phyboard-polis-imx8mn-2/获取它。要将 wic 镜像烧写到 eMMC，你还需要 phytec-headless-image-phyboard-polis-imx8mn-2.wic。

5.3.3 开发板准备

将*bootmode switch (S1)* 设置为 **USB 串行下载**。同时，将 USB 端口*X2* 连接到主机。

5.3.4 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*(X30)* 在终端上查看启动日志。

备注

UUU 工具使用的默认启动命令为 fastboot。如果您想更改此设置，请在 U-Boot 提示符下使用 setenv bootcmd_mfg 修改环境变量 bootcmd_mfg。但是请注意，当开发板再次使用 UUU 工具启动时，默认环境变量会被加载，saveenv 重启后不生效。如果您想永久的更改 U-boot 的启动命令，则需要更改 U-Boot 代码。

5.3.5 通过 UUU 工具将 U-boot 镜像烧写到 eMMC

警告

UUU 将 U-boot 刷入 eMMC BOOT（硬件）启动分区后，会在 eMMC 中设置 BOOT_PARTITION_ENABLE。这带来一个问题，因为我们希望 bootloader 保存在 eMMC 的 USER 分区中。如果烧写入新的包含 U-boot 的.wic 镜像而不禁用 BOOT_PARTITION_ENABLE 位，将导致设备始终使用保存在 BOOT 分区中的 U-boot。为了在 U-Boot 中解决此问题，需要：

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

or check Disable booting from eMMC boot partitions from Linux.

这样 bootloader 虽然会被烧写到 eMMC 的 BOOT 分区，但在启动中不会被使用！

在使用 **partup** 工具和 `.partup` 包进行 eMMC 烧写时，上述过程是默认进行的，这是 partup 的优势，简化烧写过程。

执行并给开发板上电：

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

5.3.6 通过 UUU 工具将 wic 镜像烧写到 eMMC

执行并给开发板上电：

```
host:~$ sudo uuu -b emmc_all imx-boot phytec-headless-image-phyboard-polis-imx8mn-2.wic
```

5.4 独立编译准备

在本节中，我们将描述如何在不使用 Yocto Project 的情况下编译 U-Boot 和 Linux kernel。U-Boot、Linux kernel 以及其他源码的 git 仓库都可以在我们的 Git 服务器 上找到，地址为 git://git.phytec.de。

备注

如果您的公司防火墙/网关禁止 git 协议，您可以改用 HTTP 或 HTTPS（例如：git clone git://git.phytec.de/u-boot-imx）

5.4.1 Git 仓库

- 使用的 U-Boot 仓库：

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

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

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

- 我们的 i.MX 8M Nano 内核是基于 linux-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/dynamic-layers/freescale-layer/recipes-kernel/linux/linux-imx_*.bb
meta-phytec/recipes-bsp/u-boot/u-boot-imx_*.bb
```

5.4.2 获取 SDK

您可以在此处下载 SDK 这里，或者使用 Yocto 去编译生成 SDK：

- 移动到 Yocto 的 build 目录：

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

在成功编译后，SDK 安装包保存在 `build/deploy*/sdk`。

5.4.3 安装 SDK

- 设置正确的权限并安装 SDK：

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

（续下页）

(接上页)

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

5.4.4 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/4.0.13/environment-setup-cortexa53-crypto-phytec-linux
```

5.4.5 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
```

5.5 单独编译 U-Boot

5.5.1 获取源代码

- 获取 U-Boot 源代码：

  ```
  host:~$ git clone git://git.phytec.de/u-boot-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 v2022.04_2.2.2-phy5
- 查看所需的 *U-Boot* **tag**：

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

- 设置编译环境：

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

5.5.2 获取所需的二进制文件

要编译 bootloader，您需要将这些文件复制到您的 u-boot-imx 编译目录，并将其重命名以适应 *mkimage* 脚本：

- **ARM Trusted firmware 二进制文件**（*mkimage* 工具兼容格式 **bl31.bin** ）：bl31-imx8mn.bin
- **OPTEE 镜像** (可选的)：tee.bin
- **DDR firmware files** （ *mkimage* 工具兼容格式 **lpddr4_[i,d]mem_*d_*.bin** ）：lpddr4_dmem_1d_*.bin, lpddr4_dmem_2d_*.bin, lpddr4_imem_1d_*.bin, lpddr4_imem_2d_*.bin

如果您已经使用 Yocto 编译了我们的 BSP，您可以在 yocto 工程目录中获取 bl31-imx8mn.bin、tee.bin 和 lpddr4_*.bin：*BSP Images*

或者你可以在这里下载文件：https://download.phytec.de/Software/Linux/BSP-Yocto-i.MX8MM/BSP-Yocto-NXP-i.MX8MM-PD23.1.0/images/ampliphy-vendor/phyboard-polis-imx8mn-2/imx-boot-tools/

警告

确保您重命名所需的文件，以和 *mkimage tool* 兼容。

5.5.3 编译 bootloader

- 编译 flash.bin (imx-boot):

```
host:~/u-boot-imx$ make phycore-imx8mn_defconfig
host:~/u-boot-imx$ make flash.bin
```

5.5.4 将 bootloader 烧写到块设备上

flash.bin 文件可以在 u-boot-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 8M Nano	32 kiB	0 kiB	/dev/mmcblk2

例如，烧写 SD 卡：

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

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。

5.6 单独编译内核

5.6.1 配置源代码

- 使用的 linux-imx 分支可以在 release notes 中找到
- 此版本所需的标签称为 v5.15.71_2.2.2-phy3
- Check out 所需的 linux-imx 标签：

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

- 为了提交更改，强烈建议切换到一个新分支：

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

- 设置编译环境：

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

5.6.2 编译内核

- 编译 Linux 内核：

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

- 安装内核模块，比如安装到 NFS 目录：

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

- 镜像可以在 ~/linux-imx/arch/arm64/boot/Image 找到
- dtb 文件可以在 ~/linux-imx/arch/arm64/boot/dts/freescale/imx8mn-phyboard-polis.dtb 找到
- 要（重新）编译设备树和 -overlay 文件，只需运行

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

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
```

5.6.3 将内核复制到 SD 卡

内核及 module 和对应的设备树二进制文件可以用以下方式复制到已挂载的 SD 卡上。

```
host:~/linux-imx$ cp arch/arm64/boot/Image /path/to/sdcard/boot/
host:~/linux-imx$ cp arch/arm64/boot/dts/freescale/imx8mn-phyboard-polis.dtb /path/to/sdcard/boot/oftree
host:~/linux-imx$ make INSTALL_MOD_PATH=/path/to/sdcard/root/ modules_install
```

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 imx8mn -r BSP-Yocto-NXP-i.MX8MM-PD23.1.y
```

这将初始化一个 BSP，用于跟踪当前版本（BSP-Yocto-NXP-i.MX8MM-PD23.1.0 ）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.MX8MM-PD23.1.0 ）的 PDXX.Y 数字更高的版本，并且以 `.y` 结尾：

```
host:~$ ./phyLinux init -p imx8mn
```

5.8 获取最新的 Upstream 支持

我们有一个使用 Yocto 主分支（不是 NXP 发布的）的 manifest，他使用 upstream 的 Linux 和 U-Boot。这可以用来测试最新的 upstream kernel/U-Boot。

备注

master 分支的 manifest 反映了最新的开发状态。有时会出现一些 bug。我们会定期修复 master 分支。

```
host:~$ ./phyLinux init -p imx8mn -r BSP-Yocto-Ampliphy-i.MX8MM-master
```

5.9 格式化 SD 卡启动盘以允许通过 SD 卡进行烧录

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。

5.9.1 Gparted

- 获取 GParted：

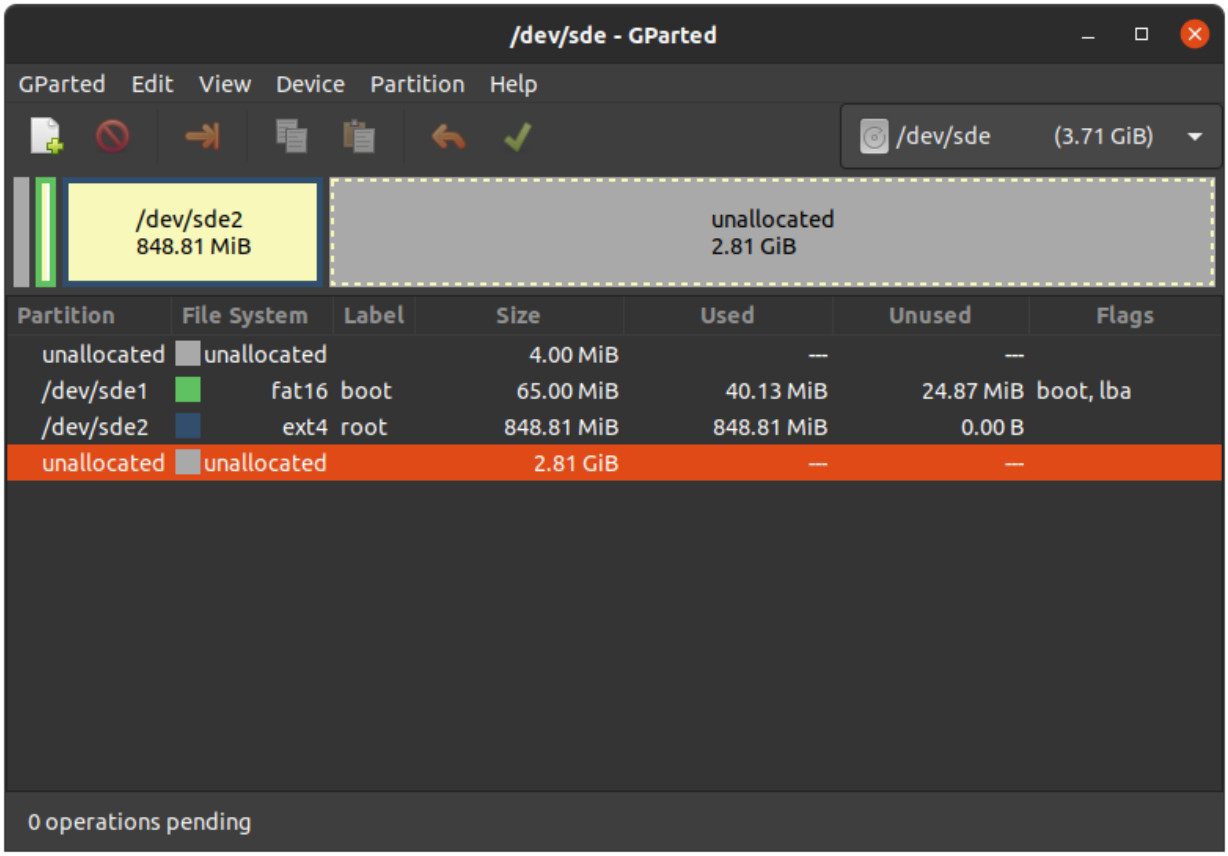
  ```
  host:~$ sudo apt install gparted
  ```

