Application Note

AFE79xx Linux 驱动程序

Akshay Kumar, Navin Maheshwari, Mustafa Chikhliwala, Phanindra

摘要

AFE79xx 高性能模拟前端在无线通信基础设施中发挥着关键作用，其应用领域包括 5G 基站、大规模 MIMO 系统和波束赋形等。本应用手册解决了在 Linux 平台上部署 AFE79xx 器件所面临的复杂集成挑战。通信设备制造商、测试测量公司以及开发先进射频系统的国防承包商，都将受益于经过验证的集成流程，这些流程涵盖内核驱动程序和用户空间 SDK 两种方法。内核集成方法可为部署的基础设施提供具有优异性能的生产就绪型系统，而用户空间方法则可加快原型和测试环境的开发周期。完整的分步流程包括在 Vivado 中进行 FPGA 硬件设计，使用 PetaLinux 或 Yocto 配置 Linux 内核，设置 SPI 接口，以及部署可引导系统。两种方法均已在 ZCU102 平台上通过硬件验证，使工程团队能够高效地实现 AFE79xx 支持，同时保持未来更新和定制的灵活性。

内容

商标

1 简介

AFE79xx 是一系列高性能、高带宽、多通道收发器，集成了四个射频采样发送器链、四个射频采样接收器链和多达两个射频采样数字化辅助链（反馈路径）。

本应用手册提供了将 AFE79xx SDK 集成到基于 Linux 的系统中的说明。涵盖三种集成方法：

1. **使用 PetaLinux Tools 的 Linux 内核驱动程序** — 直接与 Linux 内核源代码一同编译
2. **使用 Yocto Project 的 Linux 内核驱动程序** — 直接与 Linux 内核源代码一同编译
3. **用户空间应用层** — 无需修改内核即可运行

内核方法（方法 1 和 2）具有优异的性能和紧密的系统集成，主要区别在于构建系统和工作流。用户空间方法则提供了开发灵活性并简化了维护，无需依赖内核。方法的选择取决于系统架构约束、性能要求、构建基础设施和部署场景。本文假设读者熟悉 Linux 内核开发、器件驱动程序概念和嵌入式系统设计基础知识。

2 Linux 内核驱动程序

AFE79xx 内核驱动程序有助于隔离设备 SDK 的两个组件：用于用户空间应用程序的高级 API 函数，以及内核空间内的设备特定处理。这简化了用户应用程序中的 SDK 集成。

- 所有 **SPI 通信**均整合在内核中。
- 使用标准 Linux SPI 子系统作为 **SPI 从器件**集成到内核中，可使驱动程序成为从器件的唯一所有者，确保执行原子操作并防止与其他 SPI 访问发生冲突。
- 用户访问功能以**字符驱动程序**形式提供，便于与用户应用程序集成，其工作方式符合标准 POSIX 文件操作（打开、释放、读取、写入和 ioctl）
- 它在 /dev 下以 **/dev/device_nameX.Y** 格式显示已连接的器件，其中 X 表示 SPI 主端口，Y 表示从器件 ID，以便于检测和识别。示例：/dev/afe79xx1.0
- 支持通过器件树使用格式为 ***ti,device_name*** 的 compatible 字符串进行**自动 调用和从器件分配**。示例：“ti,afe79xx”。该方法支持动态器件配置，无需重新编译驱动程序。
- **多个器件**在 /dev 下以唯一 ID 向用户呈现为独立器件。各个器件上下文完全在内核代码中采用各种标准 Linux 方法进行管理，以简化客户应用程序。
- 使用各种 Linux 机制验证**线程安全操作**

以下是该驱动程序实现方案的一些显著特性

以下是为使用 Linux 内核进行器件集成所提供的典型器件 SDK 目录结构

```
Device_SDK/
├── kernel-driver
└──────── device_name/*
└── userspace
├──────────device_name/*
├──────── example/*
└──────── Makefile
```

其中，kernel-driver 下的 device 文件夹用于构建内核模块，userspace 下的 device 文件夹包含高级 API 定义并用作用户应用程序的一部分，example 文件夹包含用户应用程序示例，Makefile 展示了将 API 与用户应用程序集成的典型编译方法。

2.1 使用 PetaLinux Tools 的 Linux 内核驱动程序

本节介绍使用 PetaLinux Tools 构建集成了 AFE79xx SDK 支持的 Linux 内核的过程。该方法采用内核树内构建方式，在构建过程中将 AFE79xx 驱动程序直接编译进内核源代码树中。

