# Yocto Reference Manual Mickledore

**PHYTEC Messtechnik GmbH**

**2024 年 12 月 04 日**

# Contents

| Yocto Reference Manual | |
|---|---|
| 文档标题 | Yocto Reference Manual Mickledore |
| 文档类型 | Yocto 手册 |
| 发布日期 | XXXX/XX/XX |
| 母文档 | Yocto Reference Manual |

| **适用** BSP | BSP **发布类型** | BSP **发布日期** | BSP **状态** |
|---|---|---|---|
| BSP-Yocto-NXP-i.MX93-PD24.1.0 | 大版本 | 2024 年 5 月 2 日 | 已发布 |
| BSP-Yocto-NXP-i.MX93-PD24.1.1 | 小更新 | 2024 年 5 月 8 日 | 已发布 |

本手册适用于所有基于 Mickledore 的 PHYTEC BSP 版本。

CHAPTER 1

# PHYTEC 文档

PHYTEC 将为旗下所有产品提供各种硬件和软件文档。包括以下任一以及全部内容：

- **快速上手指南**：简单指导我们如何配置和启动 phyCORE 核心板，以及对构建 BSP、设备树和外设访问进行简要说明。
- **硬件手册**：核心板和配套底板的详细硬件描述。
- **Yocto 手册**：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 上搭建完整的应用开发环境。
- **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。大多数文档都可以在我们产品的下载页面中找到。

CHAPTER 2

# Yocto 介绍

Yocto 是最小的国际单位制前缀。就像 milli 是描述 `10^-3` 一样，yocto 是描述 `10^-24` 。Yocto 也是 Linux 基金会 支持的项目，因此得到了该领域几家主流公司的支持。在 Yocto 项目网站 上您可以阅读官方的介绍：

> Yocto 项目是一个开源协作项目，它提供模板、工具和方法，为您的嵌入式产品创建基于 Linux 的自定义系统，同时不用过多的关注底层硬件架构。该项目由众多硬件制造商、开源操作系统供应商和电子公司合作开发，成立于 2010 年，旨在引导嵌入式 Linux 系统开发走向标准化。

如前所述，该项目希望为嵌入式开发人员提供工具集。它建立在长期维护的 OpenEmbedded 项目之上。它不是一个 Linux 发行版，但它包含创建定制化的 Linux 发行版的所有工具。维护工具集的最重要步骤是定义一个通用的版本控制方案和一个参考系统。然后，所有子项目都必须兼容参考系统，并且遵守他的版本控制方案。

发布过程类似于 Linux kernel 的发布机制。Yocto 每六个月增加一次版本号，并为发布版本指定版本代号。发布列表可在此处找到：https://wiki.yoctoproject.org/wiki/Releases

CHAPTER 3

# 核心组件

*Yocto* 项目最重要的工具或子项目是：

- Bitbake：构建引擎，是一个类似 make 的任务调度程序，解析元数据
- OpenEmbedded-Core，Yocto 核心 Layer，包含软件元数据
- Yocto kernel
  - 针对嵌入式设备进行了优化
  - 包括许多子项目：rt-kernel、供应商补丁
  - Wind River 提供的基础框架
  - Yocto kernel 的替代方案：经典的内核编译方式 →Phytec 使用这种方式将我们的 kernel 集成到 *Yocto*
- *Yocto* 参考 BSP：*beagleboneblack*、*minnow max*
- *Poky*：Yocto 参考系统，是项目和工具的集合，是创建基于 *Yocto* 的 Linux 发行版的基石

CHAPTER 4

# 词汇

## 4.1 Recipes

Recipe 包含有关软件项目的信息（作者、主页和许可证）。Recipe 有版本控制，它定义了依赖项，包含源代码的 URL，并描述如何获取、配置和编译源代码。它也描述了如何打包软件，例如打包成不同的.deb 包，安装到目标系统不同的路径。Recipe 是用 *Bitbake* 自己的编程语言编写的，语法很简单。但是，Recipe 可以包含 *Python* 以及 bash 代码

## 4.2 类

类将 Recipe 中的各个方法组合成可重复使用的代码块。

## 4.3 Layers

layer 是 Recipe、类和配置元数据的集合。Layer 可以依赖于其他 Layer, 它封装了特定的功能。每个 Layer 都属于一个特别的 category：

- Base
- Machine（BSP）
- Software
- Distribution
- Miscellaneous

*Yocto* 的版本控制方案在每一个 Layer 中以版本分支的形式体现。对于每个 *Yocto* 版本，每个 Layer 在其 *Git* 仓库中都有一个对应分支。您可以在 Yocto 工程中添加一个或多个 Layer。

可以在这里 https://layers.openembedded.org/layerindex/branch/mickledore/layers/找到 OpenEmbedded Layer 的集合。搜索功能非常有用，可以轻松检索和集成软件包。

## 4.4 Machine

Machine 是用来描述所使用的目标硬件 (核心板/开发套件) 的变量。

## 4.5 发行版 (Distro)

发行版描述了软件配置以及一系列的软件特性。

CHAPTER 5

# Poky

*Poky* 是定义 *Yocto* 项目的参考系统。它将几个子项目组合成发行版：

- *Bitbake*
- *Toaster*
- OpenEmbedded Core
- *Yocto* 文档
- *Yocto* 参考 BSP

## 5.1 Bitbake

*Bitbake* 是任务调度器。它是一个 Python 脚本，解释用 *Bitbake* 自己的编程语言、*Python* 或者 bash 代码编写的 recipe。官方文档可在此处找到：https://docs.yoctoproject.org/bitbake/2.4/index.html

## 5.2 Toaster

*Toaster* 是 *Bitbake* 的用于启动和分析工程构建的 Web 前端。它提供有关编译历史和所生成镜像的统计信息。在多个案例中，安装和维护 *Toaster* 实例是有益的。PHYTEC 未对 *Poky* 提供的 *Toaster* 作任何功能添加或删除。如果想了解更多，请参考官方文档：https://docs.yoctoproject.org/4.2.4/toaster-manual/index.html

CHAPTER 6

# 官方文档

有关 *Bitbake* 和 *Poky* 的更多常见问题，请参阅手册：https://docs.yoctoproject.org/4.2.4/singleindex.html

CHAPTER 7

# Linux 主机开发环境

要编译 *Yocto*，您需要一台合适的 *Linux* 主机开发环境。支持的 Linux 发行版列表可在参考手册中找到：
https://docs.yoctoproject.org/4.2.4/ref-manual/system-requirements.html#supported-linux-distributions

CHAPTER 8

# PHYTEC BSP 介绍

## 8.1 BSP 框架

BSP 大致由三部分组成。BSP 包管理、BSP 元数据和 BSP 代码源。包管理包括 *Repo* 和 phyLinux，而元数据取决于 SOC，它描述了如何构建软件。BSP 内容源来自 PHYTEC 的 *Git* 仓库和外部源。

### 8.1.1 BSP 包管理

*Yocto* 是一个综合项目。通常情况下，这意味着会强制用户将他们的 Yocto 项目建立在几个外部仓库上。这些库需要以特定的方式进行管理。我们使用包含 XML 数据结构的清单文件来描述不同版本的 git 仓库。*Repo* 工具和我们的 phyLinux 脚本用于管理清单文件并配置 BSP 工程。

#### phyLinux

phyLinux 是 *Repo* 的包装器，用于下载和配置 BSP，提供“开箱即用”的体验。

#### Repo

*Repo* 是 *Repo* 工具集的包装器。phyLinux 脚本将把 Repo 安装在全局 PATH 中。当您首次运行 `repo init -u <url>`，它会从 *Googles* Git 服务器下载特定版本的 *Repo* 工具集到 `.repo/repo` 目录。下次运行 *Repo* 时，*Repo* 工具集相关的指令就可以直接被使用了。请注意，如果您不运行 *Repo sync*，不同构建目录中的 *Repo* 工具集版本可能会随着时间的推移而有所不同。此外，如果您要保存您的完整 BSP 工程信息，也需要保存完整的 `.repo` 文件夹。

*Repo* 需要一个 *Git* 仓库，该仓库信息是从 Repo 命令中解析得来。在 PHYTEC BSP 中，该仓库被称为 phy²octo。在此仓库中，有关软件 BSP 版本的所有信息都以 XML 形式的 *Repo* 清单存储。清单文件定义了 *Git* 服务器（称为“Remote”）的 URL、*Git* 仓库及其状态（称为“projects”）。*Git* 仓库可以呈现不同的状态。这其中的状态变化涉及仓库的分支、标签或 commit ID。这意味着仓库的状态不一定是唯一的，可以随时间而变化。这就是我们仅使用标签或 commit ID 进行发布的原因。这样的话 git 工作目录的状态就是唯一的，不会改变。

BSP 的清单文件与 BSP 版本号同名。它是 BSP 的唯一标识符。发布的 BSP 会按 SoC 平台排序。所选 SoC 将定义 phy²octo *Git* 仓库的分支，该分支将用于选择对应的清单文件。

### 8.1.2 BSP 元数据

我们在 BSP 中包含了几个第三方 Layer，无需集成其他外部项目即可运行完整的 *Linux* 发行版。以下描述了所有使用的 git 仓库。

#### Poky

PHYTEC BSP 建立在 *Poky* 之上。每个 Poky 有对应的版本，在 *Repo* 清单中定义。*Poky* 包含对应版本的 *Bitbake*。OpenEmbedded Core 的 Layer- “meta” 是我们自定义 *Linux* 系统的基石。

#### meta-openembedded

OpenEmbedded 是一组 Layer，包含有许多开源软件项目的元数据。我们的 BSP 提供了所有的 OpenEmbedded Layer，但并非所有 Layer 都已使能。我们的示例镜像包含了几个 OpenEmbedded Layer 中的 Recipe 所生成的软件包。

#### meta-qt6

该 Layer 在 *Poky* 根文件系统的基础上集成了 *Qt6* ，我们的 BSP 已经包含了该 Layer。

#### meta-nodejs

这是用于添加最新版本 Node.js 的 Layer。

#### meta-gstreamer1.0

这是用于添加最新版本 GStreamer 的 Layer。

#### meta-rauc

这一 Layer 包含搭建 RAUC 系统更新服务所需的工具. 与其他系统更新机制的对比可以在这里找到：Yocto 系统更新工具.

#### meta-phytec

这一 Layer 包含我们所有 BSP 的所有 machine 和通用特性。它是 PHYTEC 的 Yocto BSP 适用于所有受支持的硬件（从 *fido* 开始），并且设计为与 *Poky* Layer 相互独立。如果您想将 PHYTEC 的硬件集成到现有的 *Yocto* 项目中，只需要这两个 Layer 即可。其特点是：

- `recipes-bsp/barebox/` 和 `recipes-bsp/u-boot/` 中的 Bootloader
- `recipes-kernel/linux/` 和 `dynamic-layers/fsl-bsp-release/recipes-kernel/linux/` 中的 kernel
- `conf/machine/` 中有许多 machine
- 适配 AM335x 和 i.MX 6 平台的 *OpenGL ES/EGL* 上层库
- 适配 i.MX 6 平台的 *OpenCL* 库

#### meta-ampliphy

这是我们的发行版示例 layer。它扩展了 *Poky* 的基础配置，为您的特定开发场景提供了基础。当前功能包括：

- systemd 初始化系统
- 镜像：用于非图形应用的 `phytec-headless-image`
- 基于 i.MX 6 平台的相机、OpenCV 和 GStreamer 集成示例，包含在 `phytec-vision-image` 中
- 集成 RAUC：我们为 A/B 系统镜像更新提供了基础支持，该更新可在本地和远程进行

#### meta-qt6-phytec

这是我们用于集成 Qt6 和展示 Qt6 demo 的 Layer。其特点是：

- 针对 PHYTEC AM335x、i.MX 6 和 RK3288 平台的 带有 eglfs 后台的 Qt6
- 镜像：带有 *Qt6* 以及视频应用的 `phytec-qt6demo-image`
- *Qt6* 示例程序演示如何使用 *QML* widgets 和一个 *Bitbake* recipe 在 *Yocto* 搭建 *Qt6* 项目并集成到 *systemd* 中。该示例程序可以在 `sources/meta-qt6-phytec/recipes-qt/examples/phytec-qtdemo_git.bb` 中找到

#### meta-virtualization

- 该 Layer 为构建 Xen、KVM、Libvirt 以及其他基于 OpenEmbedded 的虚拟化解决方案提供必要的支持。

#### meta-security

- 该 Layer 提供安全工具、Linux 内核的强化工具以及实现安全机制的库。

#### meta-selinux

- 此 Layer 的目的是启用 SE Linux 支持。此 Layer 的大部分工作是在 *bbappend* 文件中完成的，用于在现有 Recipe 中启用 SE Linux 支持。