- 将 SD 卡插入主机并获取设备名称：

  ```
  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
  ...
  ```

- 卸载所有 SD 卡分区。
- 启动 GParted：

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

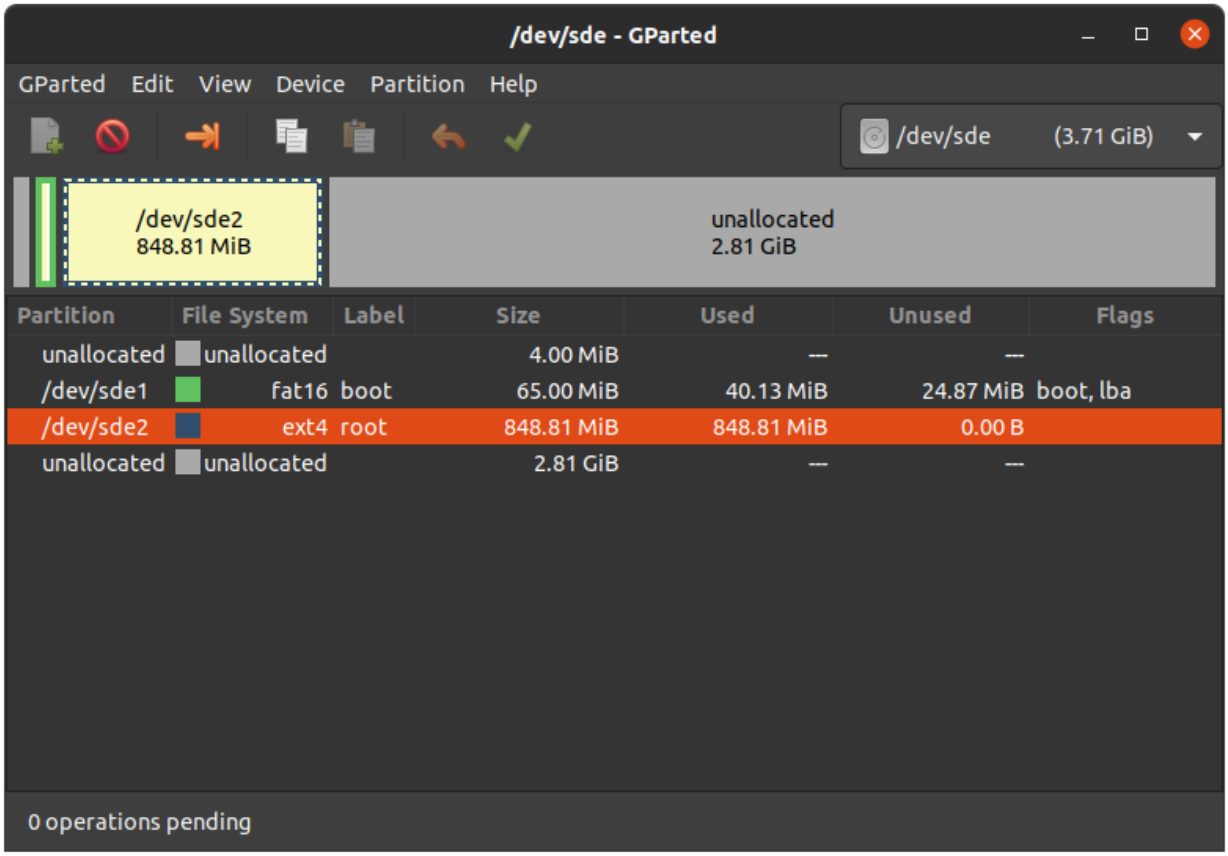


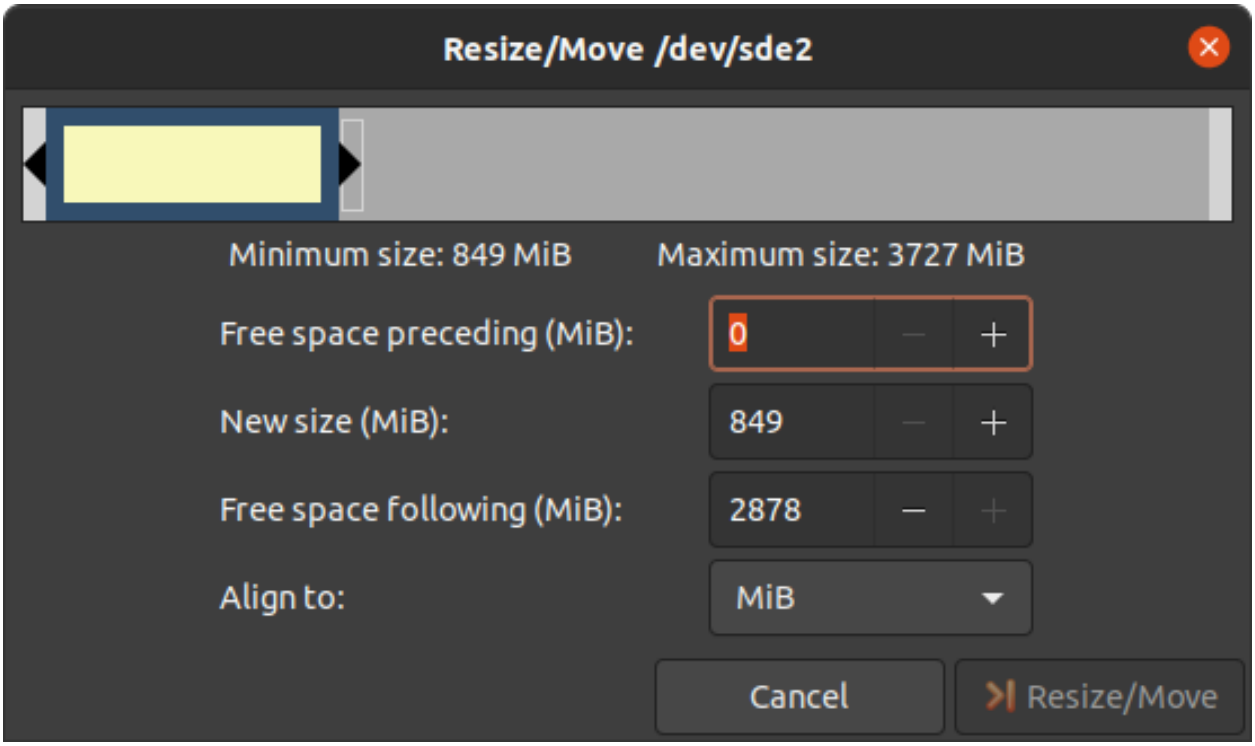
扩展根文件系统

警告

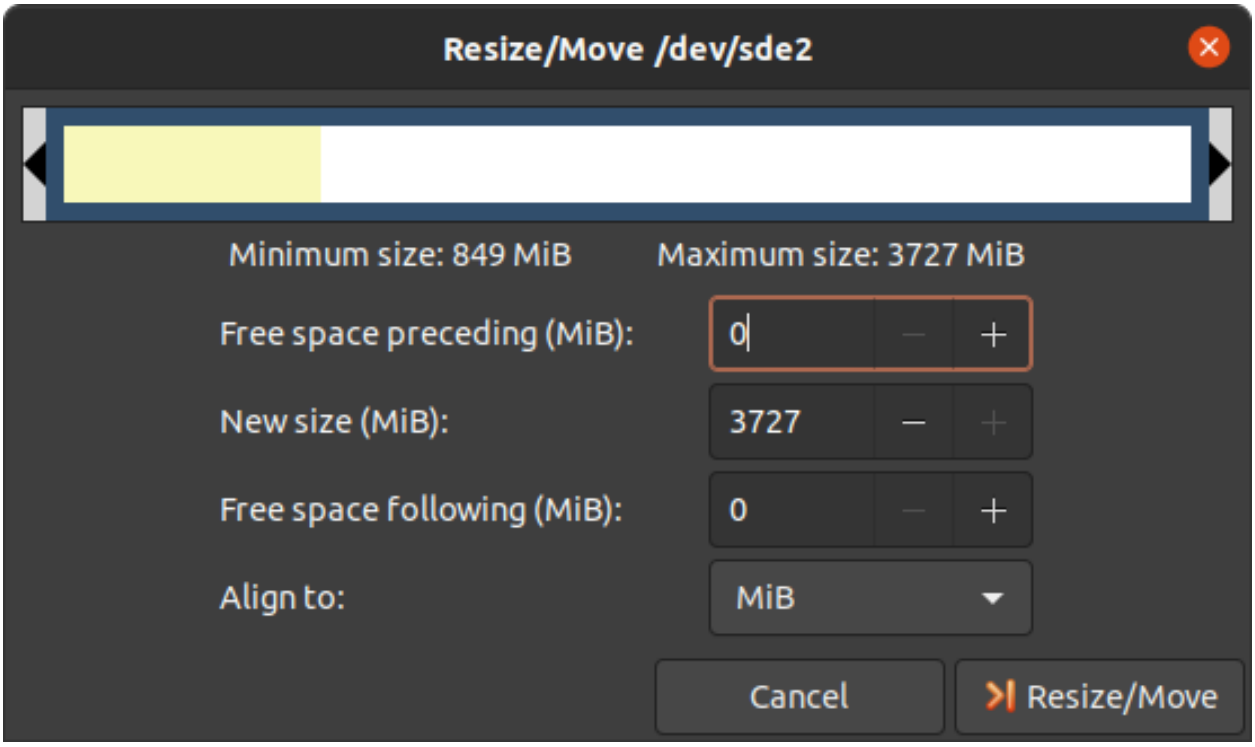
使用 resize2fs 版本 1.46.6 及更早版本的 PC 系统（例如 Ubuntu 22.04）无法烧写在 Mickledore 以及更新的 yocto 版本上创建的 partup 软件包。这个是因为 resize2fs 新增了默认选项而导致的兼容性问题。有关详细信息，请参阅 发布说明 。

- 在右上角的下拉菜单中选择您的 SD 卡设备
- 选择 ext4 根分区并点击调整大小:

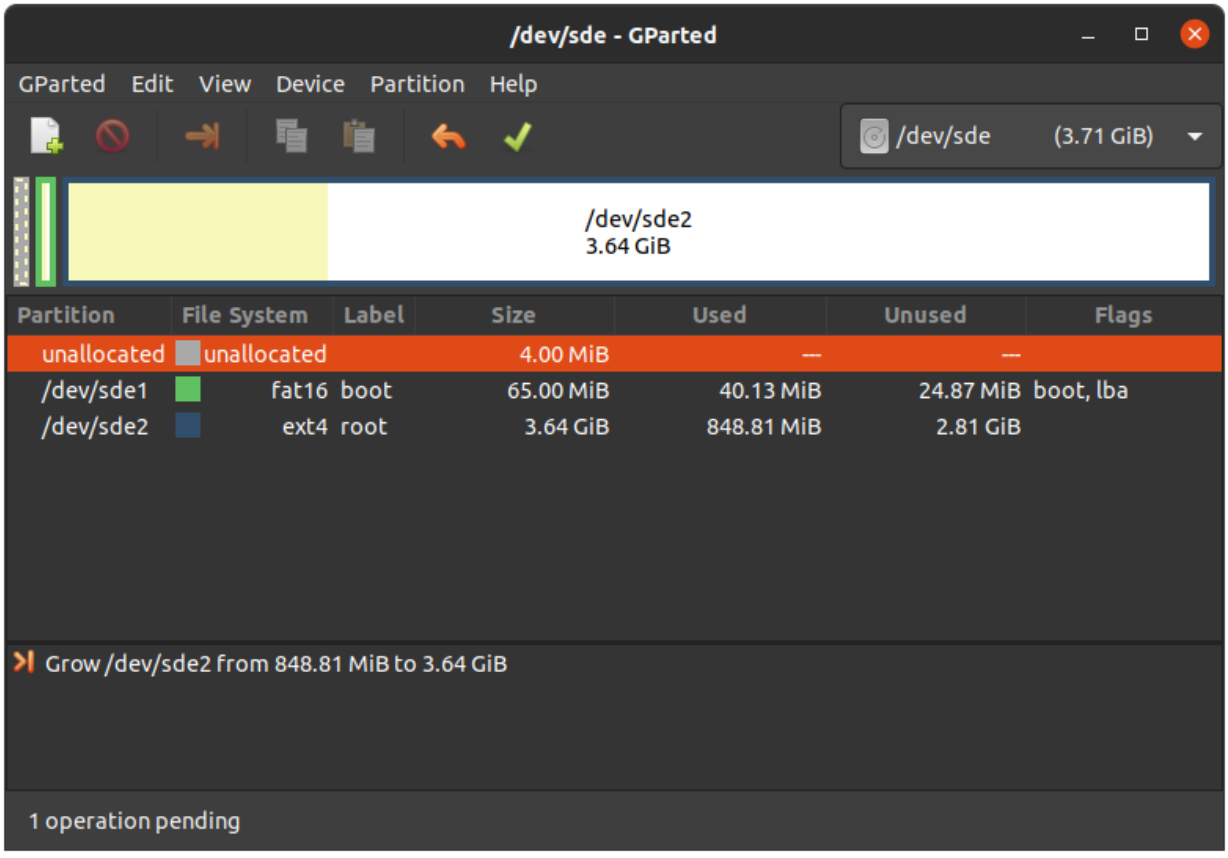




- 您可以根据需要拖动滑块或手动输入大小。



- 通过点击“Change Size”按钮确认您的输入。



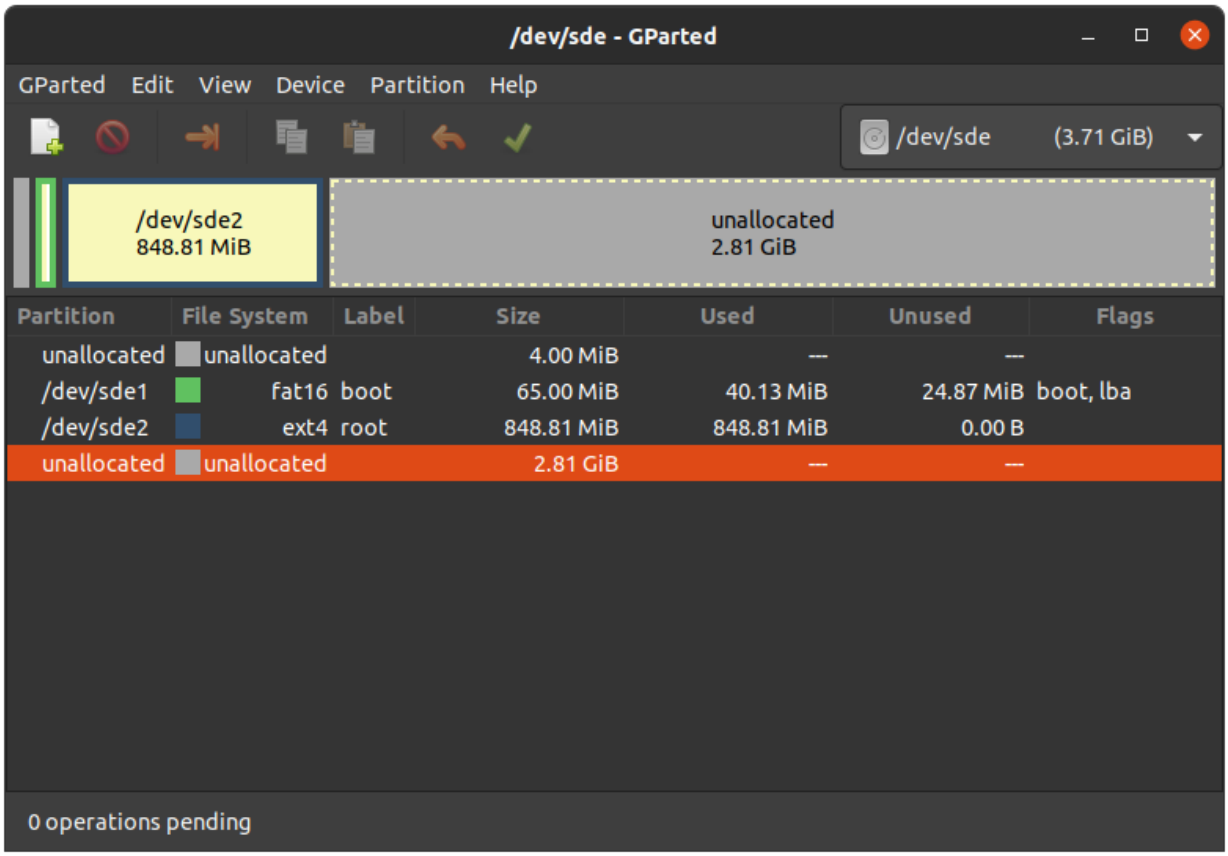
- 要应用您的更改，请按绿色勾号。

- 现在您可以挂载根分区并将 phytec-headless-image-phyboard-polis-imx8mn-2.wic 镜像复制到其中。然后再卸载它：

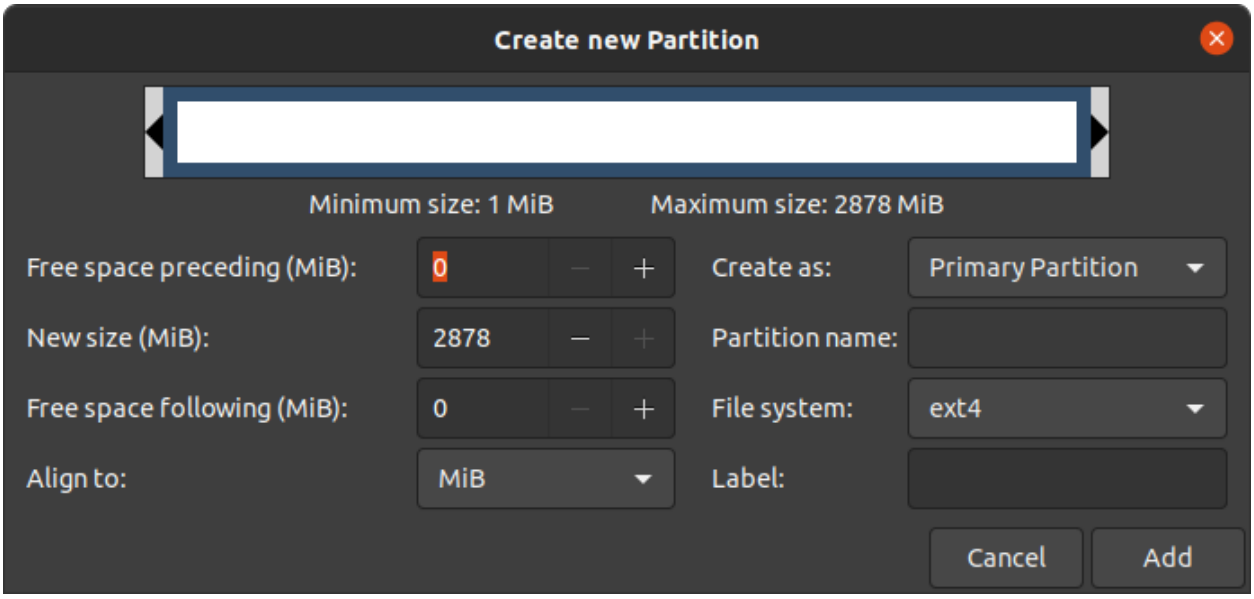
```
host:~$ sudo cp phytec-headless-image-phyboard-polis-imx8mn-2.wic /mnt/ ; sync
host:~$ umount /mnt
```

创建第三个分区

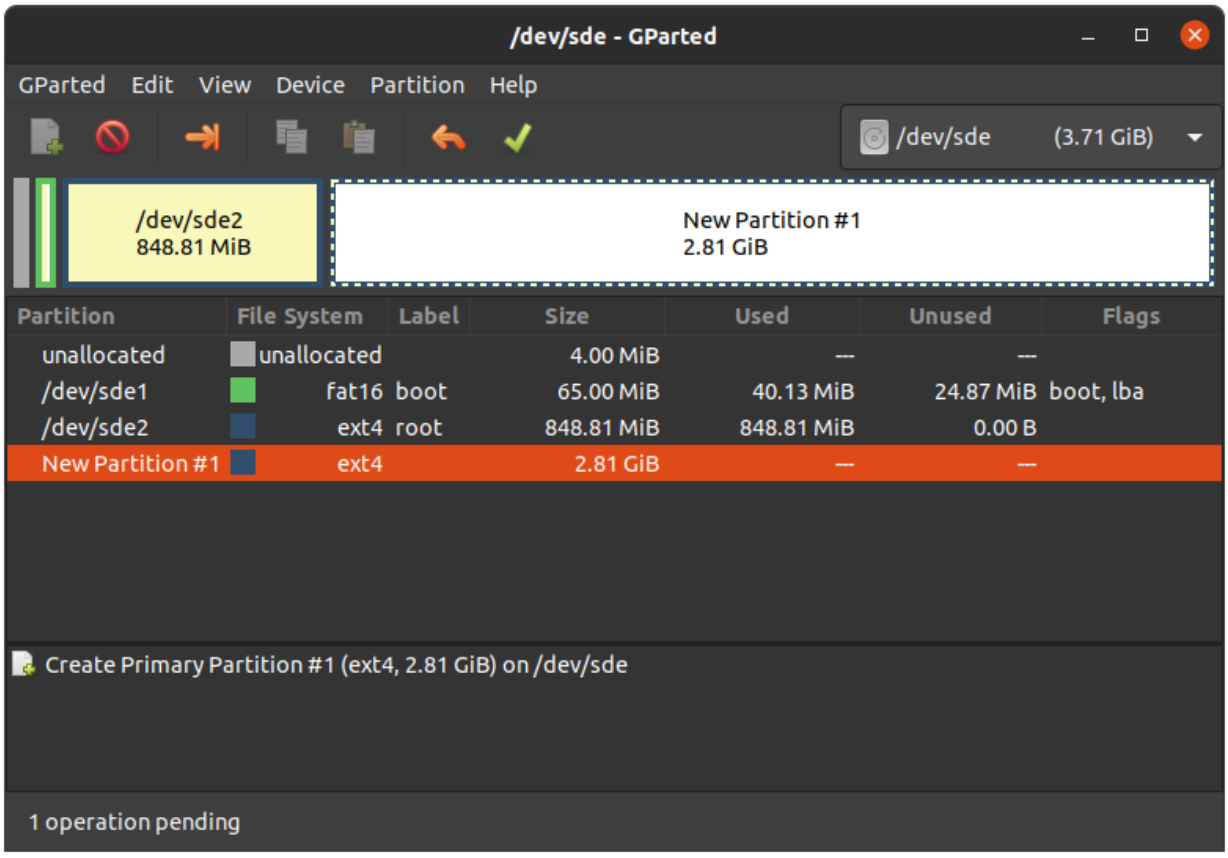
- 在右上角的下拉菜单中选择您的 SD 卡设备



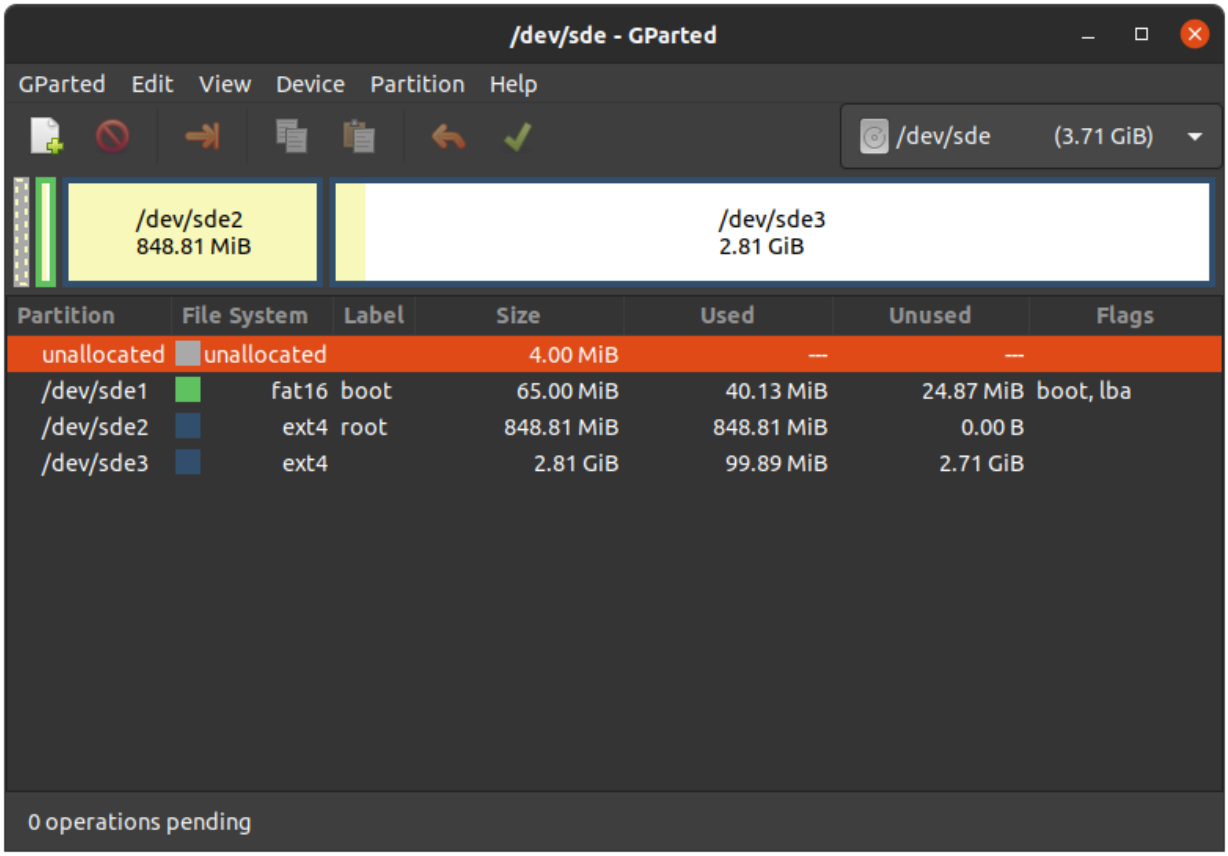
- 选择更大的未分配区域，然后点击”New”：



- 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-headless-image-phyboard-polis-imx8mn-2.wic 镜像复制到其中。然后卸载它：

```
host:~$ sudo mount /dev/sde3 /mnt
host:~$ sudo cp phytec-headless-image-phyboard-polis-imx8mn-2.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 8M Nano BSP 设备树概念

以下部分说明了 PHYTEC 配置基于 i.MX 8M Nano 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 module dtsi 文件。从 SoC i.MX 8M Nano 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 8 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。例如扩展板的硬件描述。对比源码中的 include，overlay 是用覆盖的方式来生效。overlay 也可以根据节点是否连接来设置节点状态。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/overlays` 中。

phyboard-polis-imx8mn-2.conf 可用的 overlay 文件有：