这一流程假定您的环境中已安装并提供了 PetaLinux Tools 2023.2（或更高版本）。

在开始 PetaLinux 构建过程之前，请确保用户具备：

1. **板级支持包 (BSP)**
   - 从 AMD/Xilinx 网站下载，或使用 PetaLinux 安装包中包含的版本
2. **硬件描述文件 (.xsa)**
   - 由 Vivado 硬件设计生成
   - 必须包含 FPGA 位流

关于创建参考硬件设计并为 ZCU102 板生成 XSA 文件的说明，请参阅**第 4 节**。

采用以下步骤：

第 1 步：在 Linux 终端上，运行以下命令创建一个 PetaLinux 项目

```
petalinux-create -t project -n <Project_Name> -s <path to BSP>
```

第 2 步：将目录更改为项目文件夹 **<Project_Name>** 并运行以下命令

```
petalinux-config –get-hw-description <path to XSA>
```

第 3 步：使用以下命令配置内核

```
petalinux-config -c kernel
```

此时将打开配置菜单。导航至 *Device drivers SPI support*，并启用以下选项：Cadence SPI controller、Xilinx SPI controller common module、Xilinx Zynq GQSPI controller、AMD SPI controller 和 User mode SPI device driver support。

图 2-1. 内核配置菜单

第 4 步：使用以下命令配置根文件系统

```
petalinux-config -c rootfs
```

此时将打开 RootFs 配置菜单。要启用 GCC 编译器，请导航至 *Filesystem packages-> misc > packagegroup-core-buildessential*，在其下选择 *packagegroup-core-buidessential* 和 *packagegroup-core-buidessential-dev*

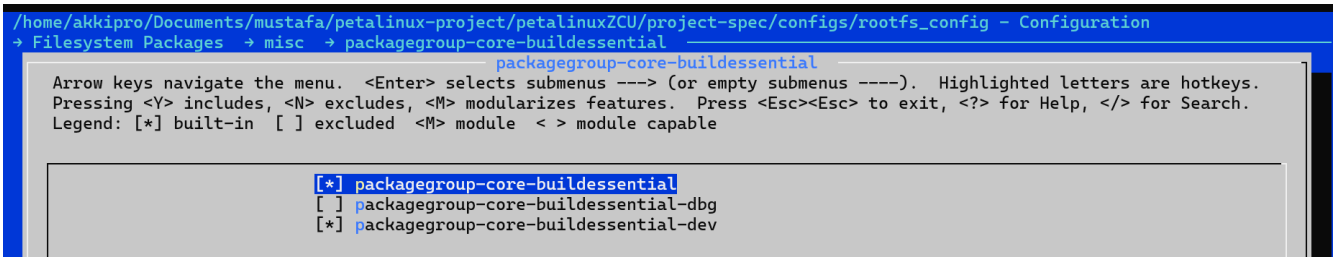

图 2-2. RootFs 配置菜单

第 5 步：运行以下命令，使内核源代码可修改：

```
petalinux-devtool modify linux-xlnx
```

该命令将内核源代码提取到工作目录。记下命令显示的输出路径，该路径通常为 <Project_Name>/components/yocto/workspace/sources/linux-xlnx/。导航至该目录：

```
cd <Path_to_kernel_Src>
```

第 6 步：在 <Path_to_kernel_Src>/drivers/char/ 中创建一个名为 afe79 的文件夹，并将 SDK 中 kernel-driver 下的 afe79 文件夹粘贴到该文件夹中

第 7 步：在 <Path_to_kernel_Src>/drivers/char 中的 Makefile 末尾添加以下行

```
obj-$(CONFIG_AFE79XX) += afe79/
```

这会告知内核构建系统在配置中启用驱动程序时编译 AFE79xx 驱动程序。

第 8 步：在 <Path_to_kernel_Src>/drivers/char 中的 Kconfig 文件中，找到文件末尾的 *endmenu* 行。将该行添加到 endmenu 之前。

```
source "drivers/char/afe79/Kconfig"
```