#### meta-browser

- 这是用于添加常用 Web 浏览器（Chromium、Firefox 等）的 Layer。

#### meta-rust

- 包括 Rust 编译器和 Rust 的 Cargo 包管理器。

#### meta-timesys

- Timesys Layer 用于 Vigiles Yocto CVE 监控、安全提醒和镜像清单的生成。

#### meta-freescale

- 该 Layer 为 i.MX、Layerscape 和 QorIQ 产品线提供支持。

#### meta-freescale-3rdparty

- 为来自不同供应商的主板提供支持。

#### meta-freescale-distro

- 该 Layer 为 freescale 的演示镜像提供支持，以便与 OpenEmbedded 和/或 Yocto Freescale 的 BSP Layer 一起使用。

#### base layer（fsl-community-bsp-base）

- 该 layer 提供 NXP 的基础 BSP 文件。

#### meta-fsl-bsp-release

- 这是 i.MX Yocto 项目发行版的 Layer。

### 8.1.3 BSP 代码源

当您首次开始使用 *Bitbake* 时，BSP Content 会从不同的线上源提取。所有文件都将下载并拷贝到在 *Yocto* 中配置为 “DL_DIR“的本地目录中。如果您想备份包含完整内容的 BSP，则也必须备份这些源文件。在生成镜像源，开启离线构建 一章中会作出进一步解释。

## 8.2 编译配置

BSP 初始化一个编译文件夹，该文件夹将包含运行 *Bitbake* 命令所生成的所有文件。它也包含一个用于处理用户输入的 `conf` 文件夹。

- `bblayers.conf` 定义使能的元 layers，
- `local.conf` 可以自定义 Yocto 工程的用户输入变量
- `site.conf` 自定义与编译主机相关的输入变量

两个最顶层的用户输入变量是 `DISTRO` 和 `MACHINE` 。当您使用 phyLinux 的方式获取 BSP 时，它们会体现在 BSP 预先配置的 `local.conf` 文件中。

我们在 BSP 中使用 “*Ampliphy*” 作为软件发行版 `DISTRO` 。此 DISTRO 将被预先选定，并为您提供实现自定义软件发行版的起点。

`MACHINE` 定义支持特定核心板和底板的二进制镜像。请查看 `machine.conf` 文件或我们的网页以了解硬件的描述。

CHAPTER 9

# 预编译镜像

对于每个 BSP，我们都提供了预编译的目标镜像，可以从 PHYTEC FTP 服务器下载：https://download.phytec.de/Software/Linux/

这些镜像也可用于 BSP 测试，他们会在生产时烧写到产品中。您可以使用提供的 `.wic` 镜像创建 SD 卡启动盘。识别您的硬件 Machine 并使用 `dd` 将下载的镜像文件烧写到格式化的 SD 卡中。有关该命令的正确用法的信息，请参阅镜像部分。

CHAPTER 10

# BSP Workspace 安装

## 10.1 设置主机

您可以设置主机或使用我们的编译容器来进行 Yocto 工程构建。您需要有一个运行中的 *Linux* 发行版系统，它应该在一台性能和存储空间强大的机器上运行，因为 Yocto 需要进行大量编译。

如果您想使用编译容器，您只需要在主机上安装以下软件包

```
host:~$ sudo apt install wget git
```

之后继续下一步*Git 配置*。关于如何使用编译-container ，您可以在本文档 phyLinux 的高级用法 之后章节找到。

如果您不想使用编译 container，*Yocto* 需要在您主机上安装一些其他的软件包。对于 *Ubuntu*，您需要

```
host:~$ sudo apt install gawk wget git diffstat unzip texinfo \
      gcc build-essential chrpath socat cpio python3 python3-pip \
      python3-pexpect xz-utils debianutils iputils-ping python3-git \
      python3-jinja2 libegl1-mesa libsdl1.2-dev \
      python3-subunit mesa-common-dev zstd liblz4-tool file locales
```

对于其他发行版，您可以在 *Yocto* Quick Build 中找到信息: https://docs.yoctoproject.org/4.2.4/brief-yoctoprojectqs/index.html

## 10.2 Git 配置

BSP 大量使用 *Git*。*Git* 需要您提供一些信息来识别谁对文件进行了更改。如果您没有 `~/.gitconfig` 这个文件，请创建一个包含以下内容的文件

```
[user]
    name = <Your Name>
    email = <Your Mail>
[core]
    editor = vim
```

(续下页)

(接上页)

```
[merge]
    tool = vimdiff
[alias]
    co = checkout
    br = branch
    ci = commit
    st = status
    unstage = reset HEAD --
    last = log -1 HEAD
[push]
    default = current
[color]
    ui = auto
```

您应该在 *Git* 配置中设置 `name` 和 `email`，否则，*Bitbake* 会在第一次构建时报错。您可以使用这两个命令直接设置它们，而无需手动编辑 `~/.gitconfig`

```
host:~$ git config --global user.email "your_email@example.com"
host:~$ git config --global user.name "name surname"
```

## 10.3 site.conf 设置

在开始编译 *Yocto* 之前，建议进行初步配置。有两个配置最为重要：下载目录和缓存目录的配置。PHYTEC 强烈建议对这两个配置项进行配置，因为它将减少您后续编译的时间。

下载目录是 *Yocto* 存储从互联网获取的所有源文件/源代码的地方。它可以包含 tar.gz、*Git* 镜像源等。建议将下载目录设置为编译主机上用户可以共享的目录。我们首先要给这个目录赋予 777 访问权限，因为如果要让不同用户共享此目录，则所有文件都需要具有组写入访问权限。这很可能与默认 *umask* 设置冲突。一种可能的解决方案是对此目录使用 ACL

```
host:~$ sudo apt-get install acl
host:~$ sudo setfacl -R -d -m g::rwx <dl_dir>
```

如果您已经创建了下载目录，但是后续想要更改权限，可以使用

```
host:~$ sudo find /home/share/ -perm /u=r ! -perm /g=r -exec chmod g+r \{\} \;
host:~$ sudo find /home/share/ -perm /u=w ! -perm /g=w -exec chmod g+w \{\} \;
host:~$ sudo find /home/share/ -perm /u=x ! -perm /g=x -exec chmod g+x \{\} \;
```

缓存目录存储编译过程的所有阶段产物。*Poky* 具有相当复杂的缓存架构。建议创建一个共享目录，这样所有构建都可以访问此缓存目录，这被称为共享状态缓存。

在大约有 50 GB 空闲空间的硬盘上创建这两个目录，并在 “build/conf/local.conf“中分配两个变量

```
DL_DIR ?= "<your_directory>/yocto_downloads"
SSTATE_DIR ?= "<your_directory>/yocto_sstate"
```

如果您想了解更多有关如何配置 Yocto 工程的信息，请参阅 Yocto 工程中的文档示例设置

```
sources/poky/meta-yocto/conf/local.conf.sample
sources/poky/meta-yocto/conf/local.conf.sample.extended
```

CHAPTER 11

# phyLinux 文档

phyLinux 脚本是使用 *Python* 编写，用来管理 PHYTEC *Yocto* BSP 版本。它主要是帮助用户快速上手 PHYTEC BSP。您无需与 *Repo* 或 *Git* 交互，只使用 phyLinux 可以获取所有 BSP 源文件

phyLinux 脚本只有一个依赖项，那就是它需要您在主机上安装 *wget* 工具。执行后它会将 Repo 工具 安装到您主机上的全局 PATH (/usr/local/bin) 中。您也可以手动安装到其他位置。如果已经在 PATH 中找到 *Repo*，phyLinux 将自动检测到并使用它。*Repo* 工具被用来管理 *Yocto* BSP 的众多 *Git* 仓库。

## 11.1 获取 phyLinux

phyLinux 脚本可以在 PHYTEC 下载服务器上找到：https://download.phytec.de/Software/Linux/Yocto/Tools/phyLinux

## 11.2 基本用法

对于 phyLinux 的基本用法，请输入

```
host:~$ ./phyLinux --help
```

这将导致

```
usage: phyLinux [-h] [-v] [--verbose] {init,info,clean} ...

This Programs sets up an environment to work with The Yocto Project on Phytecs
Development Kits. Use phyLinx <command> -h to display the help text for the
available commands.

positional arguments:
  {init,info,clean}  commands
    init             init the phytec bsp in the current directory
    info             print info about the phytec bsp in the current directory
    clean            Clean up the current working directory
```

(续下页)

(接上页)

```
optional arguments:
  -h, --help         show this help message and exit
  -v, --version      show program's version number and exit
  --verbose
```

## 11.3 初始化

创建一个新的项目文件夹

```
host:~$ mkdir ~/yocto
```

调用 phyLinux 将使用系统上安装的默认 Python 版本。从 Ubuntu 20.04 开始，默认是 Python3。如果您想启动与 Python3 不兼容的 BSP，您需要在运行 phyLinux 之前将 Python2 设置为默认值（临时）

```
host:~$ ln -s \`which python2\` python && export PATH=`pwd`:$PATH
```

现在在新文件夹下运行 phyLinux

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

空的文件夹很重要，因为 phyLinux 将会清空该目录。在非空目录运行 phyLinux 会有以下 **告警**:

```
This current directory is not empty. It could lead to errors in the BSP configuration
process if you continue from here. At the very least, you have to check your build directory
for settings in bblayers.conf and local.conf, which will not be handled correctly in
all cases. It is advisable to start from an empty directory of call:
$ ./phyLinux clean
Do you really want to continue from here?
[yes/no]:
```

在第一次初始化时，phyLinux 脚本会要求您在 */usr/local/bin* 目录中安装 *Repo* 工具。在执行 *init* 命令期间，您需要选择处理器平台 (SoC)、PHYTEC 的 BSP 版本号以及您正在使用的 Machine

```
**************************************************
* Please choose one of the available SoC Platforms:
*
*   1: am335x
*   2: am57x
*   3: am62ax
*   4: am62x
*   5: am64x
*   6: am68x
*   7: imx6
*   8: imx6ul
*   9: imx7
*   10: imx8
*   11: imx8m
*   12: imx8mm
*   13: imx8mp
*   14: imx8x
*   15: imx93
*   16: nightly
*   17: rk3288
*   18: stm32mp13x
*   19: stm32mp15x
```

(续下页)

(接上页)

```
*   20: topic

# Exemplary output for chosen imx93
************************************************
* Please choose one of the available Releases:
*
*   1: BSP-Yocto-NXP-i.MX93-ALPHA1
*   2: BSP-Yocto-NXP-i.MX93-PD24.1-rc1
*   3: BSP-Yocto-NXP-i.MX93-PD24.1.0
*   4: BSP-Yocto-NXP-i.MX93-PD24.1.1-rc1
*   5: BSP-Yocto-NXP-i.MX93-PD24.1.1-rc2
*   6: BSP-Yocto-NXP-i.MX93-PD24.1.1-rc3
*   7: BSP-Yocto-NXP-i.MX93-PD24.1.1

# Exemplary output for chosen BSP-Yocto-NXP-i.MX93-PD24.1.1
*********************************************************************
* Please choose one of the available builds:
*
no:                 machine: description and article number
                             distro: supported yocto distribution
                             target: supported build target

1: phyboard-nash-imx93-1:  PHYTEC phyBOARD-Nash i.MX93
                           2 GB RAM, eMMC
                           PB-04729-001, PCL-077-23231211I
                           distro: ampliphy-vendor
                           target: phytec-headless-image
2: phyboard-nash-imx93-1:  PHYTEC phyBOARD-Nash i.MX93
                           2 GB RAM, eMMC
                           PB-04729-001, PCL-077-23231211I
                           distro: ampliphy-vendor-rauc
                           target: phytec-headless-bundle
3: phyboard-nash-imx93-1:  PHYTEC phyBOARD-Nash i.MX93
                           2 GB RAM, eMMC
                           PB-04729-001, PCL-077-23231211I
                           distro: ampliphy-vendor-wayland
                           target: -c populate_sdk phytec-qt6demo-image
                           target: phytec-qt6demo-image
4: phyboard-segin-imx93-2: PHYTEC phyBOARD-Segin i.MX93
                           1 GB RAM, eMMC, silicon revision A1
                           PB-02029-001, PCL-077-11231010I
                           distro: ampliphy-vendor
                           target: phytec-headless-image
5: phyboard-segin-imx93-2: PHYTEC phyBOARD-Segin i.MX93
                           1 GB RAM, eMMC, silicon revision A1
                           PB-02029-001, PCL-077-11231010I
                           distro: ampliphy-vendor-rauc
                           target: phytec-headless-bundle
6: phyboard-segin-imx93-2: PHYTEC phyBOARD-Segin i.MX93
                           1 GB RAM, eMMC, silicon revision A1
                           PB-02029-001, PCL-077-11231010I
                           distro: ampliphy-vendor-wayland
                           target: phytec-qt6demo-image
```

