# phyCORE-i.MX 93 BSP Manual DRAFT

PHYTEC Messtechnik GmbH

2025 年 10 月 02 日

# Contents

| L-1069e.Ax i.MX 93 BSP 手册模板 | |
|---|---|
| 文档标题 | L-1069e.Ax i.MX 93 BSP 手册模板 |
| 文档类型 | BSP 手册 |
| 型号 | L-1069e.Ax |
| Yocto 手册 | |
| 发布日期 | XXXX/XX/XX |
| 母文档 | L-1069e.Ax i.MX 93 BSP 手册模板 |

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 |
|---|---|---|---|
| | | | |

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 **phyBOARD-Segin i.MX 93 and phyBOARD-Nash i.MX 93 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-91-93/#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.MX93-PD24.2.1 的所有 Machine 及其对应的 Article Numbers(产品型号)：请参见 网页。

如果您在 “Supported Machines” 部分选择了特定的 **Machine Name** ，您可以查看该 Machine 下可用的 **Article Number(产品型号)** 以及硬件信息的简短描述。如果您只有硬件的 **Article Number** ，可以将 **Machine Name** 下拉菜单留空，仅选择您的 **Article Number** 。现在它会显示您特定硬件所需的 **Machine Name** 。

**小技巧**

**本 BSP 手册中的终端示例仅针对 phyBOARD-Segin i.MX 93。类似的命令也可以在 phyBOARD-Nash i.MX 93 上执行。**

## 1.1 phyBOARD-Segin i.MX 93 器件

## 1.2 phyBOARD-Nash i.MX 93 器件

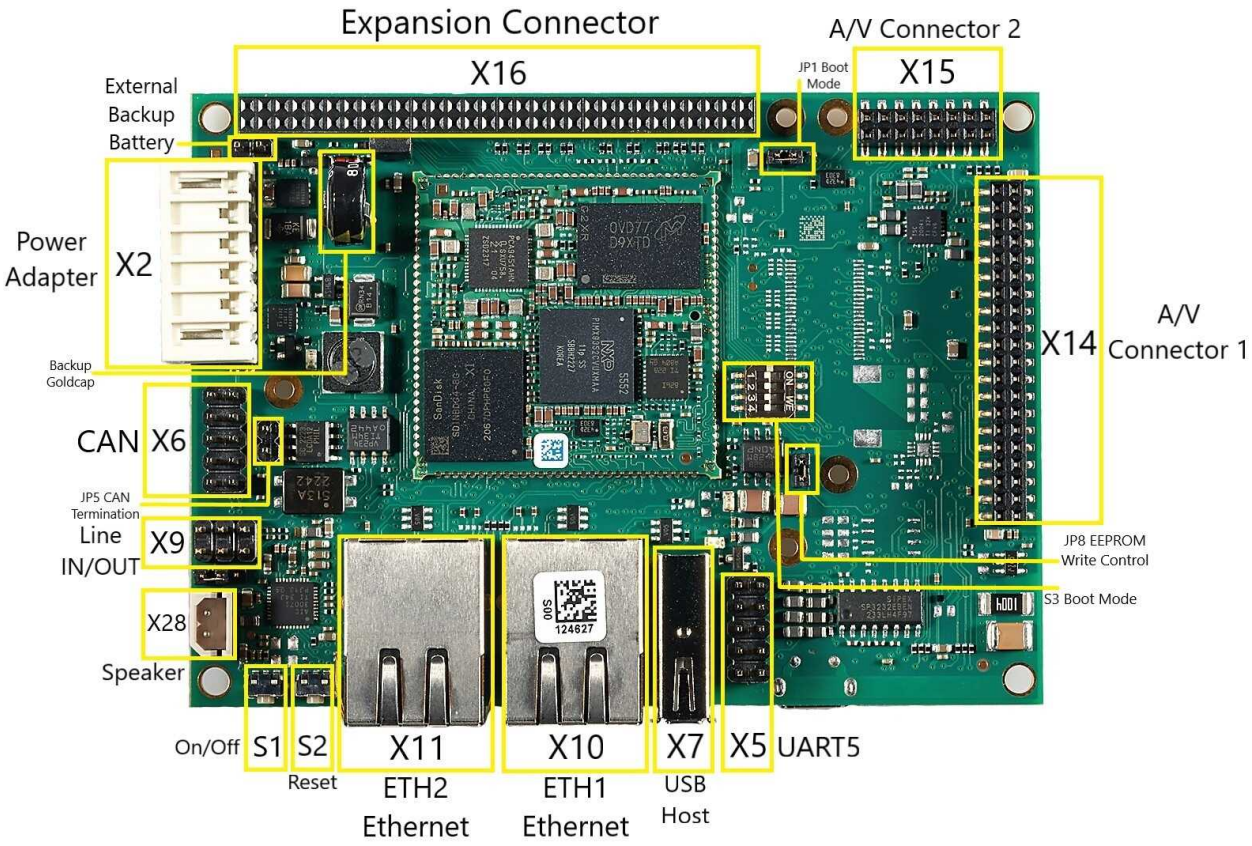



图 1: **phyBOARD-Segin i.MX 93 器件（顶部）**

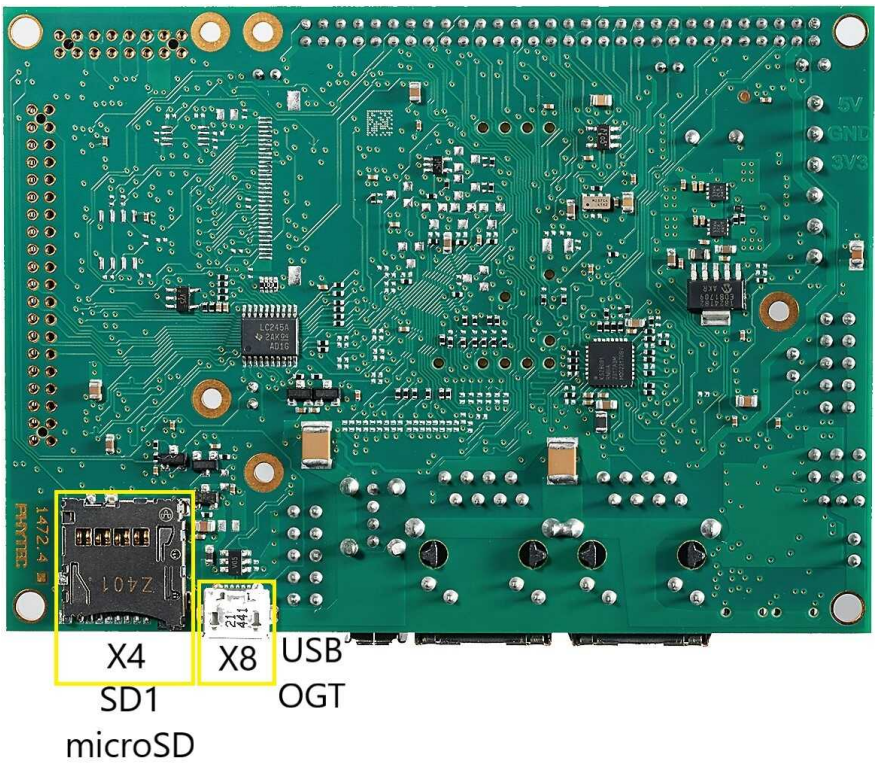



图 2: **phyBOARD-Segin i.MX 93 器件（底部）**

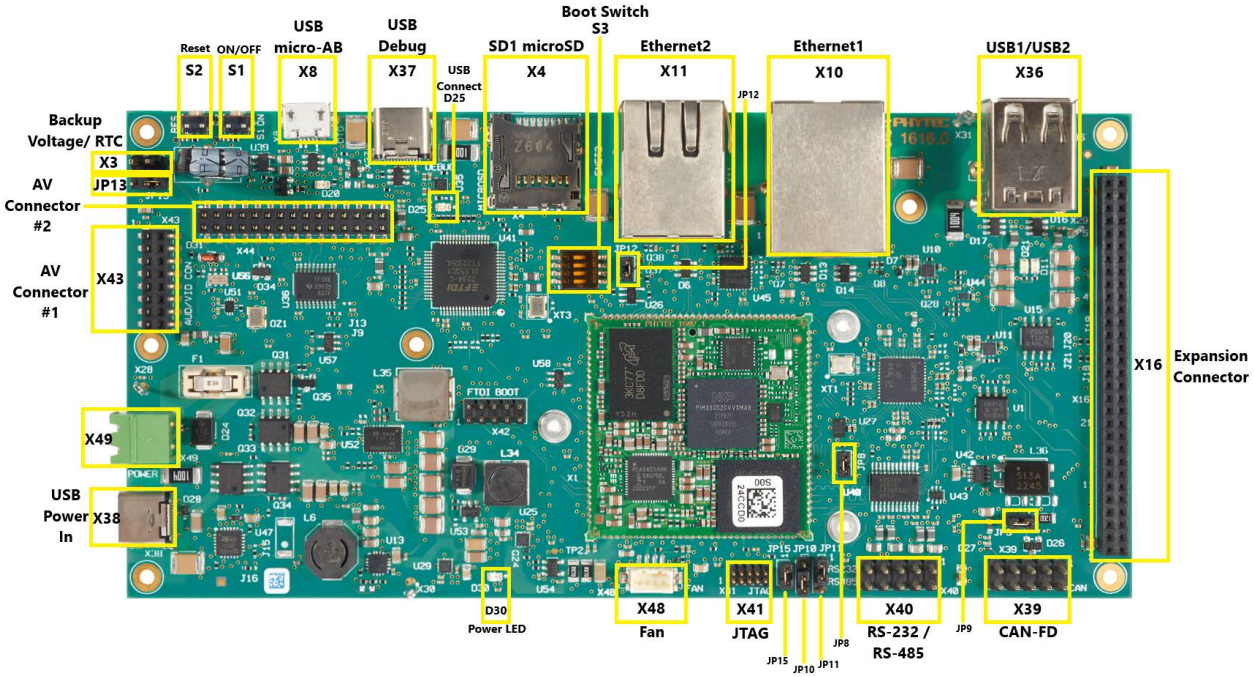



图 3: **phyBOARD-Nash i.MX 93 器件（顶部）**

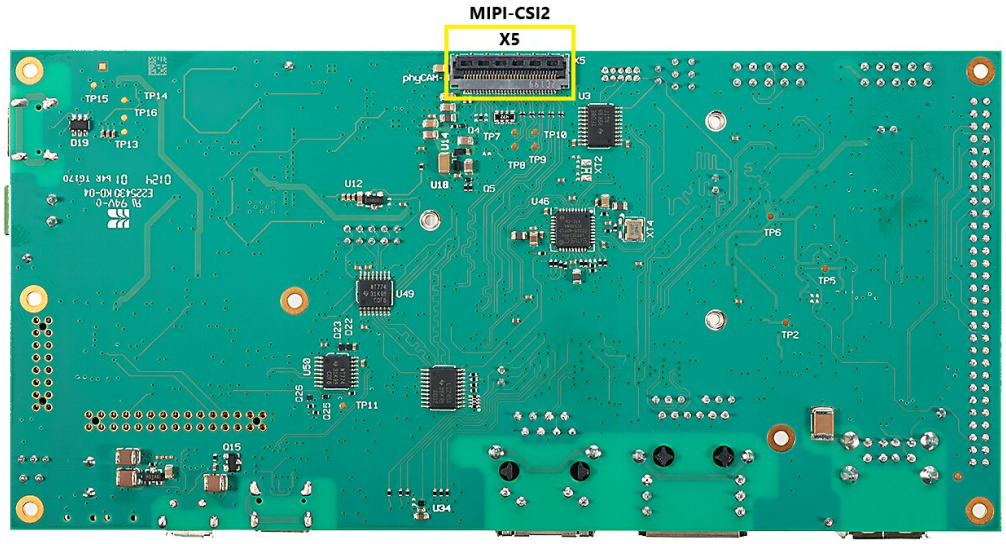



图 4: **phyBOARD-Nash i.MX 93 器件（底部）**

CHAPTER 2

# 开始使用

The **phyBOARD-Segin i.MX 93 and phyBOARD-Nash i.MX 93 Kit** is shipped with a pre-flashed SD card. It contains the phytec-qt6demo-image and can be used directly as a boot source. The e.MMC is programmed with only a U-Boot by default. You can get all sources from the PHYTEC download server. This chapter explains how to flash a BSP image to SD card and how to start the board.