系统将读取 AFE79xx Kconfig 文件。该驱动程序将出现在 menuconfig 界面中。可通过菜单启用和禁用该驱动程序

第 9 步：配置内核以启用 AFE79xx 驱动程序。运行命令

```
petalinux-config -c kernel
```

这将打开内核配置菜单。导航至 device drivers -> character devices，按 *Y* 键启用 AFE79xx。

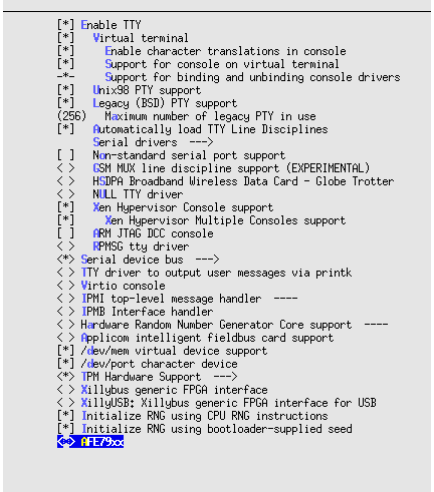

图 2-3. 字符驱动程序的内核配置菜单

第 10 步：必须更新器件树，以声明 AFE79xx 器件在 SPI 总线上，使内核能够识别并初始化硬件。在 PetaLinux 项目中，使用 *system-user.dtsi* 文件进行器件树定制修改。

转到器件树，通过编辑 *path* 中的 *system-user.dtsi* 文件，将 SPI 添加到器件树

```
< Project_Name>/project-spec/meta-user/recipes-bsp/device-tree/files/
```

在器件树中，添加 SPI 从器件条目，如 图 2-4 所示

```
/include/ "system-conf.dtsi"
/ {
};

&spi0 {
    is-decoded-cs= <0>;
    status = "okay";
    num-cs = <1>;
    afe79xx@0 {
        compatible = "ti,afe79xx";
        reg = <0>;
        spi-max-frequency = <10000000>;
    };
};
```

图 2-4. 器件树示例

这会在启动后在 /dev 下创建一个名为 */dev/afe79xx1.0* 的器件。

节点必须指定 compatible 字符串 *ti,afe79xx*，以便器件驱动程序绑定。

第 11 步：使用以下命令构建项目：

```
petalinux-build
```

此过程可能需要 30-60 分钟，具体取决于主机系统性能。

第 12 步：使用以下命令打包文件

```
petalinux-package --boot --fsbl < Project_Name >/images/linux/zynqmp_fsbl.elf --fpga <path to bit file> --u-boot -force
```

成功构建后，用户将拥有 准备 Linux 引导映像 所需的全部文件。

2.2 使用 Yocto Project 的 Linux 内核驱动程序

本节介绍如何使用 Yocto Project 将 AFE79xx SPI 内核驱动程序集成到 Linux 映像中。该方法不会绑定到任何特定的板卡或供应商工具链。它适用于任何存在 Yocto 兼容型板级支持包 (BSP) 的硬件平台。

该驱动程序以预构建的自定义 Yocto 层 **meta-afe79** 的形式提供，可添加到任何现有 Yocto 项目中。已拥有可正常工作的 Yocto 构建环境的用户可直接跳至第 2.2.4 节。首次设置 Yocto 的用户应按顺序执行所有步骤。

备注

本节中的所有配置语法均遵循 **Yocto Langdale (4.1.x)** 规范。Yocto 在不同版本之间经历了重大的语法变更。变量赋值运算符从 Dunfell 及更低版本中使用的下划线表示法（例如 IMAGE_INSTALL_append）变更为 Honister 中引入并在后续所有版本（包括 Langdale）中使用的冒号表示法 (IMAGE_INSTALL:append)。使用不同 Yocto 版本的用户必须相应地确认并更新语法。有关版本特定的更改，请参阅官方 Yocto Project 迁移指南：https://docs.yoctoproject.org。

2.2.1 Yocto Project 概述

Yocto Project 是一个开源构建框架、允许开发人员为嵌入式系统创建自定义 Linux 发行版。它本身并非 Linux 发行版，而是一套用于构建 Linux 发行版的工具和方法。Yocto 在其核心使用了一个名为 **BitBake** 的构建引擎，该引擎处理称为**配方** (recipe) 的元数据文件以获取、编译和打包软件组件。

Yocto 中的基本组织单位是**层**。层是一些相关配方、配置文件和补丁的集合，可为构建版本增添特定功能。大多数常用硬件平台都有官方层，通常由板卡或 SoC 供应商以板级支持包的形式提供。自定义层允许开发人员添加专有或项目特定软件（例如 AFE79xx 驱动），而无需修改上游层。

AFE79xx 驱动程序打包为 Linux 内核驱动程序，并作为自定义 Yocto 层的一部分提供。驱动程序源代码编译为可加载的内核模块（.ko 文件），该模块会在 Yocto 映像构建过程中构建，并安装到根文件系统中。一旦目标系统引导进入 Linux，驱动程序就会自动加载到正在运行的内核中，从而通过 SPI 实现与 AFE79xx 器件的通信。Yocto 提供了内置基础设施来构建和打包此类内核驱动程序，提供的 meta-afe79 层充分利用了这一能力。