如果您无法通过以上 phyLinux 选择器提供的信息识别您的所使用的硬件，请查看 PHYTEC 产品发票。配置完成后，您可以运行

```
host:~$ ./phyLinux info

# Exemplary output
*******************************************
* The current BSP configuration is:
*
* SoC:  refs/heads/imx93
* Release:  BSP-Yocto-NXP-i.MX93-PD24.1.1
* Machine:  phyboard-segin-imx93-2
*
*******************************************
```

查看当前 BSP 选择了哪款 SoC 和 BSP 版本。如果您不想使用 phyLinux 提供的选择器，phyLinux 还支持多种命令行参数去设置 Soc 和 BSP 版本

```
host:~$ MACHINE=phyboard-segin-imx93-2 ./phyLinux init -p imx93 -r BSP-Yocto-NXP-i.MX93-PD24.1.1
```

或者查看帮助命令以获取更多信息

```
host:~$ ./phyLinux  init --help

usage: phyLinux init [-h] [--verbose] [--no-init] [-o REPOREPO] [-b REPOREPO_BRANCH] [-x XML] [-u URL] [-p␣
↪PLATFORM] [-r RELEASE]

options:
  -h, --help          show this help message and exit
  --verbose
  --no-init           dont execute init after fetch
  -o REPOREPO         Use repo tool from another url
  -b REPOREPO_BRANCH  Checkout different branch of repo tool
  -x XML              Use a local XML manifest
  -u URL              Manifest git url
  -p PLATFORM         Processor platform
  -r RELEASE          Release version
```

执行 *init* 命令后，phyLinux 将打印一些重要说明以及后续构建步骤的信息。

## 11.4 高级用法

phyLinux 可用于通过多种媒介传输软件状态。软件状态在 *manifest.xml* 有特定标识。您可以创建一个清单文件，将其发送到另一个地方，然后使用以下方式来恢复软件状态，得到目标软件版本

```
host:~$ ./phyLinux init -x manifest.xml
```

您还可以创建一个包含软件状态的 *Git* 仓库。该 *Git* 仓库需要有除了 master 以外的分支，因为我们保留了 master 分支用于其他用途。使用 phyLinux 检查软件状态

```
host:~$ ./phyLinux -u <url-of-your-git-repo>
```

CHAPTER 12

# 使用编译容器

**警告**

目前，无法在容器内运行 phyLinux 脚本。在主机上使用 phyLinux 脚本完成初始化后，您可以使用容器进行编译。如果您的机器上没有 phyLinux 脚本，请参阅 phyLinux 文档。

运行编译容器有多种实现方式。常用的是 docker 和 podman，但我们更喜欢 podman，因为它不需要 root 权限即可运行。

## 12.1 安装

如何安装 podman：https://podman.io 如何安装 docker：https://docs.docker.com/engine/install/

## 12.2 可用容器

目前我们基于 Ubuntu LTS 版本提供了 4 个不同版本的容器：https://hub.docker.com/u/phybuilder

- yocto-ubuntu-16.04
- yocto-ubuntu-18.04
- yocto-ubuntu-20.04
- yocto-ubuntu-22.04

这些容器可以使用 podman 或 docker 运行。对于 Yocto Mickledore 版本，容器 “yocto-ubuntu-20.04” 是首选。

## 12.3 下载/拉取容器

```
host:~$ podman pull docker.io/phybuilder/yocto-ubuntu-20.04

OR

host:~$ docker pull docker.io/phybuilder/yocto-ubuntu-20.04
```

在末尾添加以冒号分隔的容器标签，您还可以拉取或运行带有特殊标签的容器。

podman pull docker.io/phybuilder/yocto-ubuntu-20.04:phy2

您可以在我们的 duckerhub 空间中找到所有可用的标签：

- https://hub.docker.com/r/phybuilder/yocto-ubuntu-16.04/tags
- https://hub.docker.com/r/phybuilder/yocto-ubuntu-18.04/tags
- https://hub.docker.com/r/phybuilder/yocto-ubuntu-20.04/tags
- https://hub.docker.com/r/phybuilder/yocto-ubuntu-22.04/tags

如果您尝试运行尚未拉取/下载的容器，它将被自动拉取/下载。

您可以使用以下命令查看所有已下载/拉取的容器：

```
$USERNAME@$HOSTNAME:~$ podman images
REPOSITORY                                TAG         IMAGE ID      CREATED       SIZE
docker.io/phybuilder/yocto-ubuntu-22.04   latest      d626178e448d  4 months ago  935 MB
docker.io/phybuilder/yocto-ubuntu-22.04   phy2        d626178e448d  4 months ago  935 MB
docker.io/phybuilder/yocto-ubuntu-20.04   phy2        e29a88b7172a  4 months ago  900 MB
docker.io/phybuilder/yocto-ubuntu-20.04   latest      e29a88b7172a  4 months ago  900 MB
docker.io/phybuilder/yocto-ubuntu-18.04   phy1        14c9c3e477d4  7 months ago  567 MB
docker.io/phybuilder/yocto-ubuntu-18.04   latest      14c9c3e477d4  7 months ago  567 MB
docker.io/phybuilder/yocto-ubuntu-16.04   phy1        28c73e13ab4f  7 months ago  599 MB
docker.io/phybuilder/yocto-ubuntu-16.04   latest      28c73e13ab4f  7 months ago  599 MB
docker.io/phybuilder/yocto-ubuntu-22.04   phy1        5a0ef4b41935  8 months ago  627 MB
docker.io/phybuilder/yocto-ubuntu-20.04   phy1        b5a26a86c39f  8 months ago  680 MB
```

## 12.4 运行容器

要运行并使用容器进行 Yocto 构建，首先进入您之前运行 phyLinux init 的文件夹。然后启动容器

```
host:~$ podman run --rm=true -v /home:/home --userns=keep-id --workdir=$PWD -it docker.io/phybuilder/yocto-
↪ubuntu-20.04 bash
```

**备注**

要使用 docker 运行和使用容器，并不像使用 podman 那么简单，必须先定义和配置容器用户。此外，默认情况下 docker 不支持转发信任凭据，所以必须对信任凭据进行配置。

现在您的命令行看起来应该是这样的（其中 $USERNAME 是调用 “podman run” 的用户，并且每次启动容器时容器代码都会有所不同）

```
$USERNAME@6593e2c7b8f6:~$
```

**警告**

如果进入容器的用户名是“root”，您将无法运行 bitbake。请确保使用您自己的用户名运行容器。

现在您可以在容器中开始构建。要停止/关闭容器，只需调用

```
$USERNAME@6593e2c7b8f6:~$ exit
```

CHAPTER 13

# 使用 Poky 和 Bitbake

## 13.1 开始构建

使用 phyLinux init 下载所有元数据后，您必须设置 shell 环境变量。每次打开新 shell 开始构建时都需要执行此操作。我们使用 *Poky* 默认提供的 shell 脚本。在项目的根目录输入

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

source 命令的缩写是一个.

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

shell 的当前工作目录会被改为 *build/*。在第一次构建之前，您应该查看主配置文件

```
host:~$ vim conf/local.conf
```

您当前构建的所有产物存储在这里。根据您使用的芯片平台，可能需要接受许可协议。例如，对于 Freescale/NXP 的芯片平台，您需要接受 GPU 和 VPU 二进制许可协议。通过取消注释相应的行来接受许可协议

```
# Uncomment to accept NXP EULA # EULA can be found under
../sources/meta-freescale/EULA ACCEPT_FSL_EULA = "1"
```

现在您可以构建第一个镜像了。我们建议从较小的非图形镜像 *phytec-headless-image* 开始，看看一切是否正常工作

```
host:~$ bitbake phytec-headless-image
```

在 Intel Core i7 上，首次编译大约需要 40 分钟。但是所有的后续构建都将复用之前的缓存，编译大约需要 3 分钟。

## 13.2 镜像

如果一切顺利，可以在以下位置找到镜像

```
host:~$ cd deploy/images/<MACHINE>
```

测试镜像最简单的方法是将核心板配置为 SD 卡启动，并将构建所得镜像烧写到 SD 卡中

```
host:~$ sudo dd if=phytec-headless-image-<MACHINE>.wic of=/dev/<your_device> bs=1M conv=fsync
```

这里 <your_device> 可以是 “sde”，具体取决于您的系统。选择硬盘驱动时要非常小心！选择错误的硬盘驱动可能会擦除您的硬盘！参数 conv=fsync 强制数据缓冲区在 dd 返回之前写入硬盘。

启动后，您可以使用串口或通过网口 *ssh* 登录。root 账户没有密码。这是因为 *conf/local.conf* 中开启了调试模式。如果您取消注释该行

```
#EXTRA_IMAGE_FEATURES = "debug-tweaks"
```

调试模式（例如设置空的 root 密码）将不会生效。

## 13.3 获取 BSP 长期维护版本之间的中间开发版本

您也可以访问 *Yocto* BSP 最新的开发中版本，这属于特殊版本，并不是正式的长期维护版本，所以它们不会显示在 phyLinux 选择菜单中，需要手动选择。可以使用以下命令行来获取 BSP

```
host:~$ ./phyLinux init -p master -r mickledore
```

这将初始化一个最新的 BSP 开发版本。运行

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

该文件夹将从我们的 Git 仓库中提取所有最新更改。

## 13.4 检查您的编译配置

*Poky* 包含多个工具来检查您的 Yocto 工程。您可以查询 Layer 工具支持的指令

```
host:~$ bitbake-layers
```

例如，它可以用来查看特定 Recipe 在哪一个 Layer 被修改了

```
host:~$ bitbake-layers show-appends
```

在运行构建之前，您还可以启动 *Toaster*，以便能够使用 Toaster Web GUI 图形界面检查构建的详细信息

```
host:~$ source toaster start
```

但是您可能需要先安装一些依赖才能运行 toaster

```
host:~$ pip3 install -r
../sources/poky/bitbake/toaster-requirements.txt
```

然后，您可以将浏览器指向 *http://0.0.0.0:8000/* 并继续使用 *Bitbake*。可以从此 Web 服务器监控和分析所有构建活动。如果您想了解有关 *Toaster* 的更多信息，请查看 https://docs.yoctoproject.org/4.2.4/toaster-manual/index.html。要关闭 *Toaster* Web GUI，请执行

```
host:~$ source toaster stop
```

## 13.5 meta-phytec 和 meta-ampliphy 的特点

### 13.5.1 *Buildinfo*

*Buildinfo* 任务是我们 Recipe 中的一个函数，可打印从公共仓库获取源代码的过程。因此您不必亲自查看对应的 Recipe，用 *Buildinfo* 即可查看过程说明（例如 *barebox* 包的说明），请执行

```
host:~$ bitbake barebox -c buildinfo
```

在您的 shell 中，会打印如下类似信息

```
(mini) HOWTO: Use a local git repository to build barebox:

To get source code for this package and version (barebox-2022.02.0-phy1), execute

$ mkdir -p ~/git
$ cd ~/git
$ git clone git://git.phytec.de/barebox barebox
$ cd ~/git/barebox
$ git switch --create v2022.02.0-phy1-local-development 7fe12e65d770f7e657e683849681f339a996418b

You now have two possible workflows for your changes:

1. Work inside the git repository:
Copy and paste the following snippet to your "local.conf":

SRC_URI:pn-barebox = "git://${HOME}/git/barebox;branch=${BRANCH}"
SRCREV:pn-barebox = "${AUTOREV}"
BRANCH:pn-barebox = "v2022.02.0-phy1-local-development"

After that you can recompile and deploy the package with

$ bitbake barebox -c compile
$ bitbake barebox -c deploy

Note: You have to commit all your changes. Otherwise yocto doesn't pick them up!

2. Work and compile from the local working directory
To work and compile in an external source directory we provide the
externalsrc.bbclass. To use it, copy and paste the following snippet to your
"local.conf":

INHERIT += "externalsrc"
EXTERNALSRC:pn-barebox = "${HOME}/git/barebox"
EXTERNALSRC_BUILD:pn-barebox = "${HOME}/git/barebox"

Note: All the compiling is done in the EXTERNALSRC directory. Every time
you build an Image, the package will be recompiled and build.

NOTE: Tasks Summary: Attempted 1 tasks of which 0 didn't need to be rerun and all succeeded.
NOTE: Writing buildhistory
```

正如您所见，控制台输出了清楚地说明了构建信息。

**警告**

使用 外部源会破坏 *Yocto* 的许多内部依赖机制。在构建过程中，源目录下的更改并不能保证被自动抓取并合并到根文件系统或 SD 卡镜像中。您必须始终使用 *--force*。例如，在编译 *barebox* 并将其重新部署到 *deploy/images/<machine>* 时需要执行

```
host:~$ bitbake barebox -c compile --force
host:~$ bitbake barebox -c deploy
```

要使用新内核或镜像更新 SD 卡镜像，首先强制编译它，然后强制重建根文件系统。使用

```
host:~$ bitbake phytec-qt6demo-image -c rootfs --force
```