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

## 2.1 下载镜像

phytec-qt6demo-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.MX93/BSP-Yocto-NXP-i.MX93-PD24.2.1/
↪images/ampliphy-vendor/phyboard-segin-imx93-2/phytec-qt6demo-image-phyboard-segin-imx93-2.rootfs.partup
host:~$ wget https://download.phytec.de/Software/Linux/BSP-Yocto-i.MX93/BSP-Yocto-NXP-i.MX93-PD24.2.1/
↪images/ampliphy-vendor/phyboard-segin-imx93-2/phytec-qt6demo-image-phyboard-segin-imx93-2.rootfs.wic.xz
```

**备注**

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

## 2.2 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 `partup`, `dd` or `bmaptool`.

### 2.2.2 Using bmaptool

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

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

通过以下命令将 WIC 镜像烧写到 SD 卡：

```
host:~$ bmaptool copy phytec-qt6demo-image-phyboard-segin-imx93-2?(.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 partup

使用 partup 烧写 SD 卡只需一个命令：

```
host:~$ sudo partup install phytec-qt6demo-image-phyboard-segin-imx93-2?(.rootfs).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 。

**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 release notes 。

**备注**

*partup* 具有清除 eMMC user 区域中特定区域的功能，我们提供的 partup 程序中用该功能擦除 U-Boot 环境变量。这是 *bmaptool* 工具所无法完成的一点，如前一部分所提到的。

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。

### 2.2.4 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-qt6demo-image-phyboard-segin-imx93-2?(.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：

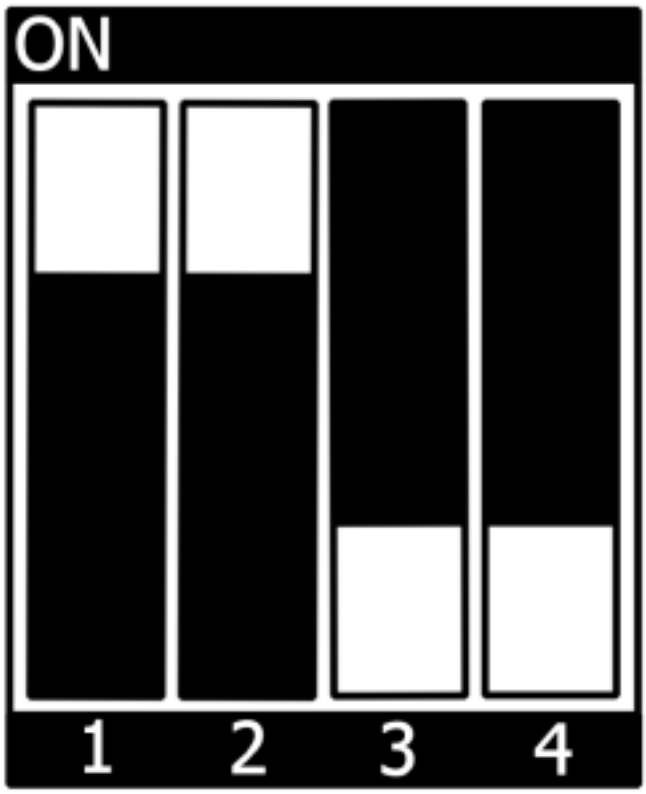
```
host:~$ xzcat phytec-qt6demo-image-phyboard-segin-imx93-2?(.rootfs).wic.xz | sudo dd of=/dev/<your_device>␣
↪bs=1M conv=fsync status=progress
```

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 (S3)* 需要设置为以下位置：




- 插入 SD 卡
- 将开发板的调试接口与您的主机连接。使用 UART1 console on PEB-EVAL-01 for **phyBOARD-Segin** and X-37 USB-C debug for **phyBOARD-Nash**。
- 给开发板通电
- 以 115200 波特率和 8N1 格式打开串口/USB 端口（您应该在终端上看到 u-boot/linux 启动信息）

CHAPTER 3

# 编译 BSP

本节将指导您使用 Yocto 和 phyLinux 脚本进行 i.MX 93 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-i.MX93 ；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=phyboard-segin-imx93-2 ./phyLinux init -p imx93 -r BSP-
↪Yocto-NXP-i.MX93-PD24.2.1
```

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 phyboard-segin-imx93-2 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-imx93.bin**: ARM 可信固件二进制文件
- **lpddr4_dmem_1d_v202201.bin, lpddr4_dmem_2d_v202201.bin, lpddr4_imem_1d_v202201.bin, lpddr4_imem_2d_v202201.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
- **imx93-phyboard-*.dtb**: 内核设备树文件
- **imx93-phy*.dtbo**: 内核设备树 overlay 文件
- **phytec-*.tar.gz**: bitbake-image 编译生成的 root 文件系统。
  - **phytec-qt6demo-image-phyboard-*-imx93-*.tar.gz**：在使用 bitbake-build 编译 `phytec-qt6demo-image` 时
  - **phytec-headless-image-phyboard-*-imx93-*.tar.gz**：在使用 bitbake-build 编译 `phytec-headless-image` 时
- **phytec-*.rootfs.wic.xz**：bitbake-image 编译生成的压缩的可引导 SD 卡镜像，包括 bootloader、设备树二进制文件（DTB）、内核和根文件系统。
  - **phytec-qt6demo-image-phyboard-*-imx93-*.rootfs.wic.xz**：在使用 bitbake-build 编译 `phytec-qt5demo-image` 时
  - **phytec-headless-image-phyboard-*-imx93-*.rootfs.wic.xz**：在使用 bitbake-build 编译 `phytec-headless-image` 时
- **imx93-11x11-evk_m33_*.bin**，Cortex-M33 MCU 的 Demo 应用程序的二进制文件；可以通过 U-Boot 或 Linux 手动加载和启动

CHAPTER 4

# 安装操作系统

## 4.1 启动模式开关 (S3)

**小技巧**

底板版本：

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

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

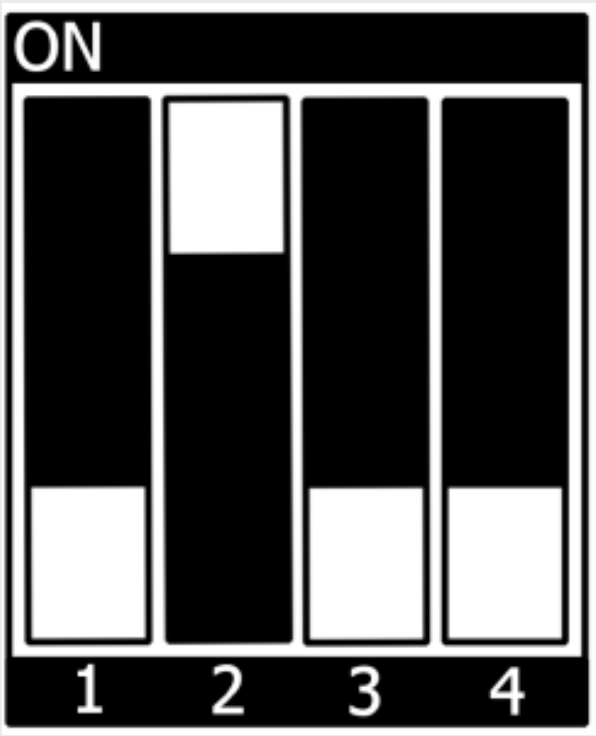



图 1: eMMC

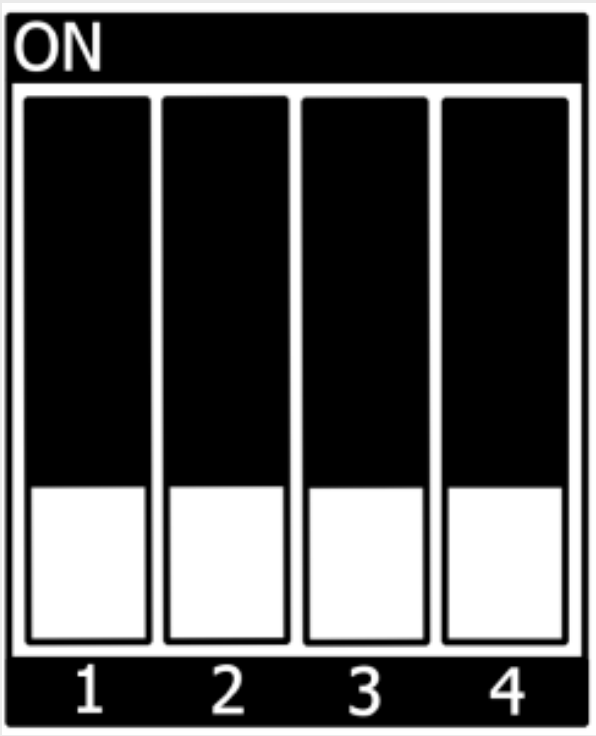



图 2: 内部 fuse

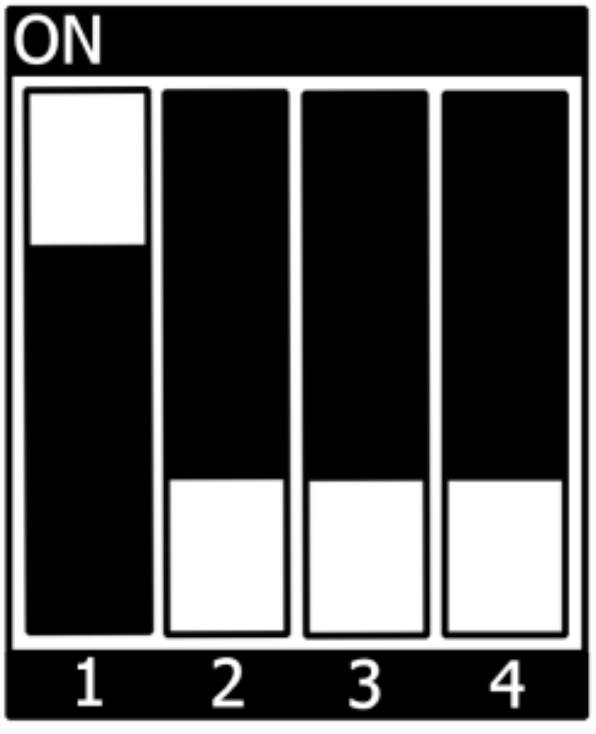



图 3: USB

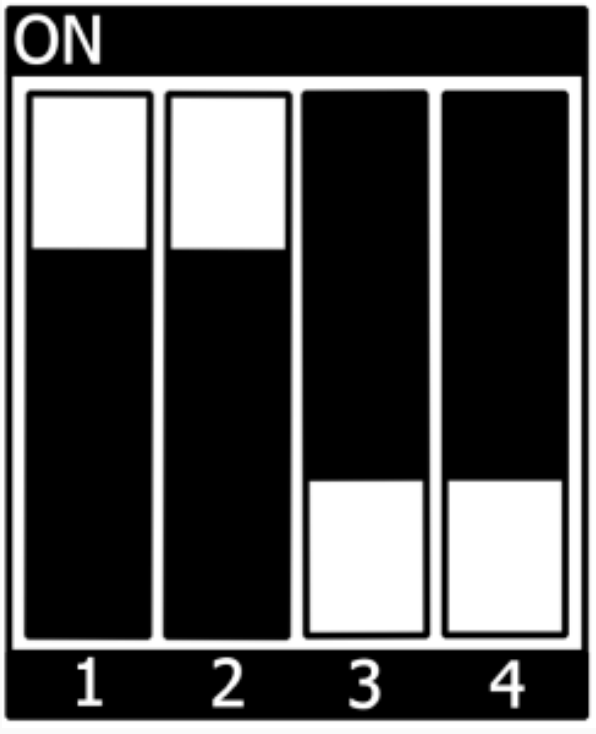



图 4: SD 卡

## 4.2 烧写 eMMC

要从 eMMC 启动，请确保 BSP 镜像已正确烧写到 eMMC，并且*bootmode switch (S3)* 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
```

可以运行以检查当前设置是否受到影响。如果 mmcblk0p1 和 mmcblk1p1 具有相同的 UUID，则该设置受到影响。

### 4.2.1 从 SD 卡烧写 eMMC

如果没有可用的网络，您可以通过 SD 卡更新 eMMC。为此，您只需在 SD 卡上准备一个可用的镜像文件（`*.wic`）。由于镜像文件相当大，您需要扩展 SD 卡以使用其全部空间（如果之前没有扩展的话）。有关如何扩展 SD 卡，请参见 “调整 ext4 根文件系统的大小”。

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 linux 环境中通过 SD 卡烧写 eMMC

您可以在 Linux 上烧写 eMMC。您只需在 SD 卡上保存一个 partup 包或 WIC 镜像即可。

- 在 SD 卡上查看您保存的 partup 包或 WIC 镜像或 WIC.XZ 镜像文件：

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

- 显示可用的 MMC 设备列表：

```
target:~$ ls /dev | grep mmc
mmcblk1
mmcblk1p1
mmcblk1p2
mmcblk0
mmcblk0boot0
mmcblk0boot1
mmcblk0p1
mmcblk0p2
mmcblk0rpmb
```

- eMMC 设备的特征是它包含两个 boot 分区：(mmcblk0 **boot0**; mmcblk0 **boot1**)
- Write the image to the phyCORE-i.MX 93 eMMC (/dev/mmcblk0 **without** partition) using partup:

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

**小技巧**

**强烈建议使用 partup，因为它更易于使用，并且可以充分利用 eMMC 设备的容量，自动调整分区大小。**

**备注**

或者，可以使用 dd 命令。

对于未压缩的 WIC 镜像（*.wic）：

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

对于压缩的 WIC 镜像（*.wic.xz）：

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

请注意，在使用 dd 进行烧写时，root 分区并没有使用全部存储容量。

- 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 (S3)* 配置为 eMMC。

### 4.2.2 从网络烧写 eMMC

i.MX 93 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。

**备注**

一些 PHYTEC 的 BSP 会生成压缩的 .wic.xz 镜像。在这种情况下，必须先对压缩镜像进行解压缩。

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

#### 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*

在主机上获取一个未压缩的镜像，并通过网络使用 ssh 将其发送到开发板的 eMMC，使用一行命令：

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

#### 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
phytec-qt6demo-image-phyboard-segin-imx93-2.rootfs.wic
```

通过网络使用 dd 命令结合 ssh 将镜像发送到您设备的 eMMC：

```
host:~$ dd if=phytec-qt6demo-image-phyboard-segin-imx93-2.rootfs.wic bs=1M status=progress | ssh root@192.
↪168.3.11 "dd of=/dev/mmcblk0 conv=fsync"
```