2.2.2 主机系统要求

构建 Yocto 映像需要以下主机系统配置。

操作系统

推荐使用 Ubuntu 18.04 LTS 或更高版本。也可使用 Yocto 支持的其他 Linux 发行版；完整列表请参考 Yocto 官方文档。

表 2-1. 系统资源

资源	最小值	推荐
RAM	8GB	16GB
可用磁盘空间	100GB	200GB
CPU 核心	4	8 个或更多

所需软件包

继续之前，在主机系统上安装以下软件包：

```
sudo apt-get update
```

```
sudo apt-get install -y 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 pylint3 xterm python3-subunit \ mesa-common-dev zstd liblz4-tool device-tree-compiler
```

2.2.3 设置 Yocto 构建环境

本节提供初始化 Yocto 构建环境的操作说明。如果用户已为其目标板卡配置了可正常工作的 Yocto 设置，可跳至**第 2.2.4 节**。

第 1 步：下载 Poky

Poky 是 Yocto Project 的参考发行版，包含 BitBake 以及核心元数据层。克隆与您所需 Yocto 版本对应的 Poky 版本。下文将 poky 根目录称为 **project_directory**。下方示例使用 Langdale 分支：

```
git clone -b langdale https://git.yoctoproject.org/poky.git
```

第 2 步：初始化构建环境

在 project_directory 文件夹中，获取 Yocto 环境设置脚本来创建构建目录并配置 shell 环境：

```
source poky/oe-init-build-env <build_directory>
```

该命令创建一个构建目录，生成默认配置文件，并将 BitBake 添加到 shell 路径中。每次启动新的终端会话时都必须运行该命令。

重要提示：请始终从项目根目录运行该命令。在构建目录内部运行它会创建嵌套的构建文件夹并重置所有配置。

2.2.4 配置板卡专用层

第 1 步：下载板卡专用层

每个硬件平台都需要对应的 BSP 层，为该板卡提供机器配置、内核和引导加载程序配方。BSP 层通常由板卡或 SoC 供应商提供，并需单独克隆。请查阅板卡供应商的文档，确定与所选 Yocto 版本兼容的正确 BSP 层和分支。

对于基于 Xilinx 的平台，相关层包括 meta-xilinx、meta-xilinx-bsp、meta-xilinx-standalone 和 meta-xilinx-tools，均可从 Xilinx GitHub 存储库获取。对于其他平台，如 Raspberry Pi、NXP 或 Intel，应使用供应商提供的等效 BSP 层。我们可以使用 git clone 获取这些层，示例如下：

Xilinx

```
git clone -b langdale https://github.com/Xilinx/meta-xilinx.git
```

```
git clone -b langdale https://github.com/Xilinx/meta-xilinx-tools.git
```

Intel - **Altera**

```
git clone https://git.yoctoproject.org/meta-intel-fpga
```

此外，许多 BSP 层依赖 OpenEmbedded 层集合 (meta-openembedded) 中的软件包。同样克隆该集合的对应分支：

```
git clone -b langdale https://github.com/openembedded/meta-openembedded.git
```

第 2 步：添加层

使用 bitbake-layers 命令将所有必需的层添加到构建中。所需的确切层集取决于目标板卡。至少必须包含核心 Poky 层、板卡 BSP 层和 AFE79 层。仅上一步克隆的层才可使用以下命令添加：

```
bitbake-layers add-layer <Path_to_layer>
```

例如，在 ZCU102 上需要以下层：

```
bitbake-layers add-layer ~/yocto-zcu102/meta-xilinx/meta-xilinx-core
```

```
bitbake-layers add-layer ~/yocto-zcu102/meta-xilinx/meta-xilinx-bsp
```

```
bitbake-layers add-layer ~/yocto-zcu102/meta-xilinx/meta-xilinx-standalone
```

```
bitbake-layers add-layer ~/yocto-zcu102/meta-xilinx-tools
```

```
bitbake-layers add-layer ~/yocto-zcu102/meta-openembedded/meta-oe
```

```
bitbake-layers add-layer ~/yocto-zcu102/meta-openembedded/meta-python
```

```
bitbake-layers add-layer ~/yocto-zcu102/meta-openembedded/meta-networking
```

```
bitbake-layers add-layer ~/yocto-zcu102/meta-openembedded/meta-multimedia
```