请注意，Yocto 构建系统不会对外部源文件目录进行修改。如果要将 *Yocto* Recipe 携带的所有补丁应用到外部源目录，请运行以下指令

```
SRCTREECOVEREDTASKS="" BB_ENV_PASSTHROUGH_ADDITIONS="$BB_ENV_PASSTHROUGH_ADDITIONS SRCTREECOVEREDTASKS"␣
↪bitbake <recipe> -c patch
```

## 13.6 自定义 BSP

为了帮助您开始使用 BSP，我们从 *Yocto* 官方文档中总结了一些基本任务。它描述了如何向镜像添加其他软件、更改 kernel 和 Bootloader 配置，以及应用 kernel 和 Bootloader 的补丁。

诸如添加软件之类的小修改是在文件 *build/conf/local.conf* 中完成的。在那里，您可以覆盖全局变量并对 recipe 进行小修改。

有两种方法可以进行重大更改：

1. 创建您自己的 Layer 并使用 *bbappend* 文件。
2. 将所有新增内容添加到 PHYTEC 的 Distro Layer *meta-ampliphy*。

创建您自己的 Layer 部分描述了如何创建您自己的 Layer。

### 13.6.1 禁用 Qt 示例 demo

默认情况下，BSP 映像 *phytec-qt6demo-image* 会在连接的显示器或监视器上启动 Qt6 示例应用程序。如果要停止示例并使用后台得 *Linux* framebuffer 控制台，请通过串口或网络 *ssh* 连接到目标并执行 shell 命令

```
target:~$ systemctl stop phytec-qtdemo.service
```

此命令暂时停止演示 app。要重新启动，请重启主板或执行

```
target:~$ systemctl start phytec-qtdemo.service
```

您可以永久禁用该服务，这样它就不会在开机时自启动

```
target:~$ systemctl disable phytec-qtdemo.service
```

**小技巧**

最后一个命令仅禁用了服务，但是它不会立即停止。要查看当前服务状态，请执行

```
target:~$ systemctl status phytec-qtdemo.service
```

如果要默认禁用该服务，请编辑文件 *build/conf/local.conf* 并添加以下行

```
# file build/conf/local.conf
SYSTEMD_AUTO_ENABLE:pn-phytec-qtdemo = "disable"
```

之后重新编译镜像

```
host:~$ bitbake phytec-qt6demo-image
```

### 13.6.2 Framebuffer 控制台

在具有显示接口的核心板上，默认启用 Framebuffer 控制台。您可以连接 USB 键盘并登录。要将键盘布局从默认的英语更改为德语，请键入

```
target:~$ loadkeys /usr/share/keymaps/i386/qwertz/de-latin1.map.gz
```

如果要退出 Framebuffer 控制台，请运行

```
target:~$ echo 0 > sys/class/vtconsole/vtcon1/bind
```

要完全停用 Framebuffer 控制台，请禁用以下内核配置选项

```
Device Drivers->Graphics Support->Support for framebuffer devices->Framebuffer Console Support
```

更多信息请访问：https://www.kernel.org/doc/Documentation/fb/fbcon.txt

### 13.6.3 预编译镜像中提供的工具

#### RAM 基准测试

RAM 和缓存性能测试的最佳方法是使用 *pmbw* （并行内存带宽基准测试/测量工具）。*Pmbw* 运行多个汇编例程，这些例程都使用不同的访问模式来访问 SoC 的缓存和 RAM。在运行测试之前，请确保设备上有大约 2 MiB 的空间用于存储日志文件。我们还降低了基准测试的级别，以便于更积极地向内核请求资源。基准测试将花费几个小时。

要开始测试，请键入

```
target:~$ nice -n -2 pmbw
```

测试完成后，日志文件可以转换为 *gnuplot* 脚本，运行

```
target:~$ stats2gnuplot stats.txt > run1.gnuplot
```

现在您可以将日志文件传输到主机并安装任意版本的 *gnuplot*

```
host:~$ sudo apt-get install gnuplot host:~$ gnuplot run1.gnuplot
```

生成的 *plots-<machine> .pdf* 文件包含所有图表。要将单个图表渲染为 *png* 文件，您可以使用 *Ghostscript*

```
host:~$ sudo apt-get install ghostscript
host:~$ gs -dNOPAUSE -dBATCH -sDEVICE=png16m -r150 -sOutputFile='page-%00d.png' plots-phyboard-wega-am335x-
↪1.pdf
```

### 13.6.4 为 BSP 镜像添加新的软件包

要向镜像添加其他软件，请查看 OpenEmbedded Layer 索引：https://layers.openembedded.org/layerindex/branch/mickledore/layers/

首先，从左上角的下拉列表中选择您拥有的 BSP 的 *Yocto* 版本，然后单击 **Recipes**。现在您可以搜索软件项目名称并找到它所在的 Layer。在某些情况下，程序位于 *meta-openembedded*、*openembedded-core* 或 *Poky* 中，这意味着 Recipe 已在您的 Yocto 工程中。本节介绍在这种情况下如何添加软件。如果软件包位于另一个 Layer，请参阅下一部分。

您还可以搜索可用 Recipe 列表

```
host:~$ bitbake -s | grep <program name> # fill in program name, like in
host:~$ bitbake -s | grep lsof
```

当程序的 Recipe 已在 *Yocto* 工程中时，您只需将他添加到文件 *build/conf/local.conf* 即可。向镜像添加其他软件的一般语法是

```
# file build/conf/local.conf
IMAGE_INSTALL:append = " <package1> <package2>"
```

例如，这一行

```
# file build/conf/local.conf
IMAGE_INSTALL:append = " ldd strace file lsof"
```

在目标镜像上安装一些辅助程序。

**警告**

程序包名称前的空格非常重要。

local.conf 中的配置项均适用于所有镜像。因此，新增的这些软件工具将被同时包含在 phytec-headless-image 和 phytec-qt6demo-image 镜像中。

#### 关于软件包和 Recipe 的说明

您正在将软件包添加到 IMAGE_INSTALL 变量中。这些不一定等同于 Layer 的 Recipe 名称。Recipe 默认定义一个具有相同名称的软件包。但 Recipe 也可以将 PACKAGES 变量设置为其他名称，并能够生成具有任意名称的软件包。当您查找软件时，严格来说，都必须搜索软件包名称，而不是 Recipe。在最坏的情况下，您必须查看所有 PACKAGES 变量。这会有点繁琐，*Toaster* 之类的工具可能对查找有所帮助。

如果您在文件夹 *sources* 提供的 Layer 中找不到您的软件，请参阅下一部分以将一个新的 Layer 包含到 *Yocto* 构建中。

参考资料：Yocto 4.2.4 文档 - 自定义 Yocto 构建

### 13.6.5 添加其他 Layer

这是关于如何在您的 *Yocto* 构建中添加另一 Layer 并从中安装软件的详细上手指南。例如，我们将网络安全扫描器 *nmap* 包含在 Layer *meta-security* 中。首先，您必须找到包含该软件的 Layer。查看 OpenEmbedded MetaData 索引 并稍微思考一下。网络扫描器 *nmap* 位于 *meta-security* Layer。请参阅 layer.openembedded.org 上的 meta-security。要将其集成到 *Yocto* 构建中，您必须拉取 git 仓库，然后切换到正确的稳定分支。由于 BSP 是基于 *Yocto* “sumo” 版本构建，因此您也应该尝试在 Layer 中使用 “sumo” 分支。

```
host:~$ cd sources
host:~$ git clone git://git.yoctoproject.org/meta-security
host:~$ cd meta-security
host:~$ git branch -r
```

所有可用的远程分支都会显示出来。通常应该有“fido”、“jethro”、“krogoth”、“master” ……

```
host:~$ git checkout mickledore
```

现在我们通过把以下代码添加到 *build/conf/bblayers.conf* 文件末尾的方式来将 Layer 包含到构建中

```
# file build/conf/bblayers.conf
BBLAYERS += "${BSPDIR}/sources/meta-security"
```

然后，您可以通过执行以下代码来检查该 Layer 是否在构建配置中可用

```
host:~$ bitbake-layers show-layers
```

如果出现类似以下错误

```
ERROR: Layer 'security' depends on layer 'perl-layer', but this layer is not enabled in your configuration
```

如果您要添加的 Layer（这里是 *meta-security*）依赖于另一个 Layer，您需要先启用被依赖的 Layer。例如，此处所需的依赖项是 *meta-openembedded* 中的 Layer（在 PHYTEC BSP 中，它位于路径 *sources/meta-openembedded/meta-perl/* 中）。要启用它，请将以下行添加到 *build/conf/bblayers.conf*

```
# file build/conf/bblayers.conf
BBLAYERS += "${BSPDIR}/sources/meta-openembedded/meta-perl"
```

执行命令 *bitbake-layers show-layers* 应该会打印所有已启用 Layer 的列表，包括 *meta-security* 和 *meta-perl*。包含该 Layer 后，您可以按照之前的描述安装 Layer 中的软件包。最简单的方法是添加以下行（这里是包 *nmap*）

```
# file build/conf/local.conf
IMAGE_INSTALL:append = " nmap"
```

到您的 *build/conf/local.conf*。不要添加后忘记重新构建镜像

```
host:~$ bitbake phytec-qt6demo-image
```

### 13.6.6 Create your own layer

创建 Layer 应该是自定义 BSP 的首要任务之一。您有两个选择。您可以复制并重命名 *meta-ampliphy*，也可以创建一个包含修改的新 Layer。哪个选择更好取决于您的使用场景。*meta-ampliphy* 是我们自定义 *Linux* 发行版的示例，该发行版也会在未来持续更新。如果您想从这些更新中受益，并且总体上对它的用户空间配置感到满意，那么在 *Ampliphy* 基础上创建自己的 Layer 可能是最佳解决方案。如果您需要重建用户空间架构并且只需要 PHYTEC 的基本硬件支持，最好复制 *meta-ampliphy*，给 Layer 重新命名并修改其内容以使其适应您的需求。您还可以查看 OpenEmbedded Layer 索引以查找不同的发行版 Layer。如果您只需要将自己的应用程序添加到镜像中，请创建自己的 Layer。

在下一章中，我们有一个名为“racer”的嵌入式项目，我们将使用我们的 *Ampliphy Linux* 发行版来集成它。首先，我们需要创建一个新的 Layer。

*Yocto* 提供了一个脚本来实现 Layer 创建。如果您已经配置好 BSP 并且 shell 已准备就绪，请输入

```
host:~$ bitbake-layers create-layer meta-racer
```

目前来说，默认选项是可行的。将该 Layer 移动到 source 目录下

```
host:~$ mv meta-racer ../sources/
```

在此 Layer 创建一个 *Git* 仓库来跟踪您的更改

```
host:~$ cd ../sources/meta-racer
host:~$ git init && git add . && git commit -s
```

现在您可以将该 Layer 直接添加到 build/conf/bblayers.conf

```
BBLAYERS += "${BSPDIR}/sources/meta-racer"
```

或者使用 *Yocto* 提供的脚本

```
host:~$ bitbake-layers add-layer meta-racer
```

### 13.6.7 kernel 和 Bootloader 的 Recipe 和版本

首先，您需要知道目标 Machine 使用哪个 kernel 和对应的版本。PHYTEC 提供多个 kernel recipe *linux-mainline*、*linux-ti* 和 *linux-imx*。第一个为 PHYTEC 的 i.MX 6 和 AM335x SOM 提供支持，他是基于 kernel.org 的 *Linux* kernel 稳定版本。*Git* 仓库 URL 为：

- *linux-mainline*：git://git.phytec.de/linux-mainline
- *linux-ti*: git://git.phytec.de/linux-ti
- *linux-imx:* git://git.phytec.de/linux-imx
- *barebox*：git://git.phytec.de/barebox
- *u-boot-imx*: git://git.phytec.de/u-boot-imx

要找出您最终所使用的 kernel recipe，请执行以下命令

```
host:~$ bitbake virtual/kernel -e | grep "PREFERRED_PROVIDER_virtual/kernel"
```

该命令打印变量 *PREFERRED_PROVIDER_virtual/kernel* 的值。该变量在 *Yocto* 构建过程被用来选择要使用的 kernel recipe。以下几行是您可能会看到的不同输出值

```
PREFERRED_PROVIDER_virtual/kernel="linux-mainline"
PREFERRED_PROVIDER_virtual/kernel="linux-ti"
PREFERRED_PROVIDER_virtual/kernel="linux-imx"
```

要查看使用的是哪个版本，请执行 *bitbake -s*。例如

```
host:~$ bitbake -s | egrep -e "linux-mainline|linux-ti|linux-imx|barebox|u-boot-imx"
```

参数 *-s* 打印所有相关 Recipe 和它的版本。输出左侧包含 Recipe 名称，右侧包含版本

```
barebox                     :2022.02.0-phy1-r7.0
barebox-hosttools-native    :2022.02.0-phy1-r7.0
barebox-targettools         :2022.02.0-phy1-r7.0
linux-mainline              :5.15.102-phy1-r0.0
```