### 4.2.3 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 0
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。

### 4.2.4 从 USB 烧写 eMMC

#### 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 U 盘烧写 eMMC。您需要一个保存了完整镜像的 U 盘和一个可启动的 WIC 镜像（例如 phytec-qt6demo-image-phyboard-segin-imx93-2.rootfs.wic）。将*bootmode switch (S3)* 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
```

(续下页)

(接上页)

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

- 显示可用的 MMC 设备列表：

```
target:~$ ls /dev | grep mmc
mmcblk1
mmcblk1p1
mmcblk1p2
mmcblk0
mmcblk0boot0
mmcblk0boot1
mmcblk0p1
mmcblk0p2
mmcblk0rpmb
```

- eMMC 设备的特征是它包含两个 boot 分区：(mmcblk0 **boot0**; mmcblk0 **boot1**)
- Write the image to the phyCORE-i.MX 93 eMMC (/dev/mmcblk0 without partition):

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

- 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 (S3)* 配置为 **eMMC**。

## 4.3 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 更新与设备管理手册。

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。

**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`.

## 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-qt6demo-image-phyboard-segin-imx93-2.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-segin-imx93-2/获取它。要将 wic 镜像烧写到 eMMC，你还需要 phytec-qt6demo-image-phyboard-segin-imx93-2.wic。

### 5.3.3 开发板准备

将*bootmode switch (S3)* 设置为 **USB 串行下载**。同时，将 USB 端口*X8 (USB micro/OTG connector)* 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 UART1 console on PEB-EVAL-01 for **phyBOARD-Segin** and X-37 USB-C debug for **phyBOARD-Nash** 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 0 0 0 0
u-boot=> mmc partconf 0
EXT_CSD[179], PARTITION_CONFIG:
BOOT_ACK: 0x0
BOOT_PARTITION_ENABLE: 0x0
PARTITION_ACCESS: 0x0
```

或者在 linux 中检查 用从 eMMC boot 分区启动

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-qt6demo-image-phyboard-segin-imx93-2.wic
```

## 5.4 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.

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

- 我们的 i.MX 93 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-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-qt6demo-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-qt6demo-image-cortexa55-toolchain-5.0.x.sh
host:~$ ./phytec-ampliphy-vendor-glibc-x86_64-phytec-qt6demo-image-cortexa55-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.
```

### 5.4.4 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/5.0.x/environment-setup-cortexa55-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
```

**警告**

在较旧的主机发行版（例如，Ubuntu 20.04 LTS）上使用 Scarthgap NXP 基础的 BSP 时，构建用于主机的 U-Boot 或 Linux 内核工具可能会出现问题。如果遇到"undefined reference" 错误，一种解决方法是在 PATH 环境变量中添加主机的 binutils。

```
host$ export PATH=/usr/bin:$PATH
```

在加载 SDK *environment-setup* 后，运行此命令。

请注意，在较新版本的发行版（如 Ubuntu 22.04）中未观察到 SDK 相关问题，这些发行版似乎无需任何修改即可正常工作。

## 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
- 此版本的 **tag** 称为 v2024.04-2.2.0-phy9
- 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/v2024.04-2.2.0-phy9
```

- 从技术上讲，您现在可以编译 U-Boot，但实际上还需要一些进一步的步骤：

– 创建您自己的开发分支：

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

**备注**

您可以根据您的需要自行命名您的开发分支，这只是一个示例。

- 设置编译环境：

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

### 5.5.2 获取所需的二进制文件

要编译 imx-boot，您需要 **收集**这些 **文件**以便后续使用 **imx-mkimage 工具**：

- **ARM Trusted firmware 二进制文件**（*mkimage* 工具兼容格式 **bl31.bin** ）：bl31-imx93.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
- **容器镜像**：mx93a1-ahab-container.img

如果您已经使用 Yocto 编译了我们的 BSP，您可以从此处提到的目录中获取这些文件：*BSP Images*

或者您可以从 PHYTEC 下载服务器（https://download.phytec.de/Software/Linux/BSP-Yocto-i.MX93/BSP-Yocto-NXP-i.MX93-PD24.2.1/images/ampliphy-vendor/phyboard-segin-imx93-2/imx-boot-tools/ ）下载文件。您可以使用下面的命令从该服务器下载所有文件：

```
host:~$ mkdir ./artefacts && cd ./artefacts
host:~/artefacts$ wget \
                  https://download.phytec.de/Software/Linux/BSP-Yocto-i.MX93/BSP-Yocto-NXP-i.MX93-PD24.2.
↪1/images/ampliphy-vendor/phyboard-segin-imx93-2/imx-boot-tools//bl31-imx93.bin \
                  https://download.phytec.de/Software/Linux/BSP-Yocto-i.MX93/BSP-Yocto-NXP-i.MX93-PD24.2.
↪1/images/ampliphy-vendor/phyboard-segin-imx93-2/imx-boot-tools//tee.bin \
                  https://download.phytec.de/Software/Linux/BSP-Yocto-i.MX93/BSP-Yocto-NXP-i.MX93-PD24.2.
↪1/images/ampliphy-vendor/phyboard-segin-imx93-2/imx-boot-tools//lpddr4_dmem_1d_v202201.bin \
                  https://download.phytec.de/Software/Linux/BSP-Yocto-i.MX93/BSP-Yocto-NXP-i.MX93-PD24.2.
↪1/images/ampliphy-vendor/phyboard-segin-imx93-2/imx-boot-tools//lpddr4_dmem_2d_v202201.bin \
                  https://download.phytec.de/Software/Linux/BSP-Yocto-i.MX93/BSP-Yocto-NXP-i.MX93-PD24.2.
↪1/images/ampliphy-vendor/phyboard-segin-imx93-2/imx-boot-tools//lpddr4_imem_1d_v202201.bin \
                  https://download.phytec.de/Software/Linux/BSP-Yocto-i.MX93/BSP-Yocto-NXP-i.MX93-PD24.2.
↪1/images/ampliphy-vendor/phyboard-segin-imx93-2/imx-boot-tools//lpddr4_imem_2d_v202201.bin \
                  https://download.phytec.de/Software/Linux/BSP-Yocto-i.MX93/BSP-Yocto-NXP-i.MX93-PD24.2.
↪1/images/ampliphy-vendor/phyboard-segin-imx93-2/imx-boot-tools//mx93a1-ahab-container.img
host:~/artefacts$ cd ..
```

### 5.5.3 编译 bootloader

- 编译 u-boot：

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

**备注**

In command above replace `<defconfig>` with `imx93-phycore_defconfig`

#### 使用 imx-mkimage 编译最终的 flash.bin

- 获取 imx-mkimage:

```
host:~$ git clone https://github.com/nxp-imx/imx-mkimage
host:~$ cd imx-mkimage/
host:~/imx-mkimage$ git checkout tags/lf-6.6.52-2.2.0
```

- 将固件二进制文件复制到 imx-mkimage

```
host:~/imx-mkimage$ cp ../artefacts/bl31-imx93.bin ./iMX9/bl31.bin
host:~/imx-mkimage$ cp \
                    ../artefacts/lpddr4_* \
                    ../artefacts/mx93a1-ahab-container.img \
                    ../artefacts/tee.bin \
                    ./iMX9/
```

- 将 u-boot 二进制文件和 DTB 复制到 imx-mkimage 中

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

**备注**

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

- 编译最终的 flash.bin 二进制文件

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

### 5.5.4 将 bootloader 烧写到块设备上

flash.bin 可以在 imx-mkimage/iMX9/目录下找到，现在可以进行烧写。需要一个特定于芯片的偏移量:

| SoC | User 分区偏移量 | Boot 分区偏移量 | eMMC 设备 |
|---|---|---|---|
| i.MX 93 | 32 kiB | 0 kiB | /dev/mmcblk0 |

例如，烧写 SD 卡:

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

**备注**

在上述命令中，将 `<sd-card>` 替换为您的 SD 卡设备名称。有关如何找到设备名称的更多信息，请参阅入门部分中的 找到正确的设备名。

**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-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-phy9
- Check out 所需的 linux-phytec-imx 标签：

```
host:~$ git clone git://github.com/phytec/linux-phytec-imx.git
host:~$ cd ~/linux-phytec-imx/
host:~/linux-phytec-imx$ git fetch --all --tags
host:~/linux-phytec-imx$ git checkout tags/v6.6.52-2.2.0-phy9
```

- 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-phytec-linux
```

### 5.6.2 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 找到
- dtb 文件可以在 ~/linux-phytec-imx/arch/arm64/boot/dts/freescale/imx93-phyboard-segin.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
```

**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-phytec-imx$ cp arch/arm64/boot/Image /path/to/sdcard/boot/
host:~/linux-phytec-imx$ cp arch/arm64/boot/dts/freescale/imx93-phyboard-segin.dtb /path/to/sdcard/boot/
↪oftree
host:~/linux-phytec-imx$ make INSTALL_MOD_PATH=/path/to/sdcard/root/ modules_install
```

## 5.7 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。

### 5.7.1 Gparted

- 获取 GParted:

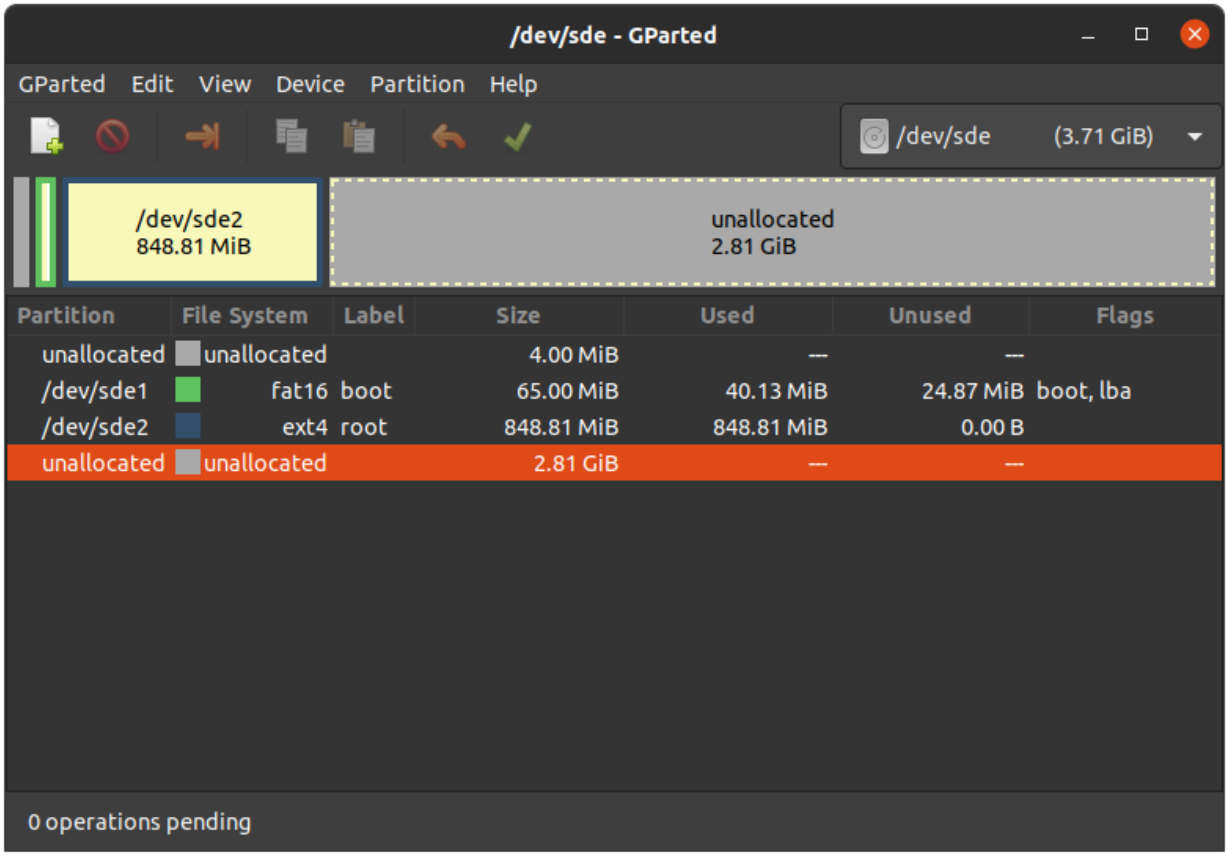
  ```
  host:~$ sudo apt install gparted
  ```

- 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
  ```