```
imx8mn-phyboard-polis-peb-eval-01.dtbo
imx8mn-phyboard-polis-peb-av-010.dtbo
imx8mn-phycore-rpmsg.dtbo
imx8mn-phycore-no-eth.dtbo
imx8mn-phycore-no-spiflash.dtbo
imx8mn-vm016.dtbo
imx8mn-vm016-fpdlink.dtbo
imx8mn-vm017.dtbo
imx8mn-vm017-fpdlink.dtbo
imx8mn-dual-vm017-fpdlink.dtbo
```

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空格分隔的 overlay 文件列表，这些 overlay 文件将在启动过程中应用。根据启动源，overlay 文件必须放置在 eMMC/SD 卡的启动分区中，或者通过 tftp 加载。在 $KERNEL_DEVICETREE 这个 Yocto machine 变量中设置的 overlay 文件将自动添加到最终 WIC 镜像的启动分区中。

${overlays} 变量可以直接在 U-Boot 环境中设置，也可以作为外部 bootenv.txt 环境文件的一部分。默认情况下，${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=imx8mn-phyboard-polis-peb-eval-01.dtbo imx8mn-phyboard-polis-peb-av-010.dtbo
```

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 imx8mn-phyboard-polis-peb-av-010.dtbo
u-boot=> printenv overlays
overlays=imx8mn-phyboard-polis-peb-av-010.dtbo
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 imx8mn-phyboard-polis-peb-av-010.dtbo
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 ${no_overlays} 变量设置为 *1*。

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

扩展命令

使用 U-Boot 扩展命令能够轻松加载由回调函数 extension_board_scan() 检测并添加到列表中的 overlay。以这种方式应用的 overlay 会禁用核心板上未贴装的组件。检测是通过读取出厂 EEPROM 数据（EEPROM SoM Detection）来实现的。

这取决于核心板型号是否会应用任何设备树 overlay。要检查在 U-Boot 中运行的 SoM 是否会应用 overlay，请运行：

```
u-boot=> extension scan
Found 1 extension board(s).
u-boot=> extension list
Extension 0: phyCORE-i.MX8MN no SPI flash
        Manufacturer:           PHYTEC
        Version:
        Devicetree overlay:     imx8mn-phycore-no-spiflash.dtbo
        Other information:      SPI flash not populated on SoM
```

如果没有可用的 EEPROM 数据，则不加载任何设备树 overlay。

为了禁止在启动时自动运行扩展命令，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=kirkstone

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

要查找本文中所述的 PHYTEC 的 phyCORE-i.MX 8M Nano BSP 支持的开发板和核心板，请访问 our BSP 网页，并在下载部分点击相应的 BSP 版本。在这里，您可以在”Hardware Article Number” 列中找到所有支持的硬件，并在”Machine Name” 下的相应单元格中找到正确的”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 8M Nano BSP 设备树概念部分，以了解我们的 i.MX 8 BSP 设备树模型。

以下部分概述了 i.MX 8 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 8M Nano 引脚复用

该 i.MX 8M Nano 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 8M Nano 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 8M Nano 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。

以下是 imx8mn-phyboard-polis.dts 中 UART1 设备的引脚复用示例：

```
pinctrl_uart1: uart1grp {
        fsl,pins = <
```

(续下页)

(接上页)

```
                MX8MN_IOMUXC_SAI2_RXFS_UART1_DCE_TX     0x00
                MX8MN_IOMUXC_SAI2_RXC_UART1_DCE_RX      0x00
                MX8MN_IOMUXC_SAI2_RXD0_UART1_DCE_RTS_B  0x00
                MX8MN_IOMUXC_SAI2_TXFS_UART1_DCE_CTS_B  0x00
        >;
};
```

字符串的第一部分 MX8MN_IOMUXC_SAI2_RXFS_UART1_DCE_TX 指定了引脚（在此示例中为 SAI2_RXFS）。字符串的第二部分 (UART1_DCE_RX) 是该引脚的复用选项。引脚设置值（右侧的十六进制值）定义了引脚的不同模式，例如，内部上拉电阻是否被激活。在当前配置下，内部电阻被禁用。

7.2 RS232/RS485

i.MX 8M Nano SoC 提供最多 4 个 UART 单元。PHYTEC 开发板支持不同数量 UART 单元。UART1 也可以用作 RS-485。为此，需要正确设置*bootmode switch (S1)*:

表 1: 切换 UART1 的 RS485 和 RS232 模式

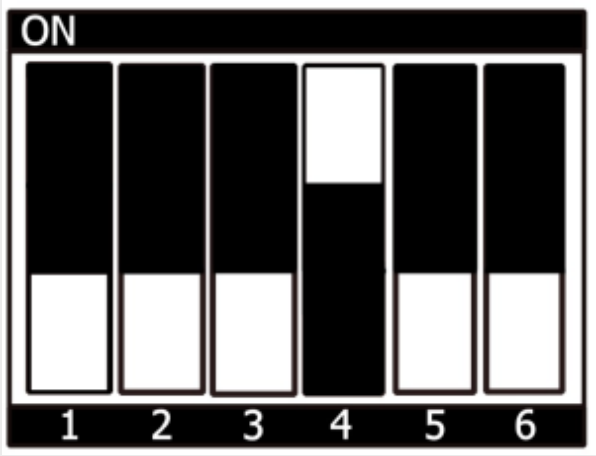


图 1: **UART1 RS485**

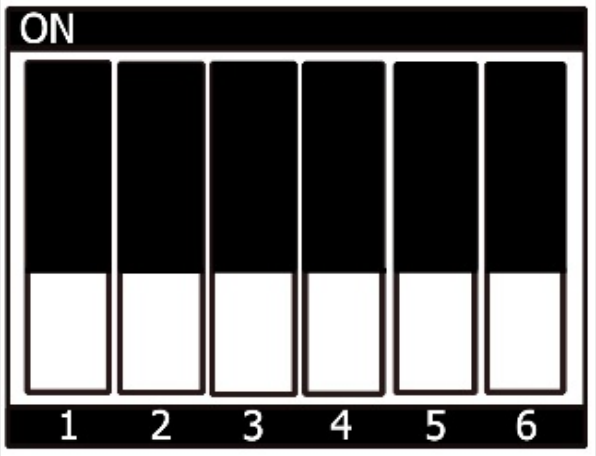


图 2: **UART1 RS232**

7.2.1 RS232

- 以人类可读的格式显示终端的当前设置:

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

- UART 接口的配置可以通过 stty 命令完成。例如:

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

- 通过简单的 echo 和 cat，可以测试基本的通信。示例:

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

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

主机应打印出”123”。

7.2.2 RS485

提示

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

为了方便测试，请查看 linux-serial-test。这个工具会通过调用 RS485 的 IOCTL，发送恒定的数据流。

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

有关 linux-serial-test 工具及其参数的更多信息，请访问此链接：linux-serial-test

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

You can show the current config with:

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

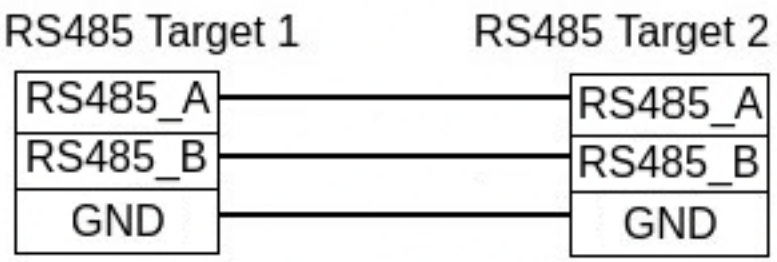
You can show all options with:

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

Linux kernel 文档描述了如何在 C 代码中调用 IOCTL：https://www.kernel.org/doc/Documentation/serial/serial-rs485.txt

RS485 half-duplex

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



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

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

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

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

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

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

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

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

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

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

RS232 和 RS485 的设备树：https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/imx8mn-phyboard-polis.dts?h=v5.15.71_2.2.2-phy3#n220

7.3 网络

phyBOARD-Polis-i.MX 8M Nano 提供一个千兆以太网接口。

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 地址可以在 *.network 文件中进行配置。eth0 的默认 IP 地址和子网掩码为：

```
eth0: 192.168.3.11/24
```

根据您的硬件配置，设备树的以太网设置可能会分为两个文件：核心板的 DT 文件和底板的 DT。FEC 以太网 IP 核心的设备树设置，其中以太网 PHY 被集成在核心板上，可以在这里找到：https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/imx8mn-phycore-som.dtsi?h=v5.15.71_2.2.2-phy3#n50。

7.3.1 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
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:~$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 50:2D:F4:19:D6:33
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