如您所见，recipe *linux-mainline* 的版本为 *5.15.102-phy1*。在 PHYTEC 的 *linux-mainline Git* 仓库中，您将找到相应的标签 *v5.15.102-phy1*。*barebox* recipe 的版本为 2022.02.0-phy1。在 i.MX8M* 模块上，输出将包含 *linux-imx* 和 *u-boot-imx*。

### 13.6.8 Kernel 和 Bootloader 配置

PHYTEC 所使用的 bootloader *barebox* 采用与 *Linux* kernel 相同的编译系统。因此，本节中的所有指令均可用于配置 kernel 以及 bootloader。要配置 kernel 或 bootloader，请执行以下命令中的一条。

```
host:~$ bitbake -c menuconfig virtual/kernel  # Using the virtual provider name
host:~$ bitbake -c menuconfig linux-ti        # Or use the recipe name directly
host:~$ bitbake -c menuconfig linux-mainline  # Or use the recipe name directly (If you use an i.MX 6 or␣
↪RK3288 Module)
host:~$ bitbake -c menuconfig linux-imx       # Or use the recipe name directly (If you use an i.MX 8M*)
host:~$ bitbake -c menuconfig barebox         # Or change the configuration of the bootloader
host:~$ bitbake -c menuconfig u-boot-imx      # Or change the configuration of the bootloader (If you use␣
↪an i.MX 8M*)
```

之后，您可以重新编译并部署 kernel 或 bootloader

```
host:~$ bitbake virtual/kernel -c compile  # Or 'barebox' for the bootloader
host:~$ bitbake virtual/kernel -c deploy   # Or 'barebox' for the bootloader
```

或者，您也可以使用以下命令重新进行完整的构建

```
host:~$ bitbake phytec-headless-image  # To update the kernel/bootloader, modules and the images
```

在最后一个命令中，您可以将镜像名称替换为您选择的镜像名称。新镜像和二进制文件位于 *build/deploy/images/<machine> /*.

**警告**

之前对 kernel 的配置并不是永久生效的。执行 *bitbake virtual/kernel -c clean* 可以清除所有内容。

要使更改在构建系统中永久生效，您需要将配置修改整合到 Layer 中。您有两种选择：

- 仅包含配置差异片段（新旧配置之间的最小 *差异*）
- 修改后的完整配置（*defconfig*）。

使用配置片段可以使开发者对不同阶段所做的更改更加清晰。您可以选择启用或禁用这些配置，对不同条件下的配置作管理，这有助于更迅速地将更改的配置迁移到新的 kernel 版本。您还可以对配置片段进行分组，以在不同的特定场景下使用。生成的完整 kernel 配置将被放在目录 *build/deploy/images/<machine>* 中。如果您没有上述这些需求，选择维护完整的 *defconfig* 文件可能会更加简单。

#### 将配置片段添加到 Recipe

以下步骤适用于 kernel 和 bootloader。只需将命令中的 recipe 名称 *linux-mainline* 替换为 *linux-ti*，或者将 bootloader 替换为 *barebox*。如果您对这个命令作用还不熟悉，请从一个干净的 Yocto 工程重新开始。否则，配置的差异可能会导致错误。

```
host:~$ bitbake linux-mainline -c clean
host:~$ bitbake linux-mainline -c menuconfig
```

在菜单中更改配置并生成配置片段

```
host:~$ bitbake linux-mainline -c diffconfig
```

他将会打印生成的配置片段文件路径

```
Config fragment has been dumped into:
/home/<path>/build/tmp/work/phyboard_mira_imx6_11-phytec-linux-gnueabi/linux-mainline/4.19.100-phy1-r0.0/
↪fragment.cfg
```

所有的配置修改都记录在文件 *fragment.cfg* 中，该文件仅包含少量配置行。以下示例展示了如何创建 *bbappend* 文件，以及如何为配置片段添加所需的行。您只需根据特定 Recipe 调整目录和名称：*linux-mainline*、*linux-ti*、linux-imx、u-boot-imx 或 *barebox*。

```
sources/<layer>/recipes-kernel/linux/linux-mainline_%.bbappend     # For the recipe linux-mainline
sources/<layer>/recipes-kernel/linux/linux-ti_%.bbappend           # For the recipe linux-ti
sources/<layer>/recipes-kernel/linux/linux-imx_%.bbappend          # For the recipe linux-imx
sources/<layer>/recipes-bsp/barebox/barebox_%.bbappend             # For the recipe barebox
sources/<layer>/recipes-bsp/u-boot/u-boot-imx_%.bbappend           # For the recipe u-boot-imx
```

将字符串 *layer* 替换为您自己创建的 layer（如上所示）（例如 *meta-racer*），或者直接使用 *meta-ampliphy*。要使用 *meta-ampliphy*，首先，需要为配置片段创建目录并为其指定一个新名称（此处为 *enable-r8169.cfg*），然后将片段放到这个 Layer 中。

```
host:~$ mkdir -p sources/meta-ampliphy/recipes-kernel/linux/features
# copy the path from the output of *diffconfig*
host:~$ cp /home/<path>/build/tmp/work/phyboard_mira_imx6_11-phytec-linux-gnueabi/linux-mainline/4.19.100-
↪phy1-r0.0/fragment.cfg \
    sources/meta-ampliphy/recipes-kernel/linux/features/enable-r8169.cfg
```

之后使用您喜欢的编辑器打开 *bbappend* 文件（在本例中为 *sources/meta-ampliphy/recipes-kernel/linux/linux-mainline_%.bbappend*）并添加以下几行

```
# contents of the file linux-mainline_%.bbappend
FILESEXTRAPATHS:prepend := "${THISDIR}/features:"
SRC_URI:append = " \
    file://enable-r8169.cfg \
"
```

**警告**

不要忘记使用正确的 *bbappend* 文件名：*linux-ti_%.bbappend* 用于 linux-ti recipe，*barebox_%.bbappend* 用于文件夹 *recipes-bsp/barebox/* 中的 bootloader!

保存 *bbappend* 文件后，您需要重新编译镜像。*Yocto* 会自动识别 recipe 的更改并生成新的镜像。

```
host:~$ bitbake phytec-headless-image # Or another image name
```

### 向 Recipe 添加一个完整配置 (*defconfig*)

这种方法与之前的方法相似，但不是添加一个片段，而是采用 *defconfig*。首先，在您想要使用的 Layer 中创建所需的文件夹，这可以是您自己的 Layer，或者是 *meta-ampliphy*。

```
host:~$ mkdir -p sources/meta-ampliphy/recipes-kernel/linux/features/ # For both linux-mainline and linux-ti
host:~$ mkdir -p sources/meta-ampliphy/recipes-bsp/barebox/features/ # Or for the bootloader
```

然后您需要创建一个合适的 *defconfig* 文件。使用 *menuconfig* 进行配置更改，然后将 *defconfig* 文件保存到 Layer 中。

```
host:~$ bitbake linux-mainline -c menuconfig # Or use recipe name linux-ti or barebox
host:~$ bitbake linux-mainline -c savedefconfig # Create file 'defconfig.temp' in the work directory
```

这将打印 defconfig 的保存路径

```
Saving defconfig to ..../defconfig.temp
```

然后，按照前面的说明，将生成的完整配置文件复制到您的 Layer 中，重命名为 *defconfig*，并在 *bbappend* 文件中添加以下行（此处为 *sources/meta-ampliphy/recipes-kernel/linux/linux-mainline_%.bbappend*）。

```
# contents of the file linux-mainline_%.bbappend
FILESEXTRAPATHS:prepend := "${THISDIR}/features:"
SRC_URI:append = " \
    file://defconfig \
"
```

**小技巧**

不要忘记使用正确的 bbappend 文件名：*linux-ti_%.bbappend* 用于 linux-ti recipe ，*barebox_%.bbappend* 用于文件夹 *recipes-bsp/barebox/* 中的 bootloader ！

这样，在重编镜像时，新的镜像将会包含所作的配置修改。

```
host:~$ bitbake phytec-headless-image # Or another image name
```

### 13.6.9 使用 *devtool* 修改 Kernel 或 Bootloader

除了使用 *recipes* 中提供的标准 *kernel* 和 *bootloader* 外，您可以选择在 *yocto* 中用 *devtool* 直接修改源代码或单独下载我们的 *git* 仓库来构建您的自定义 *kernel* 。以下是两种修改方式的优劣对比

| 优势 | 劣势 |
|---|---|
| *Yocto* 官方文档的标准工作流程 | 重复的源文件包，造成了不必要的硬盘空间浪费。 |
| 工具链无需重新编译所有内容。 | 未能有效利用缓存，导致构建成本增加。 |

*Devtool* 是一套辅助工具，旨在提升 *Yocto* 用户的工作效率。它与 1.8 版本相兼容。只需配置好 shell 环境，您就可以使用它。*Devtool* 的功能包括：

- 修改现有资源文件
- 将其他软件项目纳入您的构建配置中
- 编译软件并将修改后的软件部署到目标硬件中。

这里我们将利用 *devtool* 来修改 kernel。我们以 *linux-mainline* 作为 AM335x kernel 的实现示例。我们首先使用的命令是 *devtool modify - x <recipe><directory>*

```
host:~$ devtool modify -x linux-mainline linux-mainline
```

*Devtool* 将在 *build/workspace* 目录下创建一个 layer ，您可以在这个 Layer 下查看 *devtool* 命令进行的所有更改。它将与 recipe 相关的源代码下载到一个指定的文件夹中以及在 workspace 生成一个 *bbappend* 文件，其中的 SRC_URI 指向上述下载文件夹。使用 *Bitbake* 构建镜像时，将会使用此文件夹中的源代码。现在您可以对 kernel 进行修改。

```
host:~$ vim linux-mainline/arch/arm/boot/dts/am335x-phycore-som.dtsi
      -> make a change
host:~$ bitbake phytec-qt6demo-image
```

您的更改现在将被重新编译并添加到镜像中。如果您希望永久保存这些更改，建议您根据更改创建一个补丁，存储和备份该补丁。您可以进入 *linux-mainline* 目录并使用 *Git* 来创建补丁。有关如何创建补丁的详细信息，请参阅使用“临时方法”修补 *Kernel* 或 *Bootloader*。

如果您想了解有关 *devtool* 的更多信息，请访问：

Yocto 4.2.4 - Devtool 或 Devtool 快速指南

### 13.6.10 使用“临时方法”修补 Kernel 或 Bootloader

| 优势 | 劣势 |
| --- | --- |
| 无开销，无额外配置 | *Yocto* 非常容易覆盖您所作的修改（所有内容都会丢失!!）。 |
| 工具链无需重新编译所有内容。 | |

Yocto 允许在 *Bitbake* 配置和编译 recipe 之前，更改源代码。使用 *Bitbake* 的 *devshell* 命令跳转到 recipe 的源目录。这是 *barebox* recipe

```
host:~$ bitbake barebox -c devshell # or linux-mainline, linux-ti, linux-imx, u-boot-imx
```

执行命令后，将打开一个 shell 窗口。shell 的当前工作目录将更改为 *tmp* 文件夹中 recipe 的资源文件目录。在这里，您可以使用您最喜欢的编辑器（例如 *vim*、*emacs* 或任何其他图形编辑器）来更改源代码。完成后，通过键入 *exit* 或按 **CTRL-D** 退出 *devshell*。

离开 *devshell* 后，您可以重新编译软件包

```
host:~$ bitbake barebox -c compile --force # or linux-mainline, linux-ti, linux-imx, u-boot-imx
```

参数“--force”很重要，因为 *Yocto* 无法识别源代码是否已被更改。

**小技巧**

您无法在 *devshell* 中执行 *bitbake* 命令。您必须先退出 devshell 模式。

如果构建失败，请再次执行 devshell 命令并修复。如果构建成功，则可以部署包并创建新的 SD 卡镜像

```
host:~$ bitbake barebox -c deploy # new barebox in e.g. deploy/images/phyflex-imx6-2/barebox.bin
host:~$ bitbake phytec-headless-image # new WIC image in e.g. deploy/images/phyflex-imx6-2/phytec-headless-
↪image-phyflex-imx6-2.wic
```

**警告**

如果您进行 clean 操作，比如 *bitbake barebox -c clean*，或者 *Yocto* 重新下载源码，您所做的所有更改将会丢失!!!