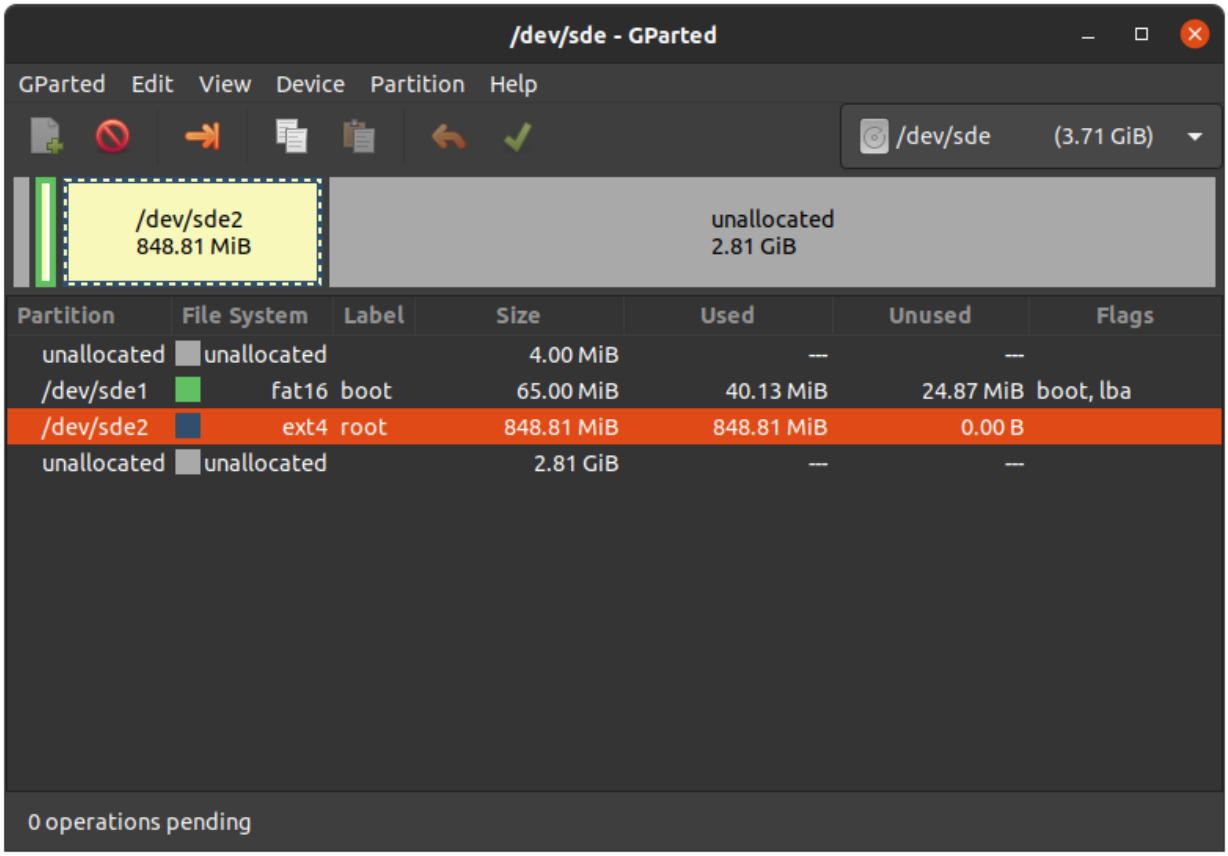


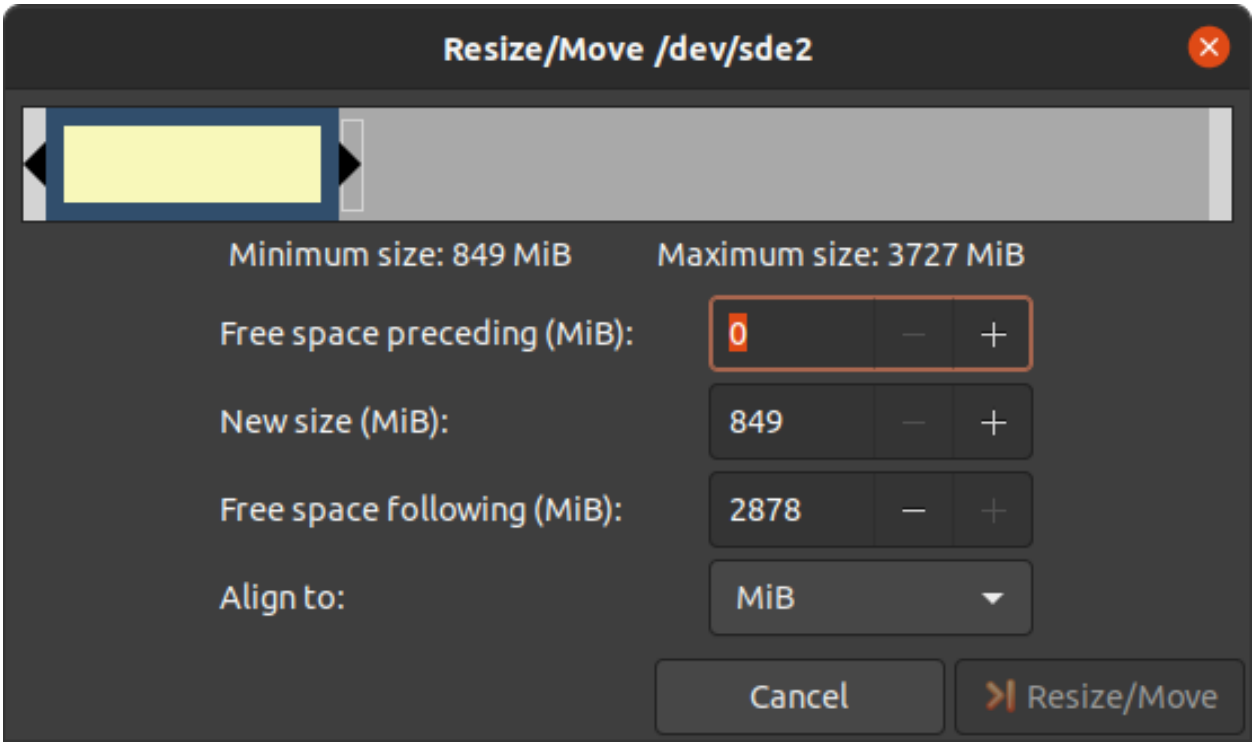

### 扩展根文件系统

**警告**

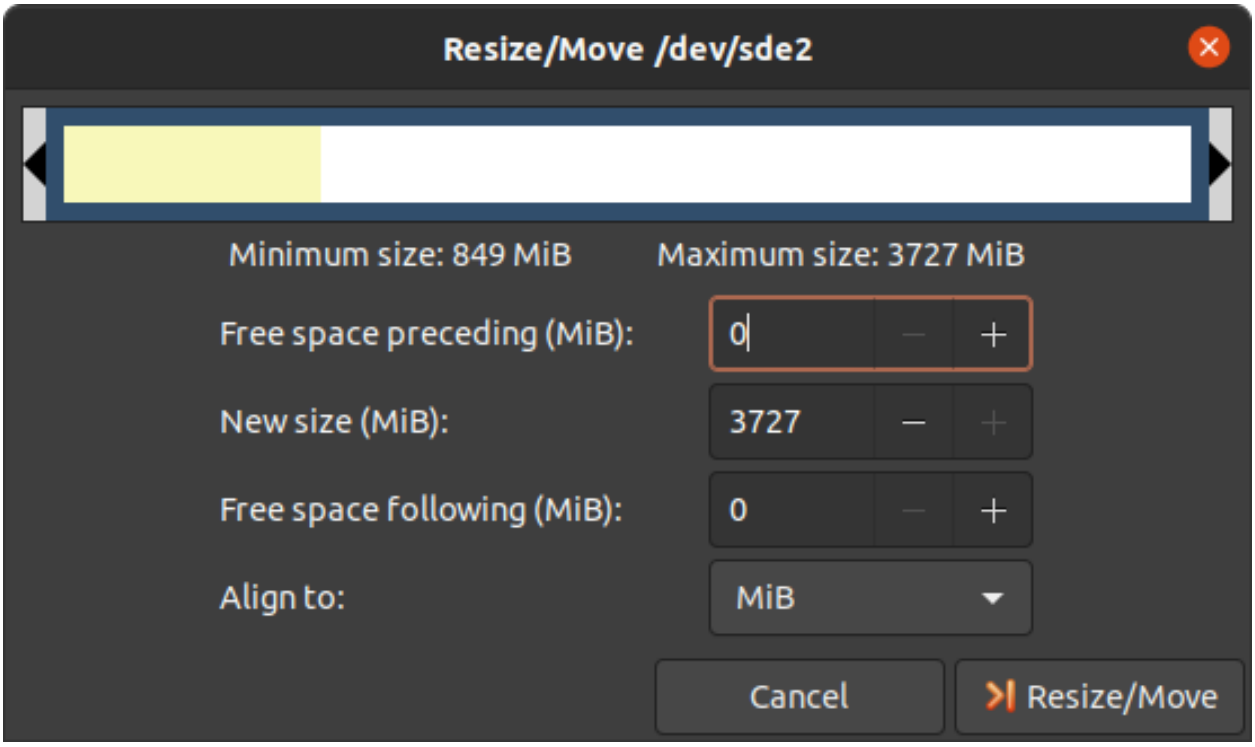
使用 resize2fs 版本 1.46.6 及更早版本的 PC 系统（例如 Ubuntu 22.04）无法烧写在 Mickledore 以及更新的 yocto 版本上创建的 partup 软件包。这个是因为 resize2fs 新增了默认选项而导致的兼容性问题。有关详细信息，请参阅 发布说明 。

- 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:

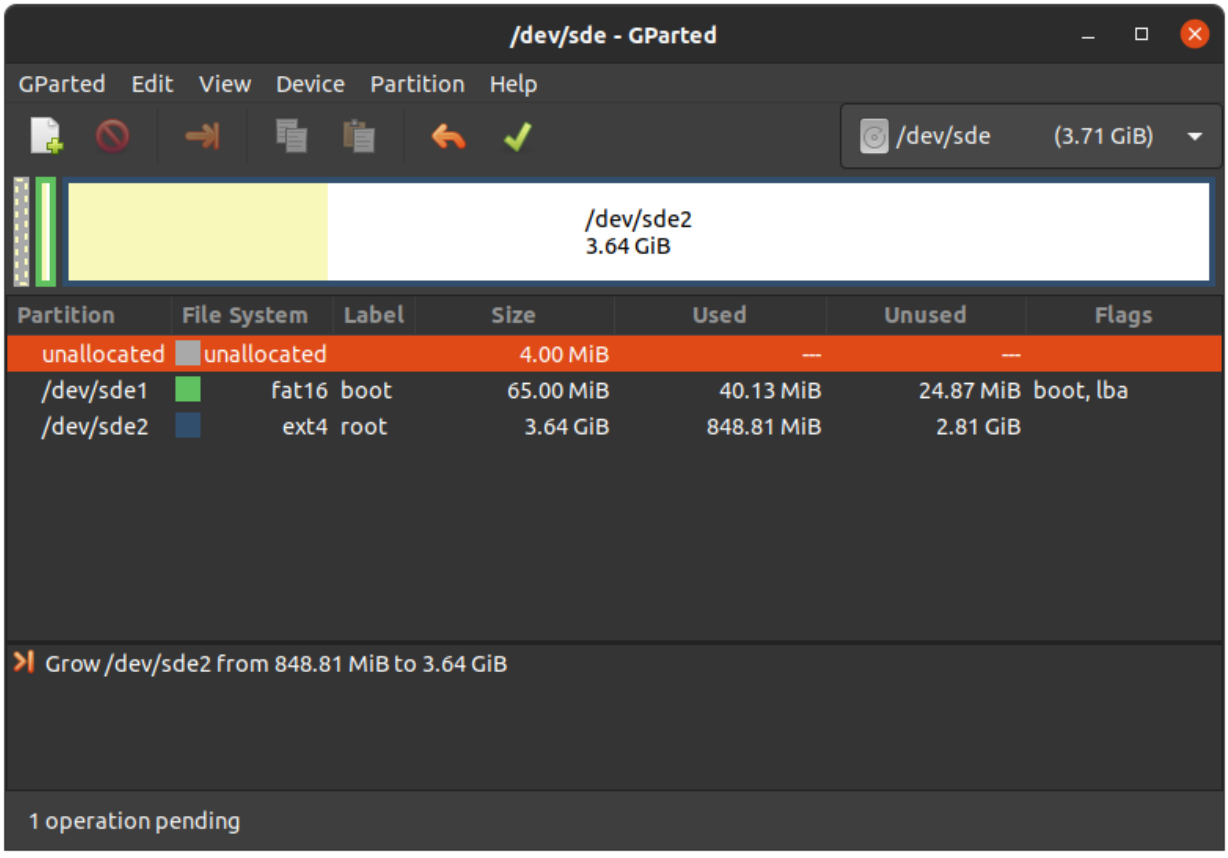



- 您可以根据需要拖动滑块或手动输入大小。




- 通过点击“Change Size”按钮确认您的输入。



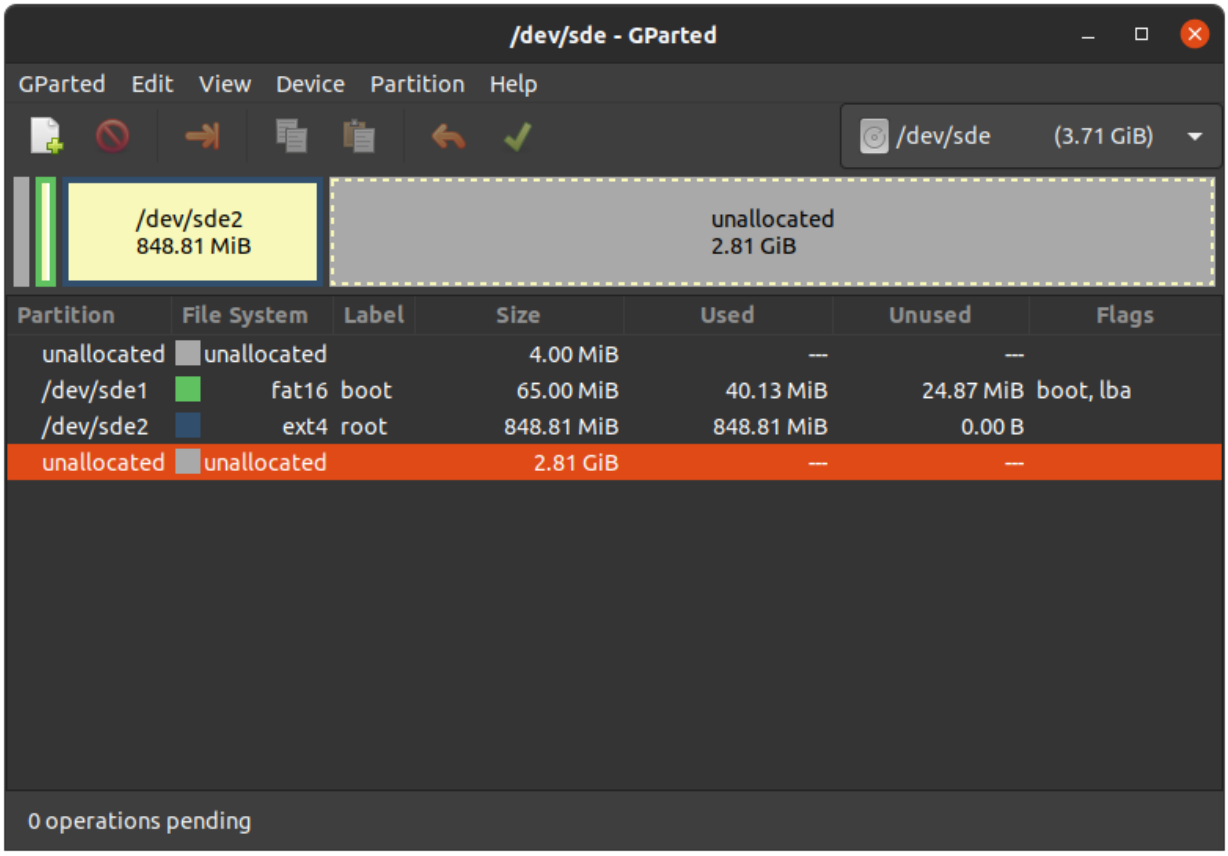

- 要应用您的更改，请按绿色勾号。

- 现在您可以挂载根分区并将 phytec-qt6demo-image-phyboard-segin-imx93-2.wic 镜像复制到其中。然后再卸载它：

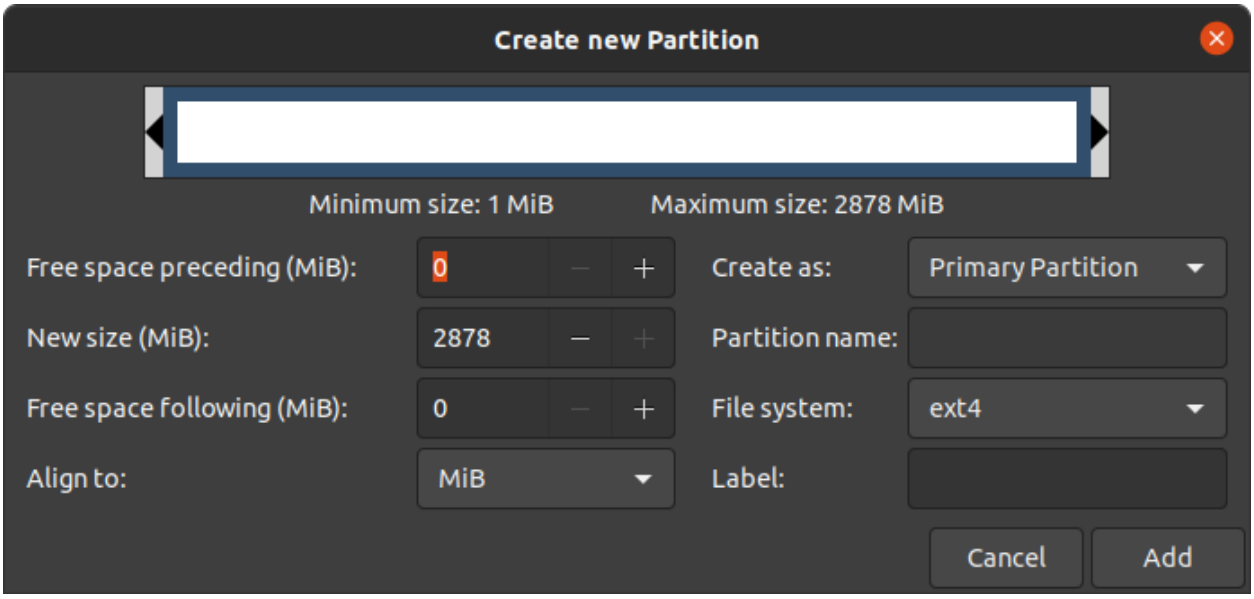
```
host:~$ sudo cp phytec-qt6demo-image-phyboard-segin-imx93-2.wic /mnt/ ; sync
host:~$ umount /mnt
```

### 创建第三个分区

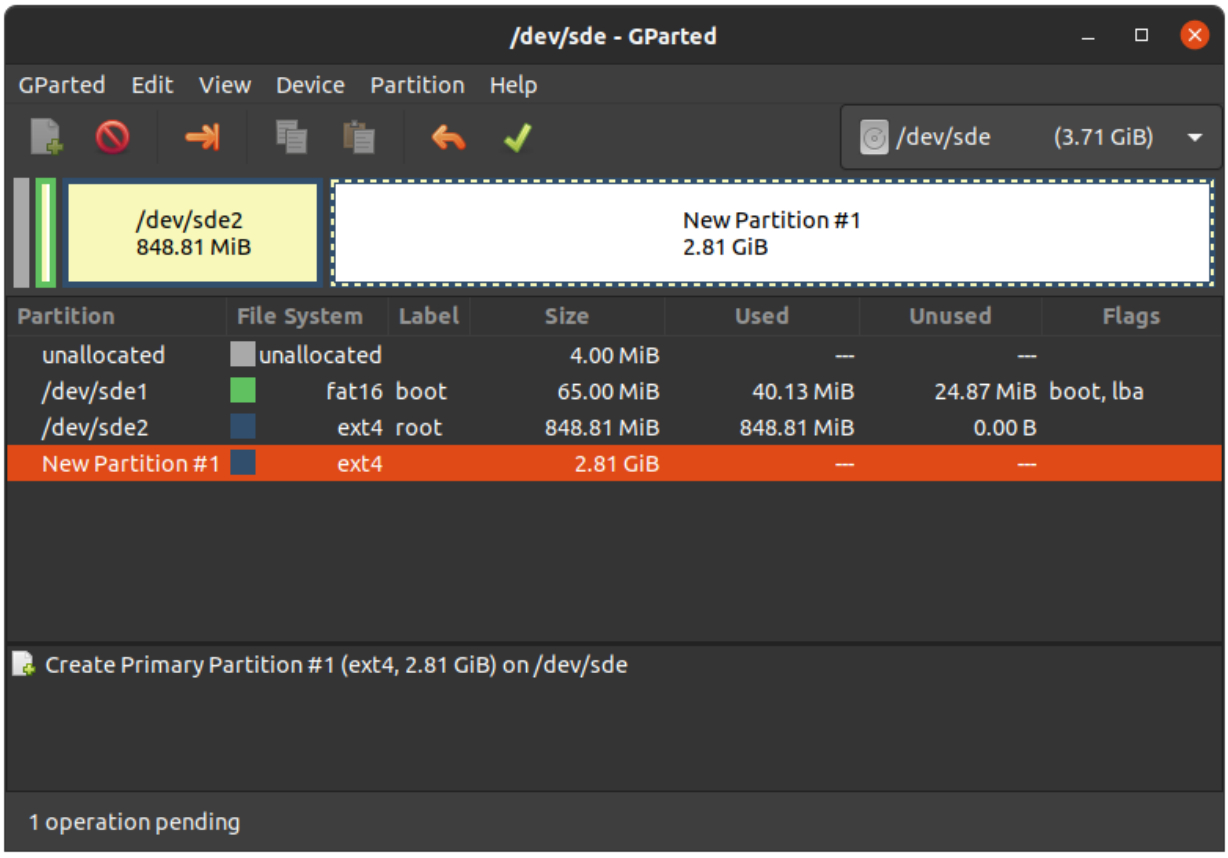
- Choose your SD card device at the drop-down menu on the top right



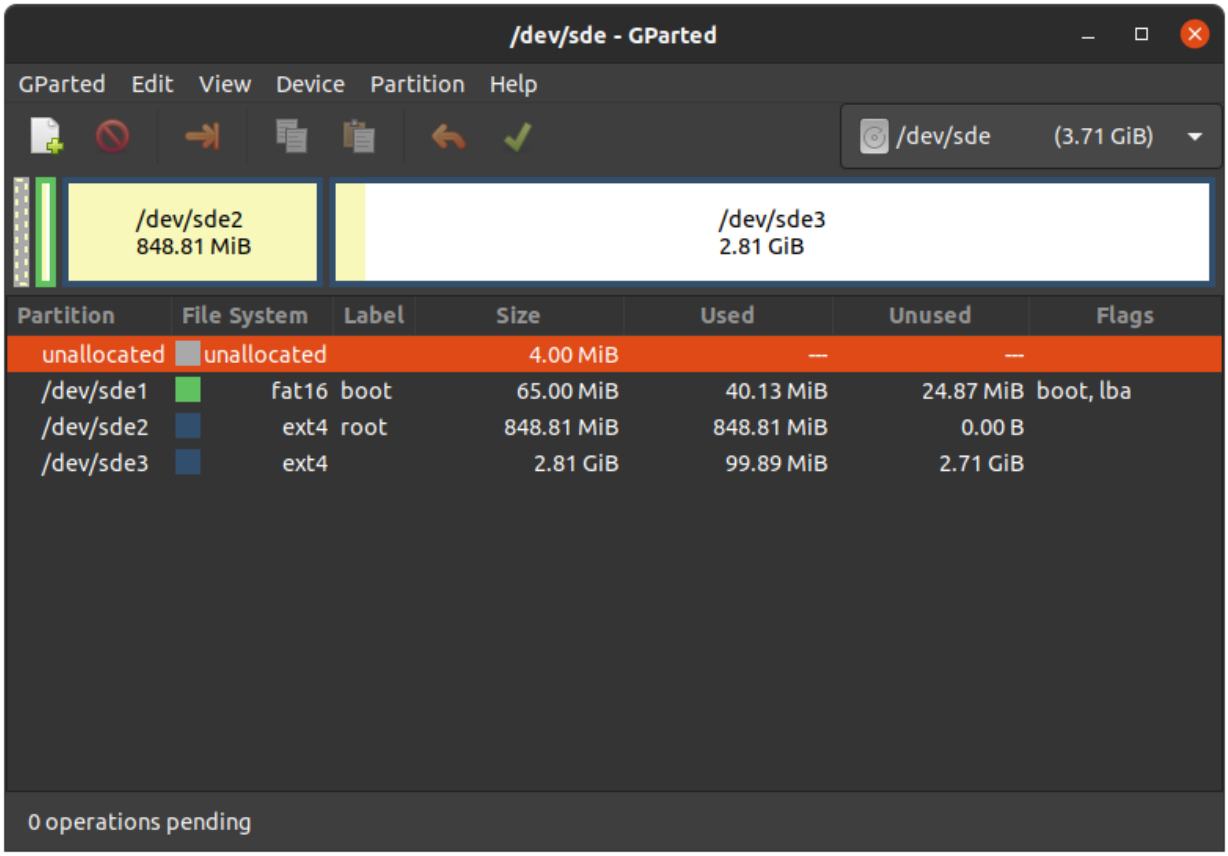

- 选择更大的未分配区域，然后点击”New”：




- 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-phyboard-segin-imx93-2.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-phyboard-segin-imx93-2.wic /mnt/ ; sync
host:~$ umount /mnt
```

## 5.8 Enabling JTAG Debug Interface on phyBOARD Nash

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

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

**备注**

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

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 93 BSP 设备树概念

以下部分说明了 PHYTEC 配置基于 i.MX 93 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 93 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。

phyboard-segin-imx93-2.conf 可用的 overlay 文件有：

```
imx93-phyboard-segin-peb-av-02.dtbo
imx93-phyboard-segin-peb-eval-01.dtbo
imx93-phyboard-segin-peb-wlbt-05.dtbo
imx93-phycore-npu.dtbo
imx93-phycore-rpmsg.dtbo
imx91-imx93-phycore-no-emmc.dtbo
imx91-imx93-phycore-no-eth.dtbo
```

phyboard-nash-imx93-1.conf 可用的 overlay 有：

```
imx93-phyboard-nash-jtag.dtbo
imx93-phyboard-nash-peb-av-10.dtbo
imx93-phyboard-nash-peb-wlbt-07.dtbo
imx93-phyboard-nash-pwm-fan.dtbo
imx93-phyboard-nash-vm016.dtbo
imx93-phycore-npu.dtbo
imx93-phycore-rpmsg.dtbo
imx91-imx93-phycore-no-emmc.dtbo
imx91-imx93-phycore-no-eth.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=imx93-phyboard-segin-peb-eval-01.dtbo imx93-phyboard-segin-peb-av-02.dtbo
```

**备注**

确保 boot 分区已挂载！如果没有，您可以使用以下命令进行挂载：

```
target:~$ mount /dev/mmcblkXp0 /boot
```

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 imx93-phyboard-segin-peb-av-02.dtbo
u-boot=> printenv overlays
overlays=imx93-phyboard-segin-peb-av-02.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 imx93-phyboard-segin-peb-av-02.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 `${overlays}` 变量来描述在运行时无法检测到的扩展板。为了禁止在启动时应用列在 `${overlays}` 变量中的 overlay，可以在 bootloader 环境中将 `${no_overlays}` 变量设置为 *1*。

```
u-boot=> setenv no_overlays 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 93 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 93 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 93 引脚复用

该 i.MX 93 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 93 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 93 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 imx93-phyboard-segin.dts 中 UART1 设备的引脚复用示例：

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

字符串的第一部分 MX93_PAD_UART1_RXD__LPUART1_RX 指定了引脚（在这个例子中是 UART1_RXD）。字符串的第二部分（LPUART1_RX）是该引脚的期望复用选项。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。在当前情况下，内部上拉电阻被激活。

The device tree representation for UART1 pinmuxing: https://github.com/phytec/linux-phytec-imx/blob/v6.6.52-2.2.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-segin.dts#L263

## 7.2 Ethernet

phyBOARD-Segin/Nash i.MX 93 提供两个以太网接口。

- 在 phyBOARD-Segin i.MX 93 上，我们有：
  - 由 phyCORE-i.MX 93 提供的百兆以太网
  - 由 phyBOARD 提供的百兆以太网。
- 在 phyBOARD-Nash i.MX 93 上，我们有：
  - 由 phyCORE-i.MX 93 提供的百兆以太网
  - 由 phyBOARD 提供的千兆以太网。

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/blob/v6.6.52-2.2.0-phy9/arch/arm64/boot/dts/freescale/imx93-phycore-som.dtsi#L61.