使用以下命令确认所有必需的层处于活动状态。

```
bitbake-layers show-layers
```

第 3 步：添加硬件描述文件和目标机器

许多 Yocto BSP 工作流需要一个硬件描述文件，用于捕获目标平台的完整硬件设计。Yocto 通过 **local.conf** 格式使用硬件描述文件，自动为特定硬件设计生成正确的 FSBL、PMU 固件和器件树。

conf/local.conf 接受硬件描述文件路径和机器变量，以正确提供构建配置。

打开构建目录中的文件 <project_directory/build_directory>/conf/local.conf，将 MACHINE 变量设置为与目标板卡对应的标识符。该标识符在板卡 BSP 层中定义。例如，ZCU102 板使用标识符 zcu102-zynqmp。

.

请参考板卡 BSP 文档获取正确的机器名称。

Xilinx 使用 .XSA 文件，Intel 使用 SOPC 信息文件。

对于 Xilinx：

将 XSA 文件放在主机系统上的已知位置。建议位置是在自定义层内：

meta-afe79/recipes-bsp/hdf/files/

然后在构建目录的 <project_directory/build_directory>/conf/local.conf 文件中添加以下行，将路径和文件名替换为您的 XSA 文件路径及机器变量：

```
MACHINE ??= "zcu102-zynqmp"
HDF_BASE = "file://"
FILESEXTRAPATHS:prepend:pn-external-hdf := "/absolute/path/to/meta-afe79/recipes-bsp/hdf/files:"
XILINX_RELEASE_VERSION = "v2023.2"
PREFERRED_VERSION_external-hdf = "2023.2"
LICENSE_FLAGS_ACCEPTED += "xilinx"
```

备注

XILINX_RELEASE_VERSION 必须与生成 XSA 文件的 Vivado 版本匹配。Vivado 2023.2 生成的 XSA 需要 XILINX_RELEASE_VERSION = “v2023.2”。使用不匹配的版本会导致 XSCT 工具处理硬件描述时失败。

根据特定 BSP 层的要求配置硬件交接路径。请参考 BSP 层 README 文件获取正确的变量名和路径格式，因为这些信息在不同供应商平台代际之间会有所不同。

2.2.5 集成 AFE79xx Yocto 层

AFE79xx 驱动程序以名为 **meta-afe79** 的即用型 Yocto 层形式提供。该层包含所有必要的配方、驱动程序源代码以及参考器件树覆盖层。除器件树配置外，无论目标板卡或 Yocto 版本如何，集成步骤均相同。

第 1 步：将器件层添加到项目中

将 TI 提供的 SDK 中的 meta-afe79 层目录复制到您的 Yocto **project_directory** 中。然后，在已激活 Yocto 环境的构建目录中，向 BitBake 注册该层：

```
bitbake-layers add-layer <path_to_meta-afe79>
```

确认该层出现在活动层列表中：

```
bitbake-layers show-layers
```

第 2 步：将器件驱动程序包包含在最终的 Linux 映像中

向构建目录中的 <project_directory/build_directory>/conf/local.conf 文件添加以下行。这将指示 Yocto 编译 AFE79xx 驱动程序模块，并将其包含在输出映像的根文件系统中：

```
IMAGE_INSTALL:append = " afe79"
```

备注

对于比 Honister 更低的 Yocto 版本，请将冒号表示法替换为下划线表示法：

```
IMAGE_INSTALL_append = " afe79"
```

一旦包含到映像中，将通过位于目标文件系统 /etc/modules-load.d/afe79.conf 处的模块自动加载配置文件，将该驱动程序模块设置为在系统启动时自动加载。

要将 GCC 编译器和开发工具添加到映像中，请添加以下几行。这将有助于在用户空间中进行编译。

```
IMAGE_FEATURES += "tools-sdk"
```

第 3 步：配置器件树覆盖层文件

Linux 驱动程序使用器件树来发现其管理的硬件。要绑定器件驱动程序，器件树必须在适当的 SPI 总线上包含一个 compatible 字符串为 **ti,afe79xx** 的节点。添加此节点的方法取决于所用板卡 BSP 使用的 Yocto 工作流。

对于基于 Xilinx 的板卡，meta-afe79 层包含一个即用型覆盖层文件（位于 recipes-bsp/device-tree/files/system-user.dtsi）以及一个对应的 bbappend 文件（该文件会自动将此覆盖层包含在构建中）。构建前，请打开 system-user.dtsi 文件，确认 SPI 节点标签和属性与您的板卡硬件设计匹配。提供的文件引用了 spi0，这与 ZCU102 评估板上使用的 SPI 控制器相对应。如果您的板卡使用不同的 SPI 控制器或节点标签，请相应地更新文件。