- 临时调整 eth0 的配置：

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

7.3.2 无线局域网

为了支持 WLAN 和蓝牙，我们使用来自 LSR 的 Sterling-LWB 模块。该模块支持 2.4 GHz，并且可以在多种模式下运行，如客户端模式、使用 WEP、WPA、WPA2 加密的接入点（AP）模式等。有关该模块的更多信息，请访问 https://connectivity-staging.s3.us-east-2.amazonaws.com/2019-09/CS-DS-SterlingLWB%20v7_2.pdf

连接到 WLAN 网络

首先设置您所在国家的正确监管域：

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

你将会看到：

```
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)
```

设置无线接口：

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

现在您可以扫描可用的网络：

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

您可以使用一个跨平台的客户端，名为 wpa_supplicant，支持 WEP、WPA 和 WPA2，以建立加密连接。

为此，请将网络凭据添加到文件 `/etc/wpa_supplicant.conf` 中：

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

现在可以建立连接：

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

这会得到如下输出：

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

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

首先，创建目录：

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

然后在 /etc/systemd/network/10-wlan0.network 中添加以下配置片段:

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

[Network]
DHCP=yes
```

现在，请重启网络守护进程以使配置生效:

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

7.3.3 蓝牙

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

```
target:~$ hciconfig hci0 up

target:~$ hciconfig -a

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

现在您可以扫描环境中的可见蓝牙设备。在默认配置下，蓝牙是不可见的。

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

可见性

要激活可见性:

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

要禁用可见性:

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

连接

```
target:~$ bluetoothctl
[bluetooth]# discoverable on
Changing discoverable on succeeded
[bluetooth]# pairable on
Changing pairable on succeeded
[bluetooth]# agent on
Agent registered
[bluetooth]# default-agent
```

(续下页)

(接上页)

```
Default agent request successful
[bluetooth]# scan on
[NEW] Device XX:XX:XX:XX:XX:XX <name>
[bluetooth]# connect XX:XX:XX:XX:XX:XX
```

备注

如果连接失败并出现错误信息："连接失败：org.bluez.Error.Failed"，请尝试使用以下命令重新启动 PulseAudio：

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

7.4 SD/MMC 卡

该 i.MX 8M Nano 支持一个用于 SD 卡和 MMC 卡的接口，作为 linux 通用块设备。这些设备可以像其他任何块设备一样使用。

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！

插入 SD/MMC 卡后，内核将在/dev 中生成新的设备节点。完整设备可以通过其/dev/mmcblk1 设备节点访问。SD/MMC 卡的分区将显示为：

```
/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。

MMC（SD 卡插槽）接口的 DT 配置：https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/imx8mn-phyboard-polis.dts?h=v5.15.71_2.2.2-phy3#n301

eMMC 接口的 DT 配置可以在这里找到：https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/imx8mn-phycore-som.dtsi?h=v5.15.71_2.2.2-phy3#n309

7.5 eMMC 设备

PHYTEC 模块如 phyCORE-i.MX 8M Nano 配备了 eMMC 存储芯片作为主要存储。eMMC 设备使用多层单元（MLC）或三层单元（TLC）技术来实现存储，并集成了处理 ECC 和磨损均衡的存储控制器。它们通过 SD/MMC 接口连接到 i.MX 8M Nano ，并在 Linux 内核中作为块设备表示，如 SD 卡、优盘或硬盘。

电气和协议规范由 JEDEC 提供（https://www.jedec.org/standards-documents/technology-focus-areas/flash-memory-ssds-ufs-emmc/e-mmc）。eMMC 制造商的数据手册相对较简单，旨在与支持的 JEDEC

eMMC 标准版本一起阅读。

PHYTEC 目前使用 JEDEC 版本 5.0 和 5.1 的 eMMC 芯片。

7.5.1 扩展 CSD 寄存器

通过扩展 CSD 寄存器可以读取 eMMC 设备其他的信息和配置。有关寄存器的详细列表，请参阅制造商的数据手册和 JEDEC 标准。

在 Linux 用户空间中，您可以查询寄存器：

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

你将会看到：

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

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

7.5.2 使能后台操作 (BKOPS)

与原始 NAND Flash 相比，eMMC 设备包含一个闪存传输层（FTL），该层负责处理原始 MLC 或 TLC 的磨损均衡、块管理和错误更正码（ECC）。这需要定期执行一些维护任务（例如擦除未使用的块）。这些任务被称为 **后台操作（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。

JEDEC 标准自版本 v4.41 起规定了一种方法，主机可以手动触发 BKOPS。有关更多详细信息，请参阅 JEDEC 标准章节 “Background Operations” 以及 eMMC 数据手册中寄存器 BKOPS_EN（寄存器：163）和 BKOPS_START（寄存器：164）的描述。

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。

自 v3.7 版本以来，Linux 内核已经实现了触发后台操作的机制。您只需在 eMMC 设备上启用 BKOPS_EN（详细信息见下文）。

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/mmcblk2 | 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/mmcblk2
```

- 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。

7.5.3 可靠写入

有两种不同的可靠写入选项：

1. 对整个 eMMC 设备/分区的可靠写入方式。
2. 单次写的可靠写入方式。

小技巧

不要将 eMMC 分区与 DOS、MBR 或 GPT 分区表的分区混淆（请参阅前一节）。

第一个可靠写入方式大多数情况下已经在 phyCORE-i.MX 8M Nano SoM 上挂载的 eMMC 上被启用。想要在运行的开发板上检查这一点：

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

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。

7.5.4 调整 ext4 根文件系统的大小

在将 SD 卡镜像写入 eMMC 时，ext4 文件系统分区没有扩展到 eMMC 的末尾。可以使用 parted 来扩展根分区。这个示例适用于任何块设备，例如 eMMC、SD 卡或硬盘。

- 获取当前设备大小：

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

- 输出如下：

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

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

- 使用 parted 将文件系统分区调整为设备的最大大小：

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

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

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

- 将文件系统调整为新的分区大小：

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

在文件系统挂载时可以增加其大小。但您也可以从 SD 卡启动板，然后在 eMMC 分区未挂载时调整文件系统的大小。

7.5.5 启用伪 SLC 模式

eMMC 设备使用 MLC 或 TLC 来存储数据。与 NAND Flash 中使用的 SLC 相比，MLC 或 TLC 在成本更低的情况下，可靠性较低且错误率较高。

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。

- 首先使用以下命令获取 eMMC 设备的当前大小：

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

  你将收到：

  ```
  BYT;
  /dev/mmcblk2:63652757504B: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/mmcblk2 | 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/mmcblk2
  ```

  你将获得：

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

- 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/mmcblk2 | 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/mmcblk2 unit B print
BYT;
/dev/mmcblk2:31742492672B:sd/mmc:512:512:unknown:MMC S0J58X:;
```

7.5.6 擦除设备

可以直接擦除 eMMC 设备，而不是通过写零覆盖。eMMC 块管理算法将擦除底层的 MLC 或 TLC，或者将这些块标记为可丢弃。设备上的数据将丢失，并将被读取为零。

- SD 卡启动后执行：

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

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/mmcblk2 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！

7.5.7 eMMC Boot 分区

eMMC 设备包含四个不同的硬件分区：User 分区、boot1 分区、boot2 分区和 rpmb 分区。

User 分区在 JEDEC 标准中称为用户数据区，是主要的存储分区。分区 boot1 和 boot2 可以用于存放 bootloader，并且更可靠。i.MX 8M Nano 使用哪个分区加载 bootloader 由 eMMC 设备的引导配置控制。分区 rpmb 是一个小分区，只能通过受信任的机制访问。

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

不要将 eMMC 分区与 DOS、MBR 或 GPT 分区表的分区混淆。

当前的 PHYTEC BSP 没有使用 eMMC 设备的额外分区功能。U-Boot 被烧写到用户分区的开始位置。U-Boot 环境被放置在 U-Boot 之后的固定位置。使用 MBR 分区表创建两个分区，一个是 FAT32 引导分区，另一个是 ext4 根文件系统分区。它们位于 U-Boot 和 U-Boot 环境之后。FAT32 引导分区包含内核和设备树。

使用 eMMC 时，可以利用专用的 boot 分区备份存储 bootloader。U-Boot 环境仍然位于第一个分区之前的用户区。用户区仍然在出厂时包含 bootloader。下面是一个示例，演示如何将 bootloader 烧写到两个启 boot 分区中的一个，并通过用户空间命令切换启动设备。

7.5.8 通过用户空间命令

在主机上运行：

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

默认情况下，boot1 和 boot2 分区是只读的。要从用户空间写入它们，您必须在 sysfs 中禁用 force_ro。

要手动将 bootloader 写入 eMMC boot 分区，首先禁用写保护：

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

将 bootloader 写入 eMMC boot 分区：

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

下表是 i.MX 8M Nano SoC 的偏移量：

SoC	User 分区偏移量	Boot 分区偏移量	eMMC 设备	bootloader 文件名
i.MX 8M Nano	32 kiB	0 kiB	/dev/mmcblk2	imx-boot

之后使用 mmc 工具从用户空间设置引导分区：

(对于'boot0')：

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

(对于'boot1')：

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

要禁用从 eMMC boot 分区启动，只需输入以下命令：

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

返回到 User 分区启动：

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

7.5.9 调整 ext4 根文件系统的大小

fdisk 可以用来扩展根文件系统。这个例子适用于任何块设备，比如 eMMC、SD 卡或硬盘。

- 获取当前设备大小：

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

- 输出如下：

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

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

- 使用 fdisk 删除并创建一个最大化使用设备容量的分区：

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

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

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

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

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

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

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

Command (m for help): n
Partition type
   p   primary partition (1-4)
```