The device tree set up for EQOS Ethernet IP core where the PHY is populated on the phyBOARD-Segin/Nash i.MX 93 can be found here: https://github.com/phytec/linux-phytec-imx/blob/v6.6.52-2.2.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-segin.dts#L118 or here: https://github.com/phytec/linux-phytec-imx/blob/v6.6.52-2.2.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-nash.dts#L87.

### 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
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
```

(续下页)

(接上页)

```
    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 WLAN/Bluetooth

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

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

**小技巧**

With the BSP Version PD24.2.0 and newer, the PEB-WLBT-05 overlay needs to be activated first, otherwise the PEB-WLBT-05 won't be recognized.

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

之后，bootenv.txt 文件应该如下所示（它还可以包含其他设备树 overlay!）:

```
overlays=imx93-phyboard-segin-peb-wlbt-05.dtbo
```

更改将在重启后应用:

```
target:~$ reboot
```

有关设备树 overlay 的更多信息，请阅读*device tree* 章节。

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

**备注**

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

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

**小技巧**

With the BSP Version PD24.2.1 and newer, the PEB-WLBT-07 overlay needs to be activated first, otherwise the PEB-WLBT-07 won't be recognized.

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

之后，bootenv.txt 文件应该如下所示（它还可以包含其他设备树 overlay！）：

```
overlays=imx93-phyboard-nash-peb-wlbt-07.dtbo
```

更改将在重启后应用：

```
target:~$ reboot
```

有关设备树 overlay 的更多信息，请阅读*device tree* 章节。

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

**备注**

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

### 7.3.3 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：

```
Successfully initialized wpa_supplicant
```

ip 地址自动通过 DHCP 配置。有关其他可能的 IP 配置，请参阅 Yocto Reference Manual (scarthgap) 中的“更改网络配置”部分。

### 7.3.4 蓝牙

Bluetooth is connected to UART5 interface. 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>
```

### 7.3.5 可见性

要激活可见性：

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

要禁用可见性：

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

### 7.3.6 连接

```
target:~$ bluetoothctl
[bluetooth]# discoverable on
Changing discoverable on succeeded
[bluetooth]# pairable on
```

(续下页)

(接上页)

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

## 7.4 SD card

The i.MX 93 supports a slot for Secure Digital cards to be used as general-purpose block devices. These devices can be used in the same way as any other block device.

**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/blob/v6.6.52-2.2.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-segin.dts#L217 or here: https://github.com/phytec/linux-phytec-imx/blob/v6.6.52-2.2.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-nash.dts#L206

DT configuration for the eMMC interface can be found here: https://github.com/phytec/linux-phytec-imx/blob/v6.6.52-2.2.0-phy9/arch/arm64/boot/dts/freescale/imx93-phycore-som.dtsi#L194 or here:

## 7.5 e.MMC Devices

PHYTEC modules like phyCORE-i.MX 93 are populated with an e.MMC memory chip as the main storage. e.MMC devices contain raw Multi-Level Cells (MLC) or Triple-Level Cells (TLC) combined with a memory controller that handles ECC and wear leveling. They are connected via an SD/MMC interface to the i.MX 93 and are represented as block devices in the Linux kernel like SD cards, flash drives, or hard disks.

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

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

### 7.5.1 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]
 [...]
```

### 7.5.2 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。

### 7.5.3 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 93 SoMs. To check this on the running target:

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

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

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.

### 7.5.5 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:;
```

### 7.5.6 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！

### 7.5.7 e.MMC Boot Partitions

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

The user partition is called the User Data Area in the JEDEC standard and is the main storage partition. The partitions boot1 and boot2 can be used to host the bootloader and are more reliable. Which partition

the i.MX 93 uses to load the bootloader is controlled by the boot configuration of the e.MMC device. The partition rpmb is a small partition and can only be accessed via a trusted mechanism.

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 93 SoC 的偏移量：

| SoC | User 分区偏移量 | Boot 分区偏移量 | e.MMC Device |
|---|---|---|---|
| i.MX 93 | 32 kiB | 0 kiB | /dev/mmcblk0 |

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

(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
```

## 7.6 GPIOs

phyBOARD-Segin/Nash i.MX 93 没有用户可以使用的 GPIO，因为所有 GPIO 都被内核设备驱动程序使用或用于其他特定目的。处理器将其 GPIO 组织为五个 32 个 GPIO 的组（GPIO1 –GPIO4）。gpiochip0、gpiochip32、gpiochip64 和 gpiochip96 是这些内部 i.MX 93 GPIO 组 GPIO1 –GPIO4 的 sysfs 表示。

GPIO 被标识为 GPIO<X>_<Y>（例如，GPIO4_07）。<X> 标识 GPIO Bank，并从 1 到 4 计数，而 <Y> 表示 Bank 内的 GPIO。<Y> 从 0 到 31 计数（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 [43810000.gpio] (32 lines)
  gpiochip1 [43820000.gpio] (32 lines)
  gpiochip2 [43830000.gpio] (32 lines)
  gpiochip3 [47400000.gpio] (32 lines)
  ```

**备注**

i.MX 93 Application Processor Reference Manual 中的 GPIO 芯片顺序与 Linux 内核中的顺序不同！

| GPIOchip 地址 | Linux | Reference Manual |
|---|---|---|
| 0x43810000 | gpiochip0 | gpiochip2 |
| 0x43820000 | gpiochip1 | gpiochip3 |
| 0x43830000 | gpiochip2 | gpiochip4 |
| 0x47400000 | gpiochip3 | gpiochip1 |

- 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 chip0 读取 GPIO 3）：

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

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

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

**备注**

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

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

This is the expected behavior in libgpiod version 2.

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

```
target:~$ gpioset -t0 -c gpiochip0 3=0
```

- 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
-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.
```

(续下页)

(接上页)

```
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。

### 7.6.1 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。要使 CONFIG_GPIO_SYSFS 在 menuconfig 中可见，必须先启用 CONFIG_EXPERT 选项。

You can also add this option for example to the imx9_phytec_defconfig config in the linux kernel sources under arch/arm64/configs

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

## 7.7 ADC

PHYTEC i.MX 93 包含通用的 ADC，可用于与模拟传感器连接。

通过 sysfs 可以读取 ADC 值：

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

在 phyBOARD-Nash i.MX 93 上，ADC 线路可以通过 X16 扩展连接器访问：

| ADC 输入 | X16 引脚 |
|---|---|
| ADC_IN0 | 47 |
| ADC_IN2 | 49 |

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

这里的 LED 灯 `green:heartbeat` 位于 phyCORE-i.MX 93 上。如果您使用的是 phyBOARD-Segin，PEB-EVAL-01 上还有一个 黄色 LED 灯。

- 打开 LED 灯：

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

- 关闭 LED：

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

Device tree configuration for the User I/O configuration can be found here: https://github.com/phytec/linux-phytec-imx/blob/v6.6.52-2.2.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-segin-peb-eval-01.dtso#L33

## 7.9 I²C 总线

该 i.MX 93 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 93 的 I²C 模块。本节描述了我们 phyBOARD-Segin/Nash i.MX 93 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 I²C3 bus configuration (e.g. imx93-phycore-som.dtsi): https://github.com/phytec/linux-phytec-imx/blob/v6.6.52-2.2.0-phy9/arch/arm64/boot/dts/freescale/imx93-phycore-som.dtsi#L88

General I²C2 bus configuration for imx93-phyboard-segin.dts: https://github.com/phytec/linux-phytec-imx/blob/v6.6.52-2.2.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-segin.dts#L159 or for imx93-phyboard-nash.dts: https://github.com/phytec/linux-phytec-imx/blob/v6.6.52-2.2.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-nash.dts#L117

## 7.10 EEPROM

在 phyCORE-i.MX 93 SoM 和 phyBOARD-Segin/Nash i.MX 93 上有两个不同的 I2C EEPROM 存储。目前只有 phyCORE-i.MX 93 上的存储被启用，它用于硬件检测。

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

phyCORE-i.MX 93 SoM 上的 I2C EEPROM 的空间被划分两部分。

- 正常区域（大小：4096 字节，总线：I2C-2，地址：0x50）

- ID 页面（大小：32 字节，总线：I2C-2，地址：0x58）

可以从设备进行读写操作：

```
target:~$ hexdump -C /sys/class/i2c-dev/i2c-2/device/2-0058/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-2/device/2-0050/eeprom bs=1 count=1024  | od -x
```

要将整个 EEPROM（ID 页）填充为零，我们首先需要禁用 EEPROM 写保护，请使用：

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

然后可以写入 EEPROM：

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

To re-enable EEPROM wire protection, use:

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

**警告**

正常 EEPROM 区域的前 256 个字节（总线：I2C-2 地址：0x50）是保留的，不应被覆盖！（见下文）

### 7.10.2 EEPROM SoM 检测

PHYTEC 在 EEPROM 正常区域的前 256 字节中存储有关核心板的信息。这包括 PCB 版本和贴装选项。

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。

如果正常区域的前 256 个字节被删除，启动加载程序将回退到 phyCORE-i.MX 93 开发板内存配置，即 1GiB RAM。

**警告**

正常 EEPROM 区域的前 256 个字节（总线：I2C-2 地址：0x50）不应被擦除或损坏！这可能会影响 bootloader 的行为。板子可能无法正确启动。

### 7.10.3 恢复 EEPROM 数据

硬件自检数据已预先写入 EEPROM 数据空间。如果您不小心删除或覆盖了数据，请联系我们的支持团队！

DT representation, e.g. in phyCORE-i.MX 93 file can be found in our PHYTEC git: https://github.com/phytec/linux-phytec-imx/blob/v6.6.52-2.2.0-phy9/arch/arm64/boot/dts/freescale/imx93-phycore-som.dtsi#L172

## 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 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/blob/v6.6.52-2.2.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-segin.dts#L177 or https://github.com/phytec/

linux-phytec-imx/blob/v6.6.52-2.2.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-nash.dts#L126

## 7.12 USB 主控制器

i.MX 93 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，传输速度高达 480 Mbps（高速'HS'）。USB 子系统具有两个独立的 USB 控制器。两个控制器都能够充当 USB Device 或 USB Host，但在 phyBOARD-Segin/Nash i.MX 93 上，其中一个被用作仅 Host 端口（USB-A 连接器）。

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。

OTG 端口提供了一个额外的引脚用于过流保护，但在 phyBOARD-Segin/Nash i.MX 93 上并未使用。由于未使用，该引脚在设备树中也被禁用。如果需要使用该引脚，请在设备树中激活过流保护，并根据设备规格设置正确的极性（高电平/低电平）。有关正确的设置，请参考 Linux 内核文档中的 Linux/Documentation/devicetree/bindings/usb/ci-hdrc-usb2.txt。

DT representation for USB Host: https://github.com/phytec/linux-phytec-imx/blob/v6.6.52-2.2.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-segin.dts#L196 or https://github.com/phytec/linux-phytec-imx/blob/v6.6.52-2.2.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-nash.dts#L185

## 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

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

**例子**:

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

**提示**

为了节省时间，请复制这些命令并在脚本中执行它们

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

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

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

**备注**

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

现在你可以创建你想要使用的功能：

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

- *acm*: 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。

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

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

**提示**

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

将定义的功能绑定到配置：

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

最后，使用以下命令启动 USB gadget：

```
target:~$ cd /sys/kernel/config/usb_gadget/g1
target:~$ ls --indicator-style=none /sys/class/udc/
ci_hdrc.0
target:~$ echo "ci_hdrc.0" > UDC
```

`ci_hdrc.0` is an example, replace it with the actual name. Any trailing `@` might be shown by `ls` to show it is a link, remove it. If your system has more than one USB Device or OTG port, you can pass the right one to the USB Device Controller (UDC).

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
```