以 Xilinx ZCU102 板为例，为 SPI0 添加 SPI 从器件条目，如下所示：

转到器件树，通过编辑 **path<path_to_meta-afe79>meta-afe79\recipes-bsp\device-tree\files** 中的 **system-user.dtsi** 文件，将 SPI 添加到器件树

以 Xilinx ZCU 板为例，在器件树中为 SPI0 添加 SPI 从器件条目，如下所示：

```
/include/ "system-conf.dtsi"
/ {
};

&spi0 {
    is-decoded-cs= <0>;
    status = "okay";
    num-cs = <1>;
    afe79xx@0 {
        compatible = "ti,afe79xx";
        reg = <0>;
        spi-max-frequency = <10000000>;
    };
};
```

图 2-5. 器件树示例

这将在启动后在 /dev 下创建一个名为“/dev/afe79xx1.0”的器件。确认 SPI 节点标签和属性与您的板卡硬件设计匹配。

第 4 步：将我们的自定义器件树文件包含到基础器件树文件中

Xilinx 使用 device-tree.bbappend 文件，intel 使用 linux-socfpga.bbappend 文件，以在不更改原始配方的情况下将自定义 dtsi 文件包含到基础器件树中。

meta-afe79 层中包含的 mata-afe79/recipes-bsp/device-tree/device-tree.bbappend 文件专属于 Xilinx DTG 工作流。其他平台的用户应移除此文件，或将其替换为适合 BSP 的机制。

第 5 步：构建映像

为目标映像配方运行 BitBake 构建命令：

```
bitbake <image-name>
```

将 <image-name> 替换为适合您项目的映像配方。常见选项包括：

- **core-image-minimal** – 仅包含基本软件包的最小可引导映像。适用于需要较小占用空间的生产部署。
- **core-image-minimal-dev** – 在 **core-image-minimal** 的基础上扩展了开发工具和头文件。在目标板卡上直接编译或调试应用程序时很有用。
- **core-image-full-cmdline** – 一个控制台映像，包含更完整的标准 Linux 命令行实用程序集。适用于开发和测试环境。
- **petalinux-image-minimal** – 使用 meta-xilinx BSP 层时可用的 Xilinx 专属最小映像。

首次构建可能需要一到三小时，具体取决于主机系统性能和网络速度，因为 BitBake 会下载并编译所有必需的源代码。由于状态缓存是共享的，后续构建将显著加快。

构建过程中，出现有关主下载 URL 失效的警告消息是正常的。BitBake 会自动使用镜像服务器重试，这些警告并不代表构建失败。

2.2.6 验证集成

将构建的映像刷写到目标板卡并引导进入 Linux 后，运行以下命令确认 AFE79xx 驱动程序已成功加载。

确认该模块出现在已加载模块列表中：

```
lsmod | grep afe79
```

查看内核日志中的驱动程序初始化消息：

```
dmesg | grep -i "afe79"
```

成功初始化将产生类似以下内容的输出：

```
afe79: loading out-of-tree module taints kernel.
```

AFE79xx SPI 驱动程序初始化

```
afe79xx spi1.0: AFE79xx device initialized successfully
```

```
afe79xx spi2.0: AFE79xx device initialized successfully
```

确认字符器件文件已创建：

```
ls -la /dev/afe79*
```

该驱动程序会为在 SPI 总线上检测到的每个 AFE79xx 实例创建一个器件文件，并根据总线编号和片选索引命名，例如 /dev/afe79xx1.0 和 /dev/afe79xx2.0。

2.2.7 故障排查

表 2-2. 故障排查

问题	症状	解决方法
引导时模块未加载	lsmod 未显示 afe79 条目	确认 local.conf 中存在 IMAGE_INSTALL:append =“afe79”。检查目标系统上是否存在 /etc/modules-load.d/afe79.conf。
构建语法错误	与 IMAGE_INSTALL 或 KERNEL_MODULE_AUTOLOAD 相关的 BitBake 解析错误	确认 Yocto 版本。对于低于 Honister 的版本，请使用下划线语法。
驱动程序探测失败	dmesg 中没有初始化消息	确认器件树在正确的 SPI 总线上包含一个具有 compatible =“ti,afe79xx”且 status =“okay”的节点。
器件文件未创建	/dev/afe79* 不存在	检查 dmesg 是否存在驱动程序绑定错误。确认 SPI 控制器已在器件树中启用。
SPI 通信故障	应用程序返回不正确的读取值或错误	确认使用了正确的器件文件。确认 SPI 频率和模式与硬件匹配。对于 Xilinx 平台，确认 FPGA 位流通过 FMC 连接器正确地将 SPI 信号路由至 AFE79xx 器件。
嵌套构建目录	配置意外复位	始终从项目根目录获取环境脚本，切勿从构建目录内获取。如果意外创建了嵌套的构建子目录，请将其删除。