(续下页)

(接上页)

```
   e   extended
p
Partition number (1-4): 2
First sector (32-14876671, default 32): 141456
Last sector or +size{,K,M,G,T} (141456-14876671, default 14876671):
Using default value 14876671

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

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

可以在文件系统挂载时修改文件系统的大小。这是一个在线调整大小的操作。但您也可以从 SD 卡启动，然后在 eMMC 分区未挂载时调整其文件系统的大小。此外，必须重启板子，以便读取新的分区表。

7.6 SPI 主设备

i.MX 8M Nano 控制器包含一个 FlexSPI 和一个 ECSPI IP 核。FlexSPI 主控制器支持两个 SPI 通道，最多可连接 4 个设备。每个通道支持单通道/双通道/四通道/八通道模式的数据传输（1/2/4/8 条数据线）。ECSPI 控制器支持 3 个 SPI 接口，每个接口都有一个专用的 CS（chip select）引脚。由于 CS 也可通过 GPIO 实现，因此每个通道上可以连接多个设备。

7.6.1 SPI NOR 烧写

phyCORE-i.MX 8M Nano 配备有一个 QSPI NOR Flash，该 Flash 连接到 i.MX 8M Nano 的 FlexSPI 接口。QSPI NOR Flash 可用于启动。有关烧写和启动模式设置的详细信息，请参见不同的章节。由于 SPI NOR Flash 的空间有限，因此仅可存储 bootloader。默认情况下，内核、设备树和根文件系统来自 eMMC。

bootloader 程序通过 EEPROM 数据检测是否安装了 SPI Flash。如果没有安装 SPI Flash，则在启动期间应用设备树 overlay，通过扩展命令禁用 SPI Flash 设备树节点。如果没有可用的 EEPROM 数据，SPI NOR Flash 节点将始终启用。有关更多信息，请参阅设备树 overlay 部分。

bootloader 程序还可以通过内核的 mtdparts 启动参数修改设备树，将 SPI MTD 分区表传递给 Linux。BSP 中的默认分区布局设置为：

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

这是一个预定义的 bootloader 环境变量，可以在运行时更改。从 Linux 用户空间，可以通过/dev/mtd<N>设备访问 NOR Flash 分区，其中 <N> 是与要访问的 NOR Flash 分区相关联的 MTD 设备编号。要找到分区的正确 MTD 设备编号，请在目标上运行：

```
root@phyboard-polis-imx8mn-2:~$ mtdinfo --all
Count of MTD devices:           4
Present MTD devices:            mtd0, mtd1, mtd2, mtd3
Sysfs interface supported:      yes

mtd0
Name:                           u-boot
Type:                           nor
Eraseblock size:                65536 bytes, 64.0 KiB
```

(续下页)

(接上页)

```
Amount of eraseblocks:          60 (3932160 bytes, 3.7 MiB)
Minimum input/output unit size: 1 byte
Sub-page size:                  1 byte
Character device major/minor:   90:0
Bad blocks are allowed:         false
Device is writable:             true

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

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

mtd3
Name:                           none
Type:                           nor
Eraseblock size:                65536 bytes, 64.0 KiB
Amount of eraseblocks:          448 (29360128 bytes, 28.0 MiB)
Minimum input/output unit size: 1 byte
Sub-page size:                  1 byte
Character device major/minor:   90:6
Bad blocks are allowed:         false
Device is writable:             true
```

它列出了所有 MTD 设备及其对应的分区名称。闪存节点在模块 DTS 中的 SPI 主节点内定义。SPI 节点包含连接到此 SPI 总线的所有设备，在这种情况下只有 SPI NOR Flash。

设备树中 SPI 主节点的定义可以在这里找到：

https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/imx8mn-phycore-som.dtsi?h=v5.15.71_2.2.2-phy3#n78

7.7 GPIOs

phyBOARD-Polis 具有一组专门用于 GPIO 的引脚。这些引脚直接连接到 i.MX 8M Nano 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 8M Nano 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 gpiochip0
```

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

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

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

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

- gpioset 的帮助文本显示了可能的选项：

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

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

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

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

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。

7.7.1 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` 。

您也可以将此选项添加到您在 Linux 内核源代码 `arch/arm64/configs/` 目录下使用的 defconfig 中。例如，我们基于 NXP 的 BSP 版本，这个 defconfig 可以是 `imx8_phytec_distro.config`

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

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

设备树 imx8mn-phyboard-polis.dts 中一些 GPIO 引脚的管脚复用：

```
pinctrl_leds: leds1grp {
        fsl,pins = <
                MX8MN_IOMUXC_GPIO1_IO01_GPIO1_IO1       0x16
                MX8MN_IOMUXC_GPIO1_IO14_GPIO1_IO14      0x16
                MX8MN_IOMUXC_GPIO1_IO15_GPIO1_IO15      0x16
        >;
};
```

7.8 LED 灯

如果有任何 LED 灯连接到 GPIO 管脚，您可以通过特定的 LED 驱动程序接口访问它们，而不是使用通用的 GPIO 接口（请参见 GPIO 部分）。您将通过 `/sys/class/leds/` 而不是 `/sys/class/gpio/` 来访问它们。LED 的最大亮度可以从 `max_brightness` 文件中读取。brightness 文件将设置 LED 的亮度（取值范围从 0 到 max_brightness）。大多数 LED 硬件上不支持调整亮度，所以在所有非零亮度下都会点亮。

下面是一个简单的例子。

要获取所有可用的 LED，请输入：

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

这里的 LED 灯包括蓝色的 mmc、绿色的心跳和红色的 emmc，它们都在 phyBOARD-Polis 上。

- 打开 LED 灯：

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

- 关闭 LED：

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

GPIO 配置的设备树配置可以在这里找到：https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/imx8mn-phyboard-polis.dts?h=v5.15.71_2.2.2-phy3#n45

7.9 I²C 总线

该 i.MX 8M Nano 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 8M Nano 的 I²C 模块。本节描述了我们 phyBOARD-Polis 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。

I²C1 总线的 DT 配置（例如 imx8mn-phycore-som.dtsi）：https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/imx8mn-phycore-som.dtsi?h=v5.15.71_2.2.2-phy3#n106

I²C3 总线 DT 配置（例如 imx8mn-phyboard-polis.dts）：https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/imx8mn-phyboard-polis.dts?h=v5.15.71_2.2.2-phy3#n196

7.10 EEPROM

在 phyCORE-i.MX8MN 上有一个 i2c EEPROM 闪存。它有两个地址。主 EEPROM 空间（总线：I2C-0 地址：0x51）和 ID 页（总线：I2C-0 地址：0x59）可以通过 Linux 中的 sysfs 接口进行访问。主 EEPROM 和 ID 页的前 256 个字节用于板检测，不可被覆盖。覆盖保留的空间将导致启动问题。

7.10.1 phyCORE-i.MX8MN 上的 I2C EEPROM

警告

EEPROM ID 页面（总线：I2C-0 地址：0x59）和正常 EEPROM 区域的前 256 个字节（总线：I2C-0 地址：0x51）不可被擦除或修改。这将影响 bootloader 的行为。板子可能无法正确启动。

phyCORE-i.MX8MN SoM 上的 I2C EEPROM 连接到 I2C-0 总线的 I2C 地址 0x51。可以直接对该设备进行读写操作：

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

要读取并以十六进制打印 EEPROM 的前 1024 字节，请执行：

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

要用零填充 4KiB 的 EEPROM（总线：I2C-0 地址：0x51），并保留 EEPROM 数据，请使用：

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

7.10.2 EEPROM SoM 检测

在 phyCORE-i.MX8MN 上配置的 I2C EEPROM 具有一个可通过 I2C 地址 0x59 在 i2c0 上寻址的独立 ID 页面，以及一个可通过 I2C 地址 0x51 在 i2c0 上寻址的正常区域。PHYTEC 使用这个 32 字节的数据区域来存储关于 SoM 的信息，包括 PCB 版本和配置。

在启动的早期阶段读取 EEPROM 数据。它用于选择正确的 DDR RAM 配置。这使得可以使用相同的 bootloader 镜像来支持不同的 RAM 大小，并自动选择正确的 DTS overlay。

如果 EEPROM ID 页面数据和正常区域的前 256 个字节被删除，bootloader 程序将回退到 phyCORE-i.MX8MN Kit RAM 设置，即 1GiB RAM。

警告

EEPROM ID 页面（总线：I2C-0 地址：0x59）和正常 EEPROM 区域的前 256 个字节（总线：I2C-0 地址：0x51）不可被擦除或修改。这将影响 bootloader 的行为。板子可能无法正确启动。

使用 API 修订版 2 数据格式烧写的核心板将在早期启动阶段打印出有关模块的信息。

7.10.3 恢复 EEPROM 数据

硬件数据已预先写入两个 EEPROM 数据空间。如果您不小心删除或覆盖了 Normal 区域，可以将 ID 区域的硬件检查数据复制到正常区域。

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

备注

如果您删除了两个 EEPROM 空间，请联系我们的支持团队！

DT 表示，例如在 phyCORE-i.MX 8M Nano 文件 imx8mn-phycore-som.dtsi 中，可以在我们的 PHYTEC git 中找到：https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/imx8mn-phycore-som.dtsi?h=v5.15.71_2.2.2-phy3#n259