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

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

**警告**

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

## 7.14 RS232/RS485

phyBOARD-Nash i.MX 93 i.MX 93 SoC 提供一个 RS232/RS485 串口。

**警告**

由于 phyBOARD-Nash i.MX 93 PCB 版本 1616.0 上的硬件缺陷，RS232 的硬件流控制和 RS485 无法正常工作

### 7.14.1 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/ttyLP6 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/ttyLP6
  ```

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

主机应打印出”123”。

### 7.14.2 RS485

**提示**

在使用较长电缆时，请记得在总线两端各使用 120 欧姆的终端电阻。

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/ttyLP6 -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

linux-serial-test 会自动设置 ioctl，也可以通过 rs485conf 手动设置。

你可以用以下命令显示当前配置：

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

您可以通过以下方式列出帮助选项：

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

The device tree representation for RS232 and RS485: https://github.com/phytec/linux-phytec-imx/blob/v6.6.52-2.2.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-nash.dts#L178

## 7.15 CAN FD

phyBOARD-Segin/Nash i.MX 93 具有一个支持 CAN FD 的 flexCAN 接口。它们由 Linux 标准 CAN 框架支持，该框架建立在 Linux 网络层之上。使用该框架，CAN 接口表现得像普通的 Linux 网络设备，并具有一些特定于 CAN 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

- 使用：

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

  查看接口的状态。CAN 接口应该显示为 can0。

- 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
  4: can0: <NOARP,UP,LOWER_UP,ECHO> mtu 72 qdisc pfifo_fast state UP mode DEFAULT group default qlen 10
     link/can  promiscuity 0 allmulti 0 minmtu 0 maxmtu 0
     can <FD> state ERROR-ACTIVE (berr-counter tx 0 rx 0) restart-ms 0
           bitrate 500000 sample-point 0.875
           tq 25 prop-seg 37 phase-seg1 32 phase-seg2 10 sjw 5 brp 1
           flexcan: tseg1 2..96 tseg2 2..32 sjw 1..16 brp 1..1024 brp_inc 1
           dbitrate 2000000 dsample-point 0.750
           dtq 25 dprop-seg 7 dphase-seg1 7 dphase-seg2 5 dsjw 2 dbrp 1
           flexcan: dtseg1 2..39 dtseg2 2..8 dsjw 1..4 dbrp 1..1024 dbrp_inc 1
           clock 40000000
           re-started bus-errors arbit-lost error-warn error-pass bus-off
           0          0          0          0          0          0         numtxqueues 1 numrxqueues 1␣
  ↪gso_max_size 65536 gso_max_segs 65535 tso_max_size 65536 tso_max_segs 65535 gro_max_size 65536
  gso_ipv4_max_size 65536 gro_ipv4_max_size 65536 parentbus platform parentdev 443a0000.can
     RX:  bytes packets errors dropped  missed   mcast
               0       0      0       0       0       0
     TX:  bytes packets errors dropped carrier collsns
               0       0      0       0       0       0
  ```

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 |

The CAN configuration is done in the systemd configuration file `/lib/systemd/network/11-can.network`. For a persistent change of (as an example, the default bitrates), change the configuration in the BSP under `./meta-ampliphy/recipes-core/systemd/systemd-conf/11-can.network` in the root filesystem and rebuild the root filesystem.

```
[Match]
Name=can*

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

**备注**

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

To disable flexible datarate manually, one can use:

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

The bitrate can also be changed manually, for example:

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

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。

Device Tree CAN configuration of imx93-phyboard-segin.dts: https://github.com/phytec/linux-phytec-imx/blob/v6.6.52-2.2.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-segin.dts#

L151 or https://github.com/phytec/linux-phytec-imx/blob/v6.6.52-2.2.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-nash.dts#L109

## 7.16 phyBOARD-Segin i.MX 93 上的音频

在 phyBOARD-Segin i.MX 93 上使用了 TI TLV320AIC3007 音频编解码器（CODEC）。它使用 I2S 进行数据传输，使用 I2C 进行控制。可用的音频信号有：

- 立体声 LINE IN，
- 立体声 LINE OUT，
- Class D 1W 的扬声器输出

要检查您的声卡驱动程序是否正确加载以及设备名称，请输入以下命令以查看播放设备：

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

或输入录音设备：

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

### 7.16.1 Alsamixer

要检查声卡的功能，请输入：

```
target:~$ alsamixer
```

您应该会看到很多选项，因为音频 IC 具有许多可以测试的功能。通过 SSH 打开 alsamixer 的图形界面比通过调试串口打开更易于使用。所有混音点都有单声道或立体声增益控制。"MM" 表示该功能被静音（左右输出均为静音），可以通过按' **m** ' 切换。您还可以通过按' < ' 左和' > ' 切换右声道输出。使用 **tab** 键，您可以在播放和录音控制之间切换。

### 7.16.2 恢复默认音量

一些默认设置存储在 `/var/lib/alsa/asound.state` 中。您可以使用以下命令保存当前的 alsa 设置：

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

您可以通过以下方式恢复已保存的 alsa 设置：

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

### 7.16.3 播放

运行 speaker-test 以检查播放功能：

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

要播放简单的音频流，您可以使用 aplay。例如，要播放 ALSA 测试音频：

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

要播放其他格式，例如 mp3，您可以使用 Gstreamer：

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

如果扬声器音量太低，可以增加音量（值范围 0-3）：

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

**提示**

扬声器输出仅为单声道，因此当播放立体声轨道时，扬声器只会播放左声道。

### 7.16.4 录音

`arecord` 是一个命令行工具，用于捕获音频流，默认输入源为线路输入（Line In）。

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

**提示**

由于播放和录音共享硬件接口，因此无法在同时进行播放和录音操作时使用不同的采样率和格式。

Device Tree Audio configuration: https://github.com/phytec/linux-phytec-imx/blob/v6.6.52-2.2.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-segin.dts#L62

## 7.17 phyBOARD-Nash i.MX 93 上的音频

**警告**

由于硬件缺陷，phyBOARD-Nash i.MX 93 PCB 版本：1616.0 上的音频功能不可用

要在 phyBOARD-Nash i.MX 93 上使用音频，需要在 Audio/Video 接口连接一个扩展板。PEB-AV-10（1531.1 修订版）可以单独购买，并未随套件一起提供。PEB-AV-10 配备了 TI TLV320AIC3007 音频编解码器（CODEC）。音频支持通过 I2S 接口实现，并通过 I2C 控制。

有一个符合 OMTP 标准的 3.5mm 耳机插孔和一个 8 针接口，用于连接带有 AV 连接器的音频设备。这个 8 针接口包含单声道扬声器、耳机和线路输入信号（line-in）。

要检查您的声卡驱动程序是否正确加载以及设备名称，请输入以下命令以查看播放设备：

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

或输入录音设备：

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

### 7.17.1 Alsamixer

要检查声卡的功能，请输入：

```
target:~$ alsamixer
```

您应该会看到很多选项，因为音频 IC 具有许多可以测试的功能。通过 SSH 打开 alsamixer 的图形界面比通过调试串口打开更易于使用。所有混音点都有单声道或立体声增益控制。”MM” 表示该功能被静音（左右输出均为静音），可以通过按’ **m** ’ 切换。您还可以通过按’ < ’ 左和’ > ’ 切换右声道输出。使用 **tab** 键，您可以在播放和录音控制之间切换。

### 7.17.2 恢复默认音量

一些默认设置存储在 `/var/lib/alsa/asound.state` 中。您可以使用以下命令保存当前的 alsa 设置：

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

您可以通过以下方式恢复已保存的 alsa 设置：

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

### 7.17.3 ALSA 配置

我们的 BSP 附带一个 ALSA 配置文件 `/etc/asound.conf` 。

ALSA 配置文件可以根据需要进行编辑或删除，它并不是 ALSA 正常工作所必需的。

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

要将 PEB-AV-10 设置为输出，请将 *playback.pcm* 从”dummy” 设置为”pebav10”：

```
[...]

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

[...]
```

如果听不到声音，请将播放设备更改为软件音量控制播放设备，将 *playback.pcm* 设置为相应的软音量播放设备，例如 “softvol_pebav10”。使用 alsamixer 控制来调整音量级别。

```
[...]

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

[...]
```

### 7.17.4 PulseAudio 配置

对于使用 *Pulseaudio* 的应用程序，请检查可用的音频输出设备：

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

要选择输出设备，请输入：

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

### 7.17.5 播放

运行 speaker-test 以检查播放功能：

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

要播放简单的音频流，您可以使用 aplay。例如，要播放 ALSA 测试音频：

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

要播放其他格式，例如 mp3，您可以使用 Gstreamer：

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

### 7.17.6 录音

`arecord` 是一个命令行工具，用于录制音频流，默认输入源为线路输入。要选择不同的音频源，可以使用 `alsamixer`。例如，打开 *右侧 PGA 混音器 Mic3R* 和 *左侧 PGA 混音器 Mic3R*，以便通过 3.5mm 插孔录制来自 TLV320 编解码器的麦克风输入音频。

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

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

**提示**

由于播放和录音共享硬件接口，因此无法在同时进行播放和录音操作时使用不同的采样率和格式。

Device Tree Audio configuration: https://github.com/phytec/linux-phytec-imx/blob/v6.6.52-2.2.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-nash-peb-av-10.dtso#L56

## 7.18 视频

### 7.18.1 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
```

## 7.19 显示

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

The **phyBOARD-Nash** i.MX 93 needs additional adapter to support 10" `edt,etml1010g3dra` LVDS display with capacitive touchscreen. The PEB-AV-10 (1531.1 revision) can be bought separately to the Kit. Device-tree overlay for the aforementioned adapter is enabled in `/boot/bootenv.txt` by default!

### 7.19.1 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。

### 7.19.2 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。

The device tree of PEB-AV-02 can be found here: https://github.com/phytec/linux-phytec-imx/blob/v6.6.52-2.2.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-segin-peb-av-02.dtso

The device tree of PEB-AV-10 can be found here: https://github.com/phytec/linux-phytec-imx/blob/v6.6.52-2.2.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-nash-peb-av-10.dtso

## 7.20 电源管理

### 7.20.1 CPU 核心频率调节

i.MX 93 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。与 i.MX8 M 系列不同，i.MX 93 不支持 动态电压和频率调整（DVFS），但支持简化的 **电压和频率调整（VFS）**。可以进入以下模式：

- nominal（ND），
- overdrive （OD），
- Low Drive（LD）和
- Low Drive（LD）与软件快速频率变化（SWFFC）。

| 模式 | CPU 频率 | DDR 数据速率 | VDD_SOC |
|---|---|---|---|
| OverDrive (OD) | 1.7 GHz | 3733 MT/s | 900mV |
| NominalDrive (ND) | 1.4 GHz | 1866 MT/s | 850mV |
| LowDrive (LD) | 900 MHz | 1866 MT/s | 800mV |
| 带有 SWFFC 的 LowDrive（LD） | 900 MHz | 625 MT/s | 800mV |

i.MX 93 BSP 支持 VFS 功能。Linux 内核提供了一个 LPM 驱动程序，可以设置 VDD_SOC、CPU 频率和 DDR 速度。

**备注**

低成本的 i.MX 93 SoC 比如 NXP IMX9301/IMX9302 这些型号不支持 VFS 功能。这些 SoC 会固定运行在 LowDrive（LD）模式。因此在使用这些 SoC 的核心板时，Linux 的 LPM 驱动会自动禁用。

- 要将设备置于 **OverDrive（OD）**模式，请输入：

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

- 要将设备置于 **NominalDrive（ND）**模式，请输入：

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

- 要将设备置于 **LowDrive（LD）**模式，请输入：

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

- 将设备置于 **LowDrive（LD）**模式，使用最低 DDR 速度，SWFFC 类型：

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

- 要检查当前的 CPU 频率，请输入：

```
target:~$ mhz
```

- 要检查当前模式和 DDR 频率，请输入：

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

- 要检查当前的 VDD_SOC 类型，请输入：

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

有关 LPM 驱动程序和模式的更详细信息，请参考 NXP 的文档：https://docs.nxp.com/bundle/AN13917/page/topics/low_power_mode_use_cases.html

### 7.20.2 CPU 核心管理

i.MX 93 SoC 在芯片上拥有多个处理器核心。例如，i.MX 93 具有 2 个 ARM 核，这些核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/
cpufreq/
[...]
```

  这里系统有两个处理器核心。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/cpu1/online
```

  作为确认，您将看到：

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

  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/cpu1/online
```

### 7.20.3 挂起到 RAM

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

设备可以通过 PEB-EVAL-01 的 S2 按钮进入休眠状态并唤醒

要通过 RTC 闹钟唤醒，请检查：*RTC Wakealarm*

## 7.21 热管理

### 7.21.1 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
```

### 7.21.2 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 93 SoC 平台上使用热管理内核 API。i.MX 9 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温度阈值。这些阈值根据 CPU 型号的不同而有所区别。包括商业级、工业级和扩展工业级。

| | 商业 | 工业 | 扩展工业级 |
|---|---|---|---|
| 被动（警告） | 85°C | 95°C | 115°C |
| 严重（关机） | 90°C | 100°C | 120°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**

如果 sysfs 文件中的 SoC 温度值达到 *trip_point_1* ，主板将立即关闭以避免任何热损伤。如果这不符合您的期望，可以实现一个外部监控电路，当温度降到选定的触发点以下时，通过 X_ONOFF 信号重新启动模块

**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。

### 7.21.3 PWM 风扇

可以在 phyBOARD-Nash i.MX 93 的连接器 X48 (标注 FAN) 连接一个 PWM 风扇。

从本版本开始，需要先激活 PWM 风扇 Overlay，否则 PWM 风扇将无法被识别。

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

bootenv.txt 文件应该如下所示（它还可以包含其他设备树 overlay！）：

```
overlays=imx93-phyboard-nash-pwm-fan.dtbo
```

更改将在重启后应用：

```
target:~$ reboot
```

有关设备树 overlay 的更多信息，请阅读*device tree* 章节。

该 SoC 只包含一个温度传感器，而该传感器被用于热频率调节，因此风扇无法通过内核进行控制。我们使用 lmsensors 和 hwmon 来代替。lmsensors 定期读取温度，并在可配置的阈值下配置风扇 PWM 的输出。默认启动 PWM 风扇的温度阈值是 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
```

## 7.22 看门狗

PHYTEC i.MX 93 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。

### 7.22.1 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
```