为了防止这种情况发生，您可以制作一个补丁并将其添加到 *bbappend* 文件中。这与配置修改章节中提到的工作流程相似。

如果您采用临时方法，让修改永久生效需要在 *devshell* 中生成补丁；如果您使用 *devtool*，则必须在 *devtool* 创建的子目录中生成补丁。

```
host:~$ bitbake barebox -c devshell            # Or linux-mainline, linux-ti
host(devshell):~$ git status                   # Show changes files
host(devshell):~$ git add <file>               # Add a special file to the staging area
host(devshell):~$ git commit -m "important modification"   # Creates a commit with a not so useful commit␣
↪message
host(devshell):~$ git format-patch -1 -o ~/    # Creates a patch of the last commit and saves it in your␣
↪home folder
/home/<user>/0001-important-modification.patch  # Git prints the path of the written patch file
host(devshell):~$ exit
```

创建补丁后，必须为其创建一个 *bbappend* 文件。三个不同 recipe 文件（*linux-mainline*、*linux-ti* 和 *barebox*）的位置如下：

```
sources/<layer>/recipes-kernel/linux/linux-mainline_%.bbappend     # For the recipe linux-mainline
sources/<layer>/recipes-kernel/linux/linux-ti_%.bbappend           # For the recipe linux-ti
sources/<layer>/recipes-kernel/linux/linux-imx_%.bbappend        # For the recipe linux-imx
sources/<layer>/recipes-bsp/barebox/barebox_%.bbappend             # For the recipe barebox
sources/<layer>/recipes-bsp/u-boot/u-boot-imx_%.bbappend           # For the recipe u-boot-imx
```

以下示例适用于 recipe *barebox*。但是必须根据您的实际情况调整补丁路径。首先，创建文件夹并将补丁移入其中。然后创建 *bbappend* 文件

```
host:~$ mkdir -p sources/meta-ampliphy/recipes-bsp/barebox/features   # Or use your own layer instead of␣
↪*meta-ampliphy*
host:~$ cp ~/0001-important-modification.patch sources/meta-ampliphy/recipes-bsp/barebox/features  # copy␣
↪patch
host:~$ touch sources/meta-ampliphy/recipes-bsp/barebox/barebox_%.bbappend
```

**小技巧**

注意您当前的工作目录。您必须在 BSP 根目录中执行命令，而不是在 *build* 目录中！

之后，使用您最喜欢的编辑器将以下内容添加到 *bbappend* 文件中（此处为 *sources/meta-ampliphy/recipes-bsp/barebox/barebox_%.bbappend*）

```
# contents of the file barebox_%.bbappend
FILESEXTRAPATHS:prepend := "${THISDIR}/features:"
SRC_URI:append = " \
    file://0001-important-modification.patch \
"
```

保存文件并使用以下方法重新编译 *barebox* recipe

```
host:~$ bitbake barebox -c clean # Or linux-ti, linux-mainline, linux-imx, u-boot-imx
host:~$ bitbake barebox
```

如果构建成功，您可以通过以下方法重新生成最终镜像。

```
host:~$ bitbake phytec-headless-image # Or another image name
```

**更多资源：**

*Yocto* 项目为软件开发人员提供了一些文档。请参考“Kernel 开发手册”以获取有关 Kernel 配置的详细信息。请注意，并非所有 *Yocto* 手册中的内容都适用于 PHYTEC BSP，因为我们采用了传统的内核编译方法，而大部分文档会假设使用的是 *Yocto* 的内核编译方式。

- Yocto-Kernel 开发手册
- Yocto-开发手册

### 13.6.11 使用 *local.conf* 文件中的 SRC_URI 来配置 Kernel 和 Bootloader

在这里，我们提供了第三个选项来修改 *kernel* 和 *Bootloader* 。您可以从外部拉取 *linux-mainline*、*linux-ti* 或 *barebox Git* 仓库。您将需要重写源代码 *URL*（变量 *SRC_URI*），以指向您的本地仓库而不是远程仓库。

| 优势 | 劣势 |
|---|---|
| 所有更改均使用 *Git* 保存 | *build/tmp-glibc/work/* 中有许多工作目录 *<machine>/<package>* / |
| | 重新编译之前必须提交所有更改 |
| | 对于每次更改，工具链都会从头开始编译所有内容（可以使用 *ccache* 避免） |

首先，您需要一个 *Git* 仓库 *barebox* 或 kernel 的本地副本。如果您还没有，请使用以下命令。

```
host:~$ mkdir ~/git
host:~$ cd ~/git
host:~$ git clone git://git.phytec.de/barebox
host:~$ cd barebox
host:~$ git switch --create v2022.02.0-phy remotes/origin/v2022.02.0-phy
```

将以下代码片段添加到 build/conf/local.conf

```
# Use your own path to the git repository
# NOTE: Branch name in variable "BRANCH_pn-barebox" should be the same as the
# branch which is used in the repository folder. Otherwise your commits won't be recognized later.
BRANCH:pn-barebox = "v2022.02.0-phy"
SRC_URI:pn-barebox = "git:///${HOME}/git/barebox;branch=${BRANCH}"
SRCREV:pn-barebox = "${AUTOREV}"
```

您需要在文件中正确设置 git 仓库分支名称。无论您是选择在 *Git* 仓库中创建自己的分支，还是使用默认的分支（这里是“v2015.02.0-phy”）。现在，您都可以用自己的源代码重新编译 *barebox*。

```
host:~$ bitbake barebox -c clean
host:~$ bitbake barebox -c compile
```

由于源代码尚未被修改，因此构建应该能够成功。

您可以在 *~/git/barebox* 或默认的 *defconfig* 中修改源代码（例如 *~/git/barebox/arch/arm/configs/imx_v7_defconfig*）。一旦您完成了代码更改，您需要对 *Yocto* 进行本地提交。如果您不这样做，*Yocto* 将无法检测到您在 git 仓库中的源代码已被更改（例如 ~/git/barebox/）。

```
host:~$ git status  # show modified files
host:~$ git diff    # show changed lines
host:~$ git commit -a -m "dummy commit for yocto"   # This command is important!
```

本地提交后尝试编译。*Yocto* 将自动注意到源代码已更改，并从头开始进行源代码获取和工程配置工作。

```
host:~$ bitbake barebox -c compile
```

如果构建失败，请返回源目录，修复问题并重新提交更改。如果构建成功，您可以获取 *barebox*，甚至创建一个新的 SD 卡镜像。.

```
host:~$ bitbake barebox -c deploy # new barebox in e.g. deploy/images/phyflex-imx6-2/barebox-phyflex-imx6-2.
↪bin
```

（续下页）

(接上页)

```
host:~$ bitbake phytec-headless-image # new sd-card image in e.g. deploy/images/phyflex-imx6-2/phytec-
↪headless-image-phyflex-imx6-2.wic
```

如果您想进行其他更改，只需在 git 仓库中进行另一次提交并再次重新编译 *barebox*。

### 13.6.12 使用“可持续方法”添加软件

现在您已经创建了自己的 Layer，您有第二个方式可以将现有软件添加到目标镜像中。我们的标准镜像在 meta-ampliphy 中定义

```
meta-ampliphy/recipes-images/images/phytec-headless-image.bb
```

在您的 layer 中，可以使用 *bbappend* 修改 recipe，无需改动源 recipe 文件

```
meta-racer/recipes-images/images/phytec-headless-image.bbappend
```

bbappend 文件内容将与源 recipe 一起解析。因此，您可以轻松覆盖源 recipe 中设置的所有变量。如果我们想要包含一些其他软件，需要在 bbapend 中将其追加到 IMAGE_INSTALL 变量中

```
IMAGE_INSTALL:append = " rsync"
```

### 13.6.13 将 Linux 固件添加到根文件系统

将额外的固件放入根文件系统的 */lib/firmware/* 目录是一项常见的需求。例如，WiFi 适配器或 PCIe 网卡可能需要专有的固件。为了解决这个问题，我们可以在 Layer 中使用 *bbappend* 。首先要创建所需的文件夹，在文件夹中创建 bbapend 文件并拷贝固件到对应的文件夹中。

```
host:~$ cd meta-racer   # go into your layer
host:~$ mkdir -p recipes-kernel/linux-firmware/linux-firmware/
host:~$ touch recipes-kernel/linux-firmware/linux-firmware_%.bbappend
host:~$ cp ~/example-firmware.bin recipes-kernel/linux-firmware/linux-firmware/    # adapt filename
```

然后将以下内容添加到 *bbappend* 文件中，用您的固件名替换每个出现的 *example-firmware.bin*。

```
# file recipes-kernel/linux-firmware/linux-firmware_%.bbappend

FILESEXTRAPATHS:prepend := "${THISDIR}/linux-firmware:"
SRC_URI += "file://example-firmware.bin"

do_install:append () {
        install -m 0644 ${WORKDIR}/example-firmware.bin ${D}/lib/firmware/example-firmware.bin
}

# NOTE: Use "=+" instead of "+=". Otherwise file is placed into the linux-firmware package.
PACKAGES =+ "${PN}-example"
FILES:${PN}-example = "/lib/firmware/example-firmware.bin"
```

现在尝试构建 linux-firmware recipe

```
host:~$ . sources/poky/oe-init-build-env
host:~$ bitbake linux-firmware
```

这会生成一个新的软件包 *deploy/ipk/all/linux-firmware-example*。

最后一步，您必须将固件包安装到您的镜像中。您可以通过在 *local.conf* 或镜像 recipe 中添加对应的代码行来实现

```
# file local.conf or image recipe
IMAGE_INSTALL += "linux-firmware-example"
```

**警告**

确保您已将包名 *linux-firmware-example* 调整为您在 *linux-firmware_%.bbappend* 中指定的名称。

### 13.6.14 使用 *bbappend* 文件更改 *u-boot* 环境变量

所有 i.MX8M* 产品均使用 u-boot 作为 bootloader。可以使用临时方法修改 u-boot 环境变量。对 *u-boot-imx* 来说，环境变量定义位于 *include/configs/phycore_imx8m** 中与处理器名称对应的头文件。新的环境变量应添加在 *CONFIG_EXTRA_ENV_SETTINGS* 的末尾

```
#define CONFIG_EXTRA_ENV_SETTINGS \
[...]
PHYCORE_FITIMAGE_ENV_BOOTLOGIC \
"newvariable=1\0"
```

提交修改并在您的 layer 中创建文件 *u-boot-imx_%.bbappend* *<layer>* */recipes-bsp/u-boot/u-boot-imx_%.bbappend*

```
# contents of the file u-boot-imx_%.bbappend
FILESEXTRAPATHS:prepend := "${THISDIR}/features:"
SRC_URI:append = " \
    file://0001-environment-addition.patch \
"
```

### 13.6.15 通过 *bbappend* 文件更改 *barebox* 环境变量

从 *BSP-Yocto-AM335x-16.2.0* 和 *BSP-Yocto-i.MX6-PD16.1.0* 开始，*meta-phytec* 中的 *barebox* 环境变量已经采用新的机制。现在可以通过 *Python* bitbake 任务 *do_env* 在 *barebox* 环境中添加、更改和删除文件。有两个 *Python* 函数可以更改环境。它们是：

- *env_add(d, ***filename as string*, ***file content as string*)*: 添加新文件或覆盖现有文件
- *env_rm(d, ***filename as string*)*: 删除文件

自定义 layer *meta-racer* 的第一个示例中用 *bbappend* 文件中展示了如何在 *barebox* 环境文件夹 */env/nv/* 中加入新的非易失性变量 *linux.bootargs.fb*。

```
# file meta-racer/recipes-bsp/barebox/barebox_2022.02.0-phy1.bbappend
python do_env:append() {
    env_add(d, "nv/linux.bootargs.fb", "imxdrm.legacyfb_depth=32\n")
}
```

第二个示例显示如何替换网络配置文件 */env/network/eth0*

```
# file meta-racer/recipes-bsp/barebox/barebox_2022.02.0-phy1.bbappend
python do_env:append() {
    env_add(d, "network/eth0",
"""#!/bin/sh

# ip setting (static/dhcp)
ip=static
global.dhcp.vendor_id=barebox-${global.hostname}
```

(续下页)

(接上页)

```
# static setup used if ip=static
ipaddr=192.168.178.5
netmask=255.255.255.0
gateway=192.168.178.1
serverip=192.168.178.1
""")
}
```

在上述示例中，*Python* 的多行字符串语法 **"""text"""** 可以避免在 *Python* 代码中插入多个换行符 *\n*。*Python* 函数 *env_add* 可以用于添加和替换环境文件。

下一个示例显示如何删除已添加的环境文件，例如，*/env/boot/mmc*

```
# file meta-racer/recipes-bsp/barebox/barebox_2022.02.0-phy1.bbappend
python do_env:append() {
    env_rm(d, "boot/mmc")
}
```

### 调试

如果您想查看在构建过程中添加的所有环境文件，您可以在 *local.conf* 中启用调试标志

```
# file local.conf
ENV_VERBOSE = "1"
```

之后，您必须重新构建 *barebox* recipe 才能看到调试输出

```
host:~$ bitbake barebox -c clean
host:~$ bitbake barebox -c configure
```