7.11 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
```

7.11.1 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。

7.11.2 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 0x11.
  ```

  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_CNT                7
```

- 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。

I²C RTC 的设备树表示：https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/imx8mn-phycore-som.dtsi?h=v5.15.71_2.2.2-phy3#n267

7.12 USB 主控制器

i.MX 8M Nano 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，通过提供一种数据传输机制，使 USB 设备之间的线路/总线速度可达到 480 Mbps（高速'HS'）。

该 i.MX 8M Nano SoC 具有一个设置为 OTG 模式的 USB 控制器 IP。要使用 Micro USB / OTG 端口，拨码开关 S1 的第 5 位必须设置为开启。

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。

7.13 USB OTG

大多数 PHYTEC 板提供 USB OTG 接口。USB OTG 端口会自动作为 USB 设备或 USB 主机工作。模式取决于连接到 USB OTG 端口的 USB 硬件。例如，如果将 USB 大容量存储设备连接到 USB OTG 端口，该设备将显示为 `/dev/sda` 。

7.13.1 作为 USB 设备

为了将开发板作为 USB 设备连接到 USB 主机（例如 PC），您需要配置相应的 USB gadget。通过 USB configfs，您可以定义 USB gadget 的参数和功能。BSP 包含作为 kernel module 的 USB configfs 支持。

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

例子:

- 首先，定义参数，例如 USB Vendor 和 product ID，并为英语（0x409）设置信息字符串：

提示

为了节省时间，请复制这些命令并在脚本中执行它们

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

- 接下来，为大容量存储 gadget 创建一个文件：

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

- 现在你可以创建你想要使用的功能：

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

- *acm*: 串行设备 gadget，创建类似 `/dev/ttyGS0` 的串行接口。
- *ecm*: 以太网 gadget，创建以太网接口，例如 usb0
- *mass_storage*: 主机可以像处理其他 USB 大容量存储设备一样，对设备的大容量存储进行分区、格式化和挂载。
- 将定义的功能绑定到配置：

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

- 最后，使用以下命令启动 USB gadget：

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

如果您的系统有多个 USB 设备或 OTG 端口，您可以将正确的端口传递给 USB 设备控制器（UDC）。

- 要停止 USB gadget 并解除绑定已使用的功能，请执行：

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

USB 接口在内核设备树 |dt-carrierboard|.dts 中配置为 Host。请参见：https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/imx8mn-phyboard-polis.dts?h=v5.15.71_2.2.2-phy3#n264

7.14 CAN FD

phyBOARD-Polis 支持两个 flexCAN 接口，支持 CAN FD。这些接口支持 Linux 标准 CAN 框架，该框架建立在 Linux 网络层之上。使用这个框架，CAN 接口表现得像普通的 Linux 网络设备，同时具备一些 CAN 特有的附加功能。更多信息可以在 Linux 内核文档中找到：https://www.kernel.org/doc/html/latest/networking/can.html

提示

phyBOARD-Polis 具有一个通过 SPI 连接的外部 CAN FD 芯片 MCP2518FD。由于使用了 SPI，限制了 CAN FD 的数据传输速率上限。

提示

在 phyBOARD-Polis-i.MX8MN 上，如果需要，可以通过将 S5 设置为 ON 来启用端接电阻。

- 使用：

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

  查看接口的状态。两个 CAN 接口显示为 can0 和 can1。

- 要获取有关 can0 的信息，例如比特率和错误计数器，请输入：

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

  can0 的信息将如下所示：

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

(续下页)

(接上页)

```
↪1 gso_max_size 65536 gso_max_segs 65535
    RX: bytes  packets  errors  dropped overrun mcast
    0          0        0       0       0       0
    TX: bytes  packets  errors  dropped carrier collsns
    0          0        0       0       0       0
```

输出包含一组标准参数，这些参数也适用于以太网接口，因此并非所有参数对于 CAN 都是相关的（例如 MAC 地址）。以下输出参数包含有用的信息：

can0	接口名称
NOARP	CAN 无法使用 ARP 协议
MTU	最大传输单元
RX packets	接收的数据包数量
TX packets	发送的数据包数量
RX bytes	接收字节数
TX bytes	发送字节数
errors...	总线错误统计信息

CAN 配置是在 systemd 配置文件 `/lib/systemd/network/can0.network` 中完成的。为了持久化更改（例如，默认比特率），请在 BSP 中更改根文件系统下的 `./meta-ampliphy/recipes-core/systemd/systemd-conf/can0.network` 中的配置，并重新编译根文件系统。

```
[Match]
Name=can0

[Can]
BitRate=500000
```

比特率也可以手动更改，例如，设置为灵活比特率（flexible bitrate）：

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

您可以使用 cansend 发送消息，或使用 candump 接收消息：

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

要生成用于测试目的的随机 CAN 流量，请使用 cangen：

```
target:~$ cangen
```

`cansend --help` 和 `candump --help` 提供了关于选项和用法的帮助信息。

警告

mcp2518fd SPI 到 CAN FD 只支持从 125kB/s 开始的波特率。可以选择更慢的速率，但可能无法正常工作。

imx8mn-phyboard-polis.dts 的设备树 CAN 配置: https://git.phytec.de/linux-imx/tree/arch/arm64/boot/dts/freescale/imx8mn-phyboard-polis.dts?h=v5.15.71_2.2.2-phy3#n125

7.15 电源管理

7.15.1 CPU 核心频率调节

i.MX 8M Nano 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 8M Nano 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 8 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` 。

7.15.2 CPU 核心管理

该 i.MX 8M Nano SoC 芯片上可以有多个处理器核心。例如，该 i.MX 8M Nano 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
```

7.15.3 挂起到 RAM

phyCORE-i.MX8MN 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/ttymxc2/power/wakeup
target:~$ echo mem > /sys/power/state
```

7.16 热管理

7.16.1 U-Boot

之前 U-Boot 中的温度控制不够理想。现在，U-Boot 增加了温度关机功能，以防止在启动过程中板子过热。关机发生的温度与内核中的温度一致。

当前温度的各个温度范围在启动日志中显示：

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

7.16.2 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）。

本节描述了如何在 i.MX 8M Nano SoC 平台上使用热管理内核 API。i.MX 8 具有用于 SoC 的内部温度传感器。

- 当前温度可以以毫摄氏度为单位读取：

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

- 例如，你将得到：

  ```
  49000
  ```

imx_thermal 内核驱动注册了两个触发点。这些触发点根据 CPU 型号的不同而有所不同。主要区分工业版和商业版。

	商业	工业
被动（警告）	85°C	95°C
严重（关机）	90°C	100°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实施冷却措施。内核中可用的热政策（也称为 thermal governor）包括：Step Wise、Fair Share、Bang Bang 和 Userspace。BSP 中使用的默认策略是 Step Wise。如果 sysfs 文件 temp 中的 SoC 温度值高于 *trip_point_0* ，则 CPU 频率将设置为最低 CPU 频率。当 SoC 温度降到 *trip_point_0* 以下时，限制将被解除。

备注

由于我们安装了不同温度等级的 CPU，因此热触发点的实际值可能会有所不同。

7.16.3 GPIO 风扇

备注

Only GPIO fan supported.

一个 GPIO 控制的风扇可以连接到 phyBOARD-Polis-i.MX 8M Nano。该 SoC 包含一个温度传感器，该传感器被用于热频率调节。风扇无法通过内核进行控制。我们使用 lmsensors 和 hwmon 来实现这一点。lmsensors

定期读取温度，并在可配置的阈值下启用或禁用风扇。对于 phyBOARD-Polis-i.MX 8M Nano，这个阈值是 60°C。

设置可以在配置文件中进行配置：

```
/etc/fancontrol
```

风扇控制在启动时由 systemd 服务启动。可以通过以下方式禁用它：

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

7.17 看门狗

PHYTEC i.MX 8M Nano 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。

7.17.1 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
```

7.18 片上一次性可编程控制器 (OCOTP_CTRL) - eFuse

该 i.MX 8M Nano 提供一次性可编程 fuse，用于存储信息，例如 MAC 地址、启动配置和其他永久设置（在 i.MX 8M Nano reference manual 中称为 “片上 OTP 控制器 (OCOTP_CTRL)”）。以下列表摘自 i.MX 8M Nano reference manual，包括 OCOTP_CTRL 中的一些有用寄存器（基地址为 0x30350000）：

名称	Bank	字	内存偏移量为 0x30350000	描述
OCOTP_MAC_AI	9	0	0x640	包含 ENET0 MAC 地址的低 32 位
OCOTP_MAC_AI	9	1	0x650	包含 ENET0 MAC 地址的高 16 位和 ENET1 MAC 地址的低 16 位
OCOTP_MAC_AI	9	2	0x660	包含 ENET1 MAC 地址的高 32 位

关于 OCOTP_CTRL 中的 fuse 与启动配置之间的完整列表和详细映射，请参阅 i.MX 8M Nano Security Reference Manual 中的”Fuse Map” 部分。

7.18.1 在 uBoot 中读取 fuse 的值

您可以使用内存映射的 shadow 寄存器读取 fuse 寄存器。要计算内存地址，请使用以下公式计算：

OCOTP_MAC_ADDR:

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

7.18.2 在 Linux 中读取 fuse 值

要访问 Linux 中的 fuse 内容，NXP 提供了 NVMEM_IMX_OCOTP 模块。所有内存映射的 shadow 寄存器的 fuse 内容可以通过 sysfs 访问：

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