## 7.23 bbnsm 电源键

连接到开/关按钮的 X_ONOFF 引脚可以长按（5 秒）以触发关机，而无需软件干预，或者用于唤醒系统以退出挂起状态。使用 *bbnsm_pwrkey* 驱动程序时，当按钮被按下时，KEY_POWER 事件也会报告给用户空间。默认情况下，systemd 被配置为忽略此类事件，并未配置无软件干预的关机功能。如果需要，可以在 `/etc/systemd/logind.conf` 中配置在按下开/关按钮时通过 systemd 触发关机：

```
HandlePowerKey=poweroff
```

## 7.24 PXP

i.MX 93 SoC 包含一个 PiXel Pipeline (PXP)。PXP 将以下内容组合成一个单独的处理引擎：

- 缩放
- 颜色空间转换 (CSC)
- 辅助色彩空间转换 (CSC2)
- 旋转

因此，减少了显示管道所需的内存占用。有关如何在 Gstreamer 和 Wayland 中使用 PXP，请查看 NXP 的《How to Use PXP in GStreamer and Wayland》（AN13829）应用笔记。

## 7.25 片上一次性可编程控制器 (OCOTP_CTRL) - eFuse

该 i.MX 93 提供一次性可编程寄存器（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 93 reference manual 中称为“片上一次性可编程控制器（OCOTP_CTRL）”）。以下列表是 i.MX 93 reference manual 的摘要，包括 OCOTP_CTRL 中的一些有用寄存器（基地址为 0x47510000）：

| 名称 | Bank | 字 | 内存偏移量为 0x47510000 | 描述 |
|---|---|---|---|---|
| BOOT_CFG0 | | 3 | 0 0x60 | 启动 fuse 设置 |
| BOOT_CFG1 | | 3 | 1 0x64 | 启动 fuse 设置 |
| BOOT_CFG2 | | 3 | 2 0x68 | 启动 fuse 设置 |
| BOOT_CFG3 | | 3 | 3 0x6c | 启动 fuse 设置 |
| MAC1_ADDR | 39 | 3 | 0x4ec | 包含 ENET0 MAC 地址的低 32 位 |
| MAC1/2_ADI | 39 | 4 | 0x4f0 | 包含 ENET0 MAC 地址的高 16 位和 ENET1 MAC 地址的低 16 位 |
| MAC2_ADDR | 39 | 5 | 0x4f4 | 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 93 Security Reference Manual 中的”Fuse Map” 部分。

### 7.25.1 在 uBoot 中读取 fuse 的值

MAC1_ADDR:

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

### 7.25.2 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\@0/47510000.efuse/fsb_s400_fuse0/nvmem
```

### 7.25.3 烧录 MAC 地址

假设我们想要烧录以下 MAC 地址：

| | |
|---|---|
| MAC1 | 12:34:56:78:90:AA |
| MAC2 | Bb:Cc:Dd:Ee:Ff:D0 |

我们将在 u-boot 中执行这个：

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

### 7.25.4 烧写启动 fuse

**警告**

fuse 只能写入一次！如果烧录了错误的启动配置，您可能会轻易地将您的板子变砖。这个过程是不可逆的！

可以在附带的名为 **i.MX93_Fusemap.xlsx** 的 excel 表格中查找应使用哪个 fuse bank/word 来编程 BOOT_CFGX，这些信息可以在 *i.MX 93 Applications Processor Reference Manual* 中找到。

这些值应该写入 BOOT_CFG0，可以从第 3 个 Bank 的第 0 字节中读取/写入 fuse。

| 启动设备 | BOOT_CFG0 |
|---|---|
| eMMC | 0x20020002 |
| SD 卡 | 0x20000103 |

要设置内部 fuse 以从 eMMC 启动，可以使用以下方法进行编程：

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

在这个例子中，我们：

- 将 Boot_Mode 设置为 0b0010 （eMMC），也就是 BOOT_CFG0[3:0]，
- 将 eMMC 总线宽度设置为 0b01（8 位），也就是 BOOT_CFG0[18:17]。
- 设置 BT_FUSE_SEL（Boot fuses already programmed）位，也就是 BOOT_CFG0[29]

确保通过阅读 *i.MX 93 Applications Processor Reference Manual* 中的 **Boot Fusemap** 章节来设置正确的位。

## 7.26 TPM

**phyBOARD-Nash** i.MX 93 配备了可信任的平台模块（TPM），提供基于硬件的安全功能。

以下是一些与 TPM 相关的示例

使用 TPM2 工具生成 4 字节随机值：

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

使用 OpenSSL 工具生成 4 字节随机值：

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

生成 RSA 私钥并验证其内容：

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

**备注**

如果您打算将这些密钥用于任何安全目的，请不要共享您的私有 RSA 密钥。

生成 RSA 公钥并验证其内容：

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

Device tree TPM configuration can be found here: https://github.com/phytec/linux-phytec-imx/blob/v6.6.52-2.2.0-phy9/arch/arm64/boot/dts/freescale/imx93-phyboard-nash.dts#L161

CHAPTER 8

# i.MX 93 M33 Core

除了 Cortex-A55 核心外，SoC 中还集成了一个 Cortex-M33 Core 作为单片机（MCU）。我们的 Yocto-Linux-BSP 在 A55 核心上运行，而 M33 Core 可以作为辅助核心使用，执行额外的任务，采用 bare-metal 的固件。两个核心都可以访问相同的外设，因此在 M33 Core 的固件或 Linux 操作系统的设备树中，需要限制外设的使用。

我们的 Yocto-BSP 包含来自 NXP 的 M33 Core 预编译固件示例。

本节描述了如何在 phyBOARD-Segin/Nash i.MX 93 上运行预编译的 M33 Core 固件示例。

## 8.1 运行 M33 Core 示例

运行 M33 Core 固件示例有两种方式：从 U-Boot bootloader 和在正在运行的 Linux 中使用 Remoteproc 子系统。

要接收调试信息，请在您的主机 PC 上启动您喜欢的终端软件（例如 Minicom、Tio 或 Tera Term），并将其配置为 115200 波特率、8 个数据位、无奇偶校验和 1 个停止位（8n1），且不使用握手。

在 phyBOARD-Segin/Nash i.MX 93 上需要一个“USB 接口 TTL 串口转换器”。转换器的 I/O 引脚应能够在 3.3V 电压水平下工作。

根据下表，将外部“USB 接口 TTL 串口转换器”信号连接到板上的*X16* 连接器：

| USB-TTL 适配器引脚 | X16 信号 | X16 引脚 |
|---|---|---|
| RXD | X_UART2_TX | 5 |
| TXD | X_UART2_RX | 8 |
| GND | GND | 4 |

### 8.1.1 从 U-Boot 运行示例

要使用 U-Boot bootloader 加载固件示例，可以使用 `bootaux` 命令：

1. 准备一张烧写了 BSP 镜像的 SD 卡
2. 通过按任意键停止自动启动

3. 列出 SD 卡第一分区上可用的 M33 Core 固件示例：

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

**备注**

可用的固件示例以 `imx93-11x11-evk_m33_TCM_*` 开头，以 `*.bin` 结尾。这些示例来自 NXP 的 Yocto 层 meta-imx，并根据与 phyBOARD-Segin/Nash i.MX 93 硬件的兼容性进行选择。

1. 加载所需的固件示例：

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

程序的输出应显示在 M33 Core 的调试 UART 上。

### 8.1.2 通过 Remoteproc 在 Linux 上运行示例

Remoteproc 是一个模块，它允许您在运行时从 Linux 控制 M33 Core 。可以加载 M33 Core 的固件示例，并在 Linux 中控制它启动或停止。要使用 Remoteproc，需要设置设备树 overlay：

在开发板的 `/boot` 目录中编辑 `bootenv.txt` 文件，添加 `imx93-phycore-rpmsg.dtbo` ：

```
overlays=imx93-phyboard-segin-peb-av-02.dtbo imx93-phycore-rpmsg.dtbo
```

重启目标并在 U-Boot 中执行:

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

针对 M33 Core 的固件示例 `*.elf` 文件可以在 `/lib/firmware` 下找到。列出可用的固件示例：

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

要加载固件，请输入：

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

要加载不同的固件，M33 Core 需要停止：

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

**备注**

在开发板的 `/lib/firmware` 目录中找到的例子来自 NXP 的 Yocto 层 meta-imx，并根据与 phyBOARD-Segin/Nash i.MX 93 硬件的兼容性进行了挑选。

NXP 的一些固件示例需要加载额外的 Linux 内核模块。

例如，当加载 `imx93-11x11-evk_m33_TCM_rpmsg_lite_str_echo_rtos.elf` 固件时，需要加载相应的 `imx_rpmsg_tty` 模块：

```
target:~$ modprobe imx_rpmsg_tty
```

这将一个 RPMsg 端点作为虚拟 TTY 暴露在 `/dev/ttyRPMSG30` 。现在可以通过输入以下内容从 A55 核心发送消息到 M33 Core ：

```
target:~$ echo "PHYTEC" > /dev/ttyRPMSG30
```

观察 M33 Core 调试 UART 应该会产生以下输出：

```
RPMSG String Echo FreeRTOS RTOS API Demo...

Nameservice sent, ready for incoming messages...
Get Message From Master Side : "PHYTEC" [len : 6]
```

## 8.2 ARM Ethos-U NPU

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

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

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

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

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

### 8.2.1 Running NPU examples

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

```
overlays=imx93-phyboard-segin-peb-av-02.dtbo imx93-phycore-rpmsg.dtbo imx93-phycore-npu.dtbo
```

Reboot the target afterwards.

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

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

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

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

Network summary for mobilenet_v1_1.0_224_quant
Accelerator configuration              Ethos_U65_256
System configuration                internal-default
```

(续下页)

(接上页)

```
Memory mode                         internal-default
Accelerator clock                               1000 MHz
Design peak SRAM bandwidth                     14.90 GB/s
Design peak DRAM bandwidth                      3.49 GB/s

Total SRAM used                               370.91 KiB
Total DRAM used                              3719.84 KiB

CPU operators = 0 (0.0%)
NPU operators = 60 (100.0%)

Average SRAM bandwidth                          5.53 GB/s
Input   SRAM bandwidth                         11.40 MB/batch
Weight  SRAM bandwidth                          9.25 MB/batch
Output  SRAM bandwidth                          4.10 MB/batch
Total   SRAM bandwidth                         24.86 MB/batch
Total   SRAM bandwidth            per input     24.86 MB/inference (batch size 1)

Average DRAM bandwidth                          1.17 GB/s
Input   DRAM bandwidth                          1.45 MB/batch
Weight  DRAM bandwidth                          3.08 MB/batch
Output  DRAM bandwidth                          0.72 MB/batch
Total   DRAM bandwidth                          5.26 MB/batch
Total   DRAM bandwidth            per input      5.26 MB/inference (batch size 1)

Neural network macs                        572406226 MACs/batch
Network Tops/s                                  0.25 Tops/s

NPU cycles                                    3732560 cycles/batch
SRAM Access cycles                            1016537 cycles/batch
DRAM Access cycles                            1676662 cycles/batch
On-chip Flash Access cycles                         0 cycles/batch
Off-chip Flash Access cycles                        0 cycles/batch
Total cycles                                  4491867 cycles/batch

Batch Inference time                 4.49 ms,  222.62 inferences/s (batch size 1)
```

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

**备注**

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

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

```
target:examples$ ./label_image -m output/mobilenet_v1_1.0_224_quant_vela.tflite \
                 -i grace_hopper.bmp -l labels.txt \
                 --external_delegate_path=/usr/lib/libethosu_delegate.so
INFO: Loaded model output/mobilenet_v1_1.0_224_quant_vela.tflite
INFO: resolved reporter
INFO: Ethosu delegate: device_name set to /dev/ethosu0.
remoteproc remoteproc0: powering up imx-rproc
```

(续下页)

(接上页)

```
remoteproc remoteproc0: Booting fw image ethosu_firmware, size 242844
INFO: Ethosu delegate: cache_file_path set to .
INFO: Ethosu delegate: timeout set to 60000000000.
INFO: Ethosu delegate: enable_cycle_counter set to 0.
INFO: Ethosu delegate: enable_profiling set to 0.
INFO: Ethosu delegate: profiling_buffer_size set to 2048.
INFO: Ethosu delegate: pmu_event0 set to 0.
INFO: Ethosu delegate: pmu_event1 set to 0.
INFO: Ethosu delegate: pmu_event2 set to 0.
INFO: Ethosu delegate: pmu_event3 set to 0.
INFO: EXTERNAL delegate created.
rproc-virtio rproc-virtio.1.auto: assigned reserved memory node vdevbuffer@a4020000
virtio_rpmsg_bus virtio0: rpmsg host is online
rproc-virtio rproc-virtio.1.auto: registered virtio0 (type 7)
rproc-virtio rproc-virtio.2.auto: assigned reserved memory node vdevbuffer@a4020000
virtio_rpmsg_bus virtio1: rpmsg host is online
virtio_rpmsg_bus virtio1: creating channel rpmsg-ethosu-channel addr 0x1e
rproc-virtio rproc-virtio.2.auto: registered virtio1 (type 7)
remoteproc remoteproc0: remote processor imx-rproc is now up
INFO: EthosuDelegate: 1 nodes delegated out of 1 nodes with 1 partitions.
INFO: Applied EXTERNAL delegate.
INFO: invoked
INFO: average time: 3.778 ms
INFO: 0.780392: 653 military uniform
INFO: 0.105882: 907 Windsor tie
INFO: 0.0156863: 458 bow tie
INFO: 0.0117647: 466 bulletproof vest
INFO: 0.00784314: 835 suit
```

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

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

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

**备注**

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