2.3 准备 Linux 引导映像

本节介绍如何对 SD 卡进行分区，将引导映像和根文件系统复制到相应分区，以及准备可引导介质。

请按照以下步骤操作：

第 1 步：使用以下命令格式化 SD 卡并创建 2 个分区

```
sudo gparted
```

第一个分区为 fat32 格式，命名为 BOOT（为其分配 1GB 空间），第二个分区为 ext4 格式，命名为 rootfs（为其分配剩余空间）。

第 2 步：将 <project_name>/images/linux/ 文件夹中的 BOOT.bin、image.ub 和 boot.scr 复制到 BOOT 分区

```
sudo cp <FileName> /media/<username>/ BOOT/
```

第 3 步：使用以下命令将文件系统（<project_name>/images/linux/ 中的 rootfs.tar.gz）解压到 rootfs 分区

```
sudo tar -xf rootfs.tar.gz -C /media/<username>/rootfs/
```

第 4 步：使用以下命令复制 rootfs.cpio 文件（来自 <project_name>/images/linux/）

```
sudo cp rootfs.cpio /media/<username>/rootfs/
```

第 5 步：弹出 SD 卡（弹出是为了避免 rootfs 损坏）。

第 6 步：现在将 SD 卡插入 ZCU 板，并将 UART 线缆连接到 PC。为板卡上电（将 SW6 设置为 SD 卡引导模式），Linux 将启动，可通过 Putty 等软件访问终端。

完成本节操作后，就已成功使用 PetaLinux Tools 和树内构建方法构建含有 AFE79xx 驱动程序的 Linux 内核。AFE SPI 器件已在器件树中正确注册，并呈现为字符器件，系统已为 AFE 驱动程序操作和应用程序开发做好充足准备。

2.4 从用户空间运行 AFE79xx 器件

CAFE 包提供了一个高级 API，用于从用户空间应用程序与 AFE79xx 内核驱动程序通信。将其与您的软件栈一同编译，即可配置和控制 AFE79xx 器件。

下方示例展示了一些带有器件实例的高级 API 调用表示。

```
#include <fcntl.h>
#include "tiAfe79xx.h"
int main(void) {
    const char *device = "/dev/afe79xx1.0"; // Assuming /dev/afe79xx1.0 is available
    int tiAfe79dev = open(device, O_RDWR); // Get the device instance using ioctl
    if (tiAfe79dev < 0){return 1;} // Sanity check if device instance is generated successfully
    Afe79_setAfeLogLvl(tiAfe79dev, 2); // Model API call with single argument
    Afe79_overrideTdd(tiAfe79dev, 0xF, 0x3, 0xF, 1); // Model API call with multiple arguments
    close(tiAfe79dev); // Release device instance to make it available to other applications
    return 0;
 }
```

备注

有关要遵循的确切 API 序列以及 API 的详细信息，请参阅器件 API 文档。

安装用户空间库

转到 SDK 中的 userspace 文件夹并运行：

1. make
2. sudo make install

使用用户空间库编译用户应用程序：

由于库和头文件已安装到 Linux 中，用户在编译时无需指定任何路径。在调用任何器件 API 的位置只需包含单个头文件 tiDeviceName.h(tiAfe79xx.h)，并在链接过程中提供 -ldevicename(-lafe79xx)。

3 在应用层构建 AFE79xx SDK

本节介绍将 AFE79xx SDK 构建为用户空间应用程序，为“在应用层构建 AFE79xx SDK”中介绍的内核树内驱动程序方法提供了另一种集成方法。在该方法中，完整的 AFE79xx 驱动程序功能在用户空间而非内核空间中实现，通过 SPI 直接与硬件通信。该方法为开发和调试提供了更大的灵活性，因为用户空间应用程序更新时无需重新编译内核或重启系统。