最后一个命令的输出如下所示

```
[...]
WARNING: barebox-2022.02.0-phy1-r7.0 do_env_write: File 'nv/allow_color' content "false"
WARNING: barebox-2022.02.0-phy1-r7.0 do_env_write: File 'nv/linux.bootargs.base' content "consoleblank=0"
WARNING: barebox-2022.02.0-phy1-r7.0 do_env_write: File 'nv/linux.bootargs.fb' content "imxdrm.legacyfb_
↪depth=32"
WARNING: barebox-2022.02.0-phy1-r7.0 do_env_write: File 'nv/linux.bootargs.rootfs' content "rootwait ro␣
↪fsck.repair=yes"
```

### 修改环境（依赖所使用的 machine）

如果您只需将一些 *barebox* 环境设置应用于一台或多台设备，您可以利用 *Bitbake* 的 machine 覆盖语法。这个语法是，将 machine 名称或 SoC 名称（例如 *mx6*、*ti33x* 或 *rk3288*）通过下划线附加到变量或任务上。

```
DEPENDS:remove:mx6 = "virtual/libgl" or
python do_env_append_phyboard-mira-imx6-4().
```

下一个示例仅当 MACHINE 设置为 *phyboard-mira-imx6-4* 时才添加环境变量

```
# file meta-phytec/recipes-bsp/barebox/barebox_2022.02.0-phy1.bbappend
python do_env:append:phyboard-mira-imx6-4() {
    env_add(d, "nv/linux.bootargs.cma", "cma=64M\n")
}
```

*Bitbake* 变量覆盖语法的更详细解释如下：https://docs.yoctoproject.org/bitbake/2.4/bitbake-user-manual/bitbake-user-manual-metadata.html#conditional-metadata

#### 在旧的 BSP 版本升级 *barebox* 环境

在 BSP 版本 *BSP-Yocto-AM335x-16.2.0* 和 *BSP-Yocto-i.MX6-PD16.1.0* 之前，通过 *bbappend* 文件进行的 *barebox* 环境更改的机制和现在有所不同。例如，meta-layer（此处为 *meta-skeleton* ）中的目录结构可能如下所示

```
host:~$ tree -a sources/meta-skeleton/recipes-bsp/barebox/
sources/meta-skeleton/recipes-bsp/barebox
├── barebox
│   └── phyboard-wega-am335x-3
│       ├── boardenv
│       │   └── .gitignore
│       └── machineenv
│           └── nv
│               └── linux.bootargs.cma
└── barebox_%.bbappend
```

并且文件 *barebox_%.bbappend* 包含

```
# file sources/meta-skeleton/recipes-bsp/barebox/barebox_%.bbappend
FILESEXTRAPATHS:prepend := "${THISDIR}/barebox:"
```

在这个示例中，将会忽略 Layer *meta-phytec* 中目录 *boardenv* 下的所有环境变量更改，并添加文件 *nv/linux.bootargs.cma* 。而在 *barebox* 环境最新的处理机制下，您可以在 *Python* 任务 *do_env* 中使用 *Python* 函数 *env_add* 和 *env_rm* 来添加和删除环境。现在，上述示例被转换为文件 *barebox_%.bbappend* 中的一个单独的 *Python* 函数，如下所示。

```
# file sources/meta-skeleton/recipes-bsp/barebox/barebox_%.bbappend
FILESEXTRAPATHS:prepend := "${THISDIR}/barebox:"
python do_env:append() {
    # Removing files (previously boardenv)
    env_rm(d, "config-expansions")
    # Adding new files (previously machineenv)
    env_add(d, "nv/linux.bootargs.cma", "cma=64M\n")
}
```

### 13.6.16 更改网络配置

要在系统运行时调整 IP 地址、路由和网关，可以使用工具 *ifconfig* 和 *ip* 。以下是一些示例

```
target:~$ ip addr                                        # Show all network interfaces
target:~$ ip route                                       # Show all routes
target:~$ ip addr add 192.168.178.11/24 dev eth0         # Add static ip and route to interface eth0
target:~$ ip route add default via 192.168.178.1 dev eth0 # Add default gateway 192.168.178.1
target:~$ ip addr del 192.168.178.11/24 dev eth0         # Remove static ip address from interface eth0
```

网络配置由 *systemd-networkd* 管理。要查询当前状态，请使用

```
target:~$ networkctl status
target:~$ networkctl list
```

网络守护进程从目录 */etc/systemd/network/* 、*/run/systemd/network/* 和 */lib/systemd/network/* 读取配置（从高到低优先级）。*/lib/systemd/network/10-eth0.network* 中的示例配置如下所示

```
# file /lib/systemd/network/10-eth0.network
[Match]
Name=eth0

[Network]
Address=192.168.3.11/24
Gateway=192.168.3.10
```

这些文件 **.network* 取代了其他发行版中的 */etc/network/interfaces*。您可以直接编辑文件 *10-eth0.network*，或者将其复制到 */etc/systemd/network/* 并进行修改。修改文件后，您需要重启守护进程以使更改生效。

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

要查看网络守护进程的系统日志消息，请使用

```
target:~$ journalctl --unit=systemd-networkd.service
```

要在编译时修改网络配置，请查看 recipe *sources/meta-ampliphy/recipes-core/systemd/systemd-machine-units.bb* 和文件夹 *meta-ampliphy/recipes-core/systemd/systemd-machine-units/* 中的接口文件，其中定义了 *eth0* （以及可选的 *eth1* ）的静态 IP 地址配置。

有关更多信息，请参阅 https://wiki.archlinux.org/title/Systemd-networkd 和 https://www.freedesktop.org/software/systemd/man/latest/systemd.network.html。

## 13.6.17 更改无线网络配置

### 连接至 WLAN 网络

- 请首先为您的国家或地区选择合适的网络地域。

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

您会看到

```
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    # list all interfaces. Search for wlan*
target:~$ ip link set up dev wlan0
```

- 现在您可以扫描可用网络

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

您可以使用支持 *WEP*、*WPA* 和 *WPA2* 的跨平台身份认证客户端（称为 *wpa_supplicant*）来建立加密连接。

- 为此，请将网络凭据添加到文件 */etc/wpa_supplicant.conf*

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

- 现在可以建立连接

```
target:~$ wpa_supplicant -Dnl80211 -c/etc/wpa_supplicant.conf -iwlan0 -B
```

这会有以下结果

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

最后，您可以配置 DHCP 以获取 IP 地址（大多数 WLAN 接入点都支持此功能）。有关其他可能的 IP 配置，请参考更改网络配置 部分。

- 首先，创建目录

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

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

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

[Network]
DHCP=yes
```

- 现在，重新启动网络守护程序以使配置生效

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

### 创建 WLAN 接入点

本节提供基本的 *WPA2* 网络接入点 (AP) 的配置。

使用以下命令查找 WLAN 接口名称

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

编辑 */etc/hostapd.conf* 文件中的设置。这在很大程度上依赖于具体的应用场景。以下是一些示例。

```
# file /etc/hostapd.conf
interface=wlan0
driver=nl80211
ieee80211d=1
country_code=DE
hw_mode=g
ieee80211n=1
ssid=Test-Wifi
channel=2
wpa=2
wpa_passphrase=12345678
wpa_key_mgmt=WPA-PSK
wpa_pairwise=CCMP
```

通过 *systemd-networkd* 配置并启用网络接口 *wlan0* 的 DHCP 服务

```
target:~$ mkdir -p /etc/systemd/network/
target:~$ vi /etc/systemd/network/10-wlan0.network
```

将以下文本插入到文件中

```
[Match]
Name=wlan0

[Network]
Address=192.168.0.1/24
DHCPServer=yes

[DHCPServer]
EmitDNS=yes
target:~$ systemctl restart systemd-networkd
target:~$ systemctl status  systemd-networkd -l   # check status and see errors
```

启动用户空间后台进程 *hostapd*

```
target:~$ systemctl start hostapd
target:~$ systemctl status hostapd -l # check for errors
```

现在，您应该可以在终端设备（笔记本电脑、智能手机等）上看到 WLAN 网络 *Test-Wifi*。

如果接入点出现问题，您可以使用

```
target:~$ journalctl --unit=hostapd
```

或者从命令行以调试模式启动守护进程

```
target:~$ systemctl stop hostapd
target:~$ hostapd -d /etc/hostapd.conf -P /var/run/hostapd.pid
```

您会看到

```
...
wlan0: interface state UNINITIALIZED->ENABLED
wlan0: AP-ENABLED
```

有关 AP 设置和用户空间守护进程 *hostapd* 的更多信息，请访问

```
https://wireless.wiki.kernel.org/en/users/documentation/hostapd
https://w1.fi/hostapd/
```

#### phyCORE-i.MX 6UL/ULL 蓝牙

在使用 phyCORE-i.MX 6UL/ULL 的蓝牙功能时，需要特别谨慎。了解更多详情，请参考 L-844e.A5 i.MX 6UL/ULL BSP 手册 - 蓝牙。

### 13.6.18 添加 OpenCV 库和示例

*OpenCV*（开源计算机视觉 https://opencv.org/）是一个计算机视觉应用的开源库。

要安装库和示例，请编辑 *Yocto* 工程中的文件 *conf/local.conf* 并添加

```
# file conf/local.conf
# Installing OpenCV libraries and examples
LICENSE_FLAGS_ACCEPTED += "commercial_libav"
LICENSE_FLAGS_ACCEPTED += "commercial_x264"
IMAGE_INSTALL:append = " \
    opencv \
    opencv-samples \
```

(续下页)

(接上页)

```
    libopencv-calib3d2.4 \
    libopencv-contrib2.4 \
    libopencv-core2.4 \
    libopencv-flann2.4 \
    libopencv-gpu2.4 \
    libopencv-highgui2.4 \
    libopencv-imgproc2.4 \
    libopencv-legacy2.4 \
    libopencv-ml2.4 \
    libopencv-nonfree2.4 \
    libopencv-objdetect2.4 \
    libopencv-ocl2.4 \
    libopencv-photo2.4 \
    libopencv-stitching2.4 \
    libopencv-superres2.4 \
    libopencv-video2.4 \
    libopencv-videostab2.4 \
"
```

然后重编译镜像

```
host:~$ bitbake phytec-qt6demo-image
```

**小技巧**

大多数示例无法开箱即用，因为它们依赖于 *GTK* 图形库。此 BSP 仅支持 *Qt6* 。

### 13.6.19 使用 *lightpd* 安装最小的 PHP Web 运行环境

这个示例说明如何在目标镜像上添加 PHP 应用程序和 Web 服务。Lighttpd 可以通过 cgi 与 PHP 命令行工具一起使用。此解决方案仅占用 5.5 MiB 的磁盘空间。它已在 meta-ampliphy 中预先配置。只需调整 yocto 工程构建配置即可将其安装到镜像中。

```
# file conf/local.conf
# install lighttpd with php cgi module
IMAGE_INSTALL:append = " lighttpd"
```

启动镜像后，您会在 */www/pages* 目录下找到示例网页内容。为了测试 php，您可以删除 *index.html* 并将其替换为 *index.php* 文件。

```
<html>
  <head>
    <title>PHP-Test</title>
  </head>
  <body>
    <?php phpinfo(); ?>
  </body>
</html>
```

在您的主机上，您可以将浏览器指向主板的 IP（例如 192.168.3.11），然后 phpinfo 就会显示出来。

## 13.7 常见任务

### 13.7.1 调试用户空间应用程序

phytec-qt6demo-image 无需任何调整就可以进行交叉调试。您只需确保主机的 sysroot 与所使用的镜像相匹配。因此，您需要为该镜像创建一个编译工具链。

```
host:~$ bitbake -c populate_sdk phytec-qt6demo-image
```

此外，如果您希望对镜像中的所有程序和库具有完整的调试和回溯功能，您可以添加

```
DEBUG_BUILD = "1"
```

到 `conf/local.conf`。这在某些情况下并不是必需的。这样配置之后，编译器选项将从 FULL_OPTIMIZATION 切换到 DEBUG_OPTIMIZATION。查看 *Poky* 源代码以了解 DEBUG_OPTIMIZATION 的默认设置。

要开始交叉调试，请按照之前的说明安装 SDK，设置 SDK 环境，然后在同一个 shell 中启动 *Qt Creator*。如果您不使用 *Qt Creator*，可以在 souce SDK 环境脚本后直接运行 arm-<..>-gdb 调试器，该 gdb 调试器在 source 后会默认配置到您的 PATH 中。

如果您使用 *Qt Creator*，请查看随产品提供的相应文档（快速入门或应用程序指南），以获取有关如何设置工具链的信息。

使用调试器启动用户空间的应用程序时，您可能会遇到 SIGILL，这是来自 *libcrypto* 的非法指令。*Openssl* 通过捕获这些非法指令来检测系统功能，这将导致 *GDB* 中断被触发。您可以选择忽略此操作并点击 **继续**（命令 c）。如果希望永久忽略此信号，可以添加

```
handle SIGILL nostop
```

到您的 *GDB* 启动脚本或 *Qt Creator GDB* 配置面板中。其次，您可以禁用安全功能，这通过添加

```
set auto-load safe-path /
```

到同一个启动脚本，它允许从任何位置自动加载库。

如果您想要进行本地调试，需要在目标设备上安装调试符号表。您可以通过在 *conf/local.conf* 中添加以下行来实现这一点。

```
EXTRA_IMAGE_FEATURES += "dbg-pkgs"
```

对于交叉调试，这并不是必需的，因为调试符号表会从 PC 侧加载，并且 dbg-pkgs 也会包含在镜像的 SDK 中。

### 13.7.2 生成镜像源，开启离线构建

参照如下方式，修改您的 *site.conf* (如果您不使用 *site.conf*, 请修改 *local.conf* )

```
#DL_DIR ?= "" don't set it! It will default to a directory inside /build
SOURCE_MIRROR_URL = "file:///home/share/yocto_downloads/"
INHERIT += "own-mirrors"
BB_GENERATE_MIRROR_TARBALLS = "1"
```

现在运行

```
host:~$ bitbake --runall=fetch <image>
```

这适用于所有镜像及您希望提供镜像源的所有 MACHINE。它将生成所需的所有 *tar* 文件。我们可以移除所有 SCM 子文件夹，因为它们是和 tar 文件重复的。

```
host:~$ rm -rf build/download/git2/
etc...
```

请注意，我们使用本地镜像源来生成 dl_dir。这样，有一些文件是本地链接文件。

首先，我们需要将所有文件包括符号链接的源文件拷贝到新的镜像源目录中。

```
host:~$ rsync -vaL <dl_dir> ${TOPDIR}/../src_mirror/
```

Now we clean the */build* directory by deleting everything except */build/conf/* but including */build/conf/sanity*. We change *site.conf* as follows

```
SOURCE_MIRROR_URL = "file://${TOPDIR}/../src_mirror"
INHERIT += "own-mirrors"
BB_NO_NETWORK = "1"
SCONF_VERSION = "1"
```

BSP 目录现在可以使用以下方式压缩

```
host:~$ tar cfJ <filename>.tar.xz <folder>
```

其中文件名和文件夹应该以完整的 BSP 版本命名。

### 13.7.3 在目标主机上进行编译

在您的 *local.conf* 中添加

```
IMAGE_FEATURES:append = " tools-sdk dev-pkgs"
```

### 13.7.4 不同的工具链

在 *Poky* 中有多种方法可以创建工具链安装程序。一种方法是运行

```
host:~$ bitbake meta-toolchain
```

这将在 *build/deploy/sdk* 中生成一个工具链安装程序，可用于交叉编译目标应用。但是，这个安装程序不包含添加到您自定义镜像中的库，因此它只是一个单纯的 *GCC* 编译器。这适用于 bootloader 和 kernel 开发。

另一种方法是运行

```
host:~$ bitbake -c populate_sdk <your_image>
```

这将创建一个工具链安装程序，包含所有安装在目标根文件系统上的软件开发包。该安装程序涵盖开发应用程序所需的所有组件，可以被用户空间应用程序开发团队所使用。如果镜像中包含 *QT* 库，那么所有相关依赖库也将随安装程序提供。

第三个选项是创建 ADT（应用程序开发工具包）安装程序。它将包含交叉工具链和一些帮助软件开发人员的工具，例如 *Eclipse* 插件和 *QEMU* 系统模拟器。

```
host:~$ bitbake adt-installer
```

目前，我们的 BSP 尚未使用 ADT 进行测试。

### 使用 SDK

使用以下方式生成 SDK 后

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

使用以下方式运行生成的二进制文件

```
host:~$ deploy/sdk/ampliphy-glibc-x86_64-phytec-qt6demo-image-cortexa9hf-vfp-neon-toolchain-i.MX6-PD15.3-rc.
↪sh
Enter target directory for SDK (default: /opt/ampliphy/i.MX6-PD15.3-rc):
You are about to install the SDK to "/opt/ampliphy/i.MX6-PD15.3-rc". Proceed[Y/n]?
Extracting SDK...done
Setting it up...done
SDK has been successfully set up and is ready to be used.
```

您可以通过 source 工具链目录中的 *environment-setup* 脚本来激活当前 shell 的工具链环境

```
host:~$ source /opt/ampliphy/i.MX6-PD15.3-rc/environment-setup-cortexa9hf-vfp-neon-phytec-linux-gnueabi
```

然后，交叉编译器和链接器等必要工具就在您的 PATH 中了。要编译一个简单的 *C* 程序，请使用

```
host:~$ $CC main.c -o main
```

环境变量 $CC 包含 ARM 交叉编译器的路径和其他所需的编译器参数，如 *-march*、*-sysroot* 和 *--mfloat-abi*。

**小技巧**

您不能仅使用编译器名称来编译程序，例如

```
host:~$ arm-phytec-linux-gnueabi-gcc main.c -o main
```

在许多情况下它会导致编译失败。请使用 *CC*、CFLAGS、LDFLAGS 等。

为了方便起见，*environment-setup* 脚本会导出其他环境变量，如 CXX、LD、SDKTARGETSYSROOT。

编译 *C* 和 *C++* 程序的一个简单的 makefile 可能如下所示

```
# Makefile
TARGETS=c-program cpp-program

all: $(TARGETS)

c-program: c-program.c
    $(CC) $(CFLAGS) $(LDFLAGS) $< -o $@

cpp-program: cpp-program.cpp
    $(CXX) $(CXXFLAGS) $(LDFLAGS) $< -o $@

.PHONY: clean
clean:
    rm -f $(TARGETS)
```

要对目标进行编译，只需在执行 make 之前在当前 shell 中 source 工具链即可

```
host:~$ make     # Compiling with host CC, CXX for host architecture
host:~$ source /opt/ampliphy/i.MX6-PD15.3-rc/environment-setup-cortexa9hf-vfp-neon-phytec-linux-gnueabi
host:~$ make     # Compiling with target CC, CXX for target architecture
```

如果您需要在工具链的 sysroot 中指定额外包含的目录，则可以在 *-I* 参数中使用“=”符号，例如

```
-I=/usr/include/SDL
```

*GCC* 会用工具链中的 sysroot 路径替换它（此处为 */opt/ampliphy/i.MX6-PD15.3-rc/sysroots/cortexa9hf-vfp-neon-phytec-linux-gnueabi/*）。有关更多信息，请参阅 *GCC* 主页。

**小技巧**

变量 $CFLAGS 和 $CXXFLAGS 默认会包含编译器的调试标志“-g”。这会在二进制文件中加入调试信息，从而使文件变得更大。应该从正式生产的镜像中去除这个标志。如果您编写 *Bitbake* recipe，默认情况下也会启用“-g”。调试符号在 SDK 根文件系统中是有用的，以便在主机调用 *GDB* 时获取调试信息。但是在将软件包安装到目标根文件系统之前，*Bitbake* 会在程序上执行 *strip* 命令，以去除调试符号。因为默认情况下，这些符号在目标根文件系统中是不需要的。

#### 将 SDK 与 GNU Autotools 结合使用

*Yocto* SDK 是 *GNU Autotools* 项目会用到的一个工具。传统的主机编译过程通常是

```
host:~$ ./autogen.sh # maybe not needed
host:~$ ./configure
host:~$ make
host:~$ make install DESTDIR=$PWD/build/
```

用 *Yocto* SDK 对目标 machine 进行构建的命令也是非常相似的。以下命令假设 SDK 已解压到目录 */opt/phytec-ampliphy/i.MX6-PD15.3.0/*（可根据实际情况修改路径）

```
host:~$ source /opt/phytec-ampliphy/i.MX6-PD15.3.0/environment-setup-cortexa9hf-vfp-neon-phytec-linux-
↪gnueabi
host:~$ ./autogen.sh  # maybe not needed
host:~$ ./configure ${CONFIGURE_FLAGS}
host:~$ make
host:~$ make install DESTDIR=$PWD/build/
```

请参考官方 *Yocto* 文档以获取更多信息：https://docs.yoctoproject.org/4.2.4/singleindex.html#autotools-based-projects

### 13.7.5 使用 Kernel 模块

如果您需要为内核模块配置一些选项，或者将一个模块加入黑名单。您可以通过 *udev* 进行，并写入 **.conf* 文件中。

```
/etc/modprobe.d/\*.conf.
```

如果您希望在构建过程中指定 kernel 模块的一些选项，则有三个相关的变量。如果您仅想自动加载那些没有自动加载功能的模块，请将其添加到

```
KERNEL_MODULE_AUTOLOAD += "your-module"
```

无论是在 kernel recipe 中还是涉及到全局变量。如果您需要为模块指定选项，您可以这样做

```
KERNEL_MODULE_AUTOLOAD += "your-module"
KERNEL_MODULE_PROBECONF += "your-module"
module_conf_your-module = "options your-module parametername=parametervalue"
```

如果您想将某个模块列入自动加载黑名单，您可以直接使用

```
KERNEL_MODULE_AUTOLOAD += "your-module"
KERNEL_MODULE_PROBECONF += "your-module"
module_conf_your-module = "blacklist your-module"
```

### 13.7.6 使用 *udev*

Udev（Linux 动态设备管理）是一个系统守护进程，用于处理 /dev 中的动态设备管理。它由位于 */etc/udev/rules.d*（系统管理员配置空间）和 */lib/udev/rules.d/*（供应商提供）中的 *udev* 规则控制。以下是 *udev* 规则文件的示例

```
# file /etc/udev/rules.d/touchscreen.rules
# Create a symlink to any touchscreen input device
SUBSYSTEM=="input", KERNEL=="event[0-9]*", ATTRS{modalias}=="input:*-e0*,3,*a0,1,*18,*", SYMLINK+="input/
↪touchscreen0"
SUBSYSTEM=="input", KERNEL=="event[0-9]*", ATTRS{modalias}=="ads7846", SYMLINK+="input/touchscreen0"
```

有关语法和用法的更详细信息，请访问 https://www.freedesktop.org/software/systemd/man/latest/udev.html。要获取可以在 *udev* 规则中使用的设备属性列表，您可以利用 *udevadm info* 工具。该工具将打印出设备节点及其父节点的所有现有属性。输出中的键值对可以直接复制并粘贴到规则文件中。以下是一些示例。

```
target:~$ udevadm info -a /dev/mmcblk0
target:~$ udevadm info -a /dev/v4l-subdev25
target:~$ udevadm info -a -p /sys/class/net/eth0
```

更改 *udev* 规则后，您必须通知 udev 守护进程。否则，您的更改不会生效。使用以下命令

```
target:~$ udevadm control --reload-rules
```

在制定 *udev* 规则时，您应该监视事件，以便查看设备何时连接到系统或从系统断开连接。使用

```
target:~$ udevadm monitor
```

在另一个 shell 中监控系统日志也是非常有帮助的，尤其是在 udev 规则中执行了外部脚本的情况下。执行

```
target:~$ journalctl -f
```

**小技巧**

您无法在 *RUN* 属性中启动守护进程或大型脚本。请参阅 https://www.freedesktop.org/software/systemd/man/latest/udev.html#RUN%7Btype%7D 。

这仅适用于执行时间非常短的前台任务。长时间运行的事件进程可能会阻碍此设备或从设备的后续所有事件。启动守护进程或其他长时间运行的进程并不适合 *udev*；派生进程（无论是否和母进程分离）都将在母进程事件处理完成后无条件被终止。如果需要执行其他任务，您可以考虑使用特殊属性 *ENV{SYSTEMD_WANTS}="service-name.service"* 和 *systemd* 服务。

请参阅 https://unix.stackexchange.com/questions/63232/what-is-the-correct-way-to-write-a-udev-rule-to-stop-a-service-under-systemd。

CHAPTER 14

# 故障排除

## 14.1 setscene 任务告警

当 Yocto 缓存处于 dirty 状态时会出现此告警。

```
WARNING: Setscene task X ([...]) failed with exit code '1' - real task
```

但是请避免强制取消构建，或者如果必须取消，请按一次 Ctrl-C 并等待构建过程停止。要删除所有这些告警，请清除 sstate 缓存并删除 build 文件夹。

```
host:~$ bitbake phytec-headless-image -c cleansstate && rm -rf tmp deploy/ipk
```

CHAPTER 15

# Yocto 文档

对于 BSP 用户来说，最重要的文档可能是开发人员手册。https://docs.yoctoproject.org/4.2.4/dev-manual/index.html

关于常见任务的部分是一个很好的起始点。https://docs.yoctoproject.org/4.2.4/dev-manual/common-tasks.html#common-tasks

完整的文档可以在一个单独的 HTML 页面上找到，适合用户搜索功能或变量名称。https://docs.yoctoproject.org/4.2.4/singleindex.html