3.1 使用 PetaLinux Tools 构建不含 AFE79xx SDK 的 Linux 内核

本节介绍为 ZCU102 板构建不含 AFE79xx 驱动程序的标准 Linux 内核，为用户空间驱动程序方法建立基础系统。由于 AFE79xx SDK 将在后续章节中编译并部署为用户空间应用程序，因此内核构建不需要驱动程序定制修改或 AFE79xx 特定配置。生成的内核映像仅包含通用 SPI 设备驱动程序 (spidev) 和标准 ZCU102 板支持，为用户空间 AFE79xx SDK 的运行提供基础。该方法简化了内核管理，并使 AFE79xx 驱动程序能够独立于内核进行开发、更新和部署。

AFE79xx SDK 需要通过 SPI 接口与器件通信。**第 5 节**介绍了为 ZCU102 FPGA 添加处理器和 SPI 的参考设计。我们可以完全按照该节步骤生成 .xsa 和 .bit。

按照以下步骤构建不含 AFE79xx 驱动程序的内核。其中仅包含配置 AFE79xx 所需的 SPI：

第 1 步：在 Linux 终端上，运行以下命令创建一个 PetaLinux 项目

```
petalinux-create -t project -n <Project_Name> -s <path to BSP>
```

第 2 步：将目录更改为项目文件夹 **<Project_Name>** 并运行以下命令

```
petalinux-config –get-hw-description <path to XSA>
```

第 3 步：使用以下命令配置内核

```
petalinux-config -c kernel
```

此时将打开配置菜单。导航至“Device drivers”->“SPI support”并启用选项 Cadence SPI、Xilinx SPI、Xilinx Zynq QSPI 和 User mode SPI device driver。

图 3-1. 内核配置菜单

第 4 步：使用以下命令配置根文件系统

```
petalinux-config -c rootfs
```

此时将打开 RootFs 配置菜单。要启用 GCC 编译器，导航至“Filesystem packages”->“misc”->“packagegroup-core-buildessential”，在其下选择“packagegroup-core-buidessential”和“packagegroup-core-buidessential-dev”

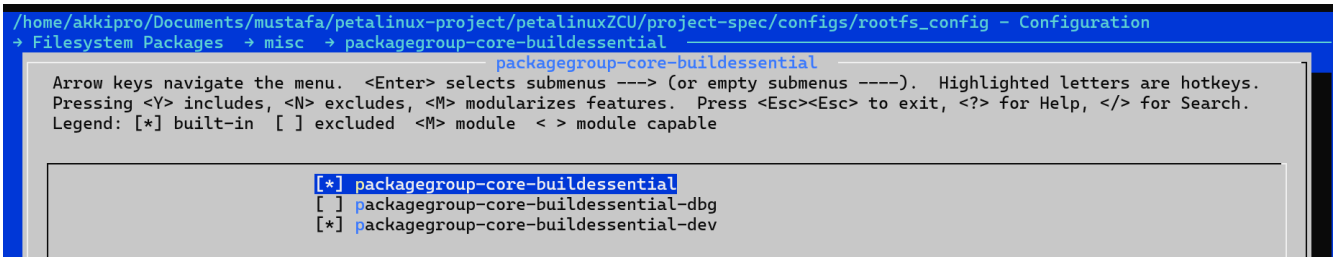

图 3-2. 构建依赖项的 RootFs 配置菜单

第 5 步：必须更新器件树，以声明 AFE79xx 器件在 SPI 总线上，使内核能够识别并初始化硬件。在 PetaLinux 项目中，使用 **system-user.dtsi** 文件进行器件树定制修改。

转到器件树，通过编辑 **path** 中的 **system-user.dtsi** 文件，将 SPI 添加到器件树

```
<Project_Name>/project-spec/meta-user/recipes-bsp/device-tree/files/
```

在器件树中，添加 SPI 从器件条目，如下所示：

```
/include/ "system-conf.dtsi"
/ {
};
&spi0 {
    num-cs = <1>;
    status = "okay";
    spidev@0 {
        compatible="rohm,dh2228fv";
        reg =<0>;
        spi-max-frequency = <10000000>;
    };
};
```

图 3-3. 器件树示例

这将在启动后在 /dev 下创建一个名为“/dev/spidev1.0”的器件。

第 6 步：使用以下命令构建项目：

```
petalinux-build
```

此过程可能需要 30-60 分钟，具体取决于您的主机系统性能。

第 7 步：使用以下命令打包文件

```
petalinux-package --boot --fsbl < Project_Name >/images/linux/zynqmp_fsbl.elf --fpga <path to bit file> --u-boot -force
```

成功构建后，我们应拥有第 4.2 节所需的所有文件。我们可以严格按照第 4.2 节的步骤，对 SD 卡进行分区，将引导映像和根文件系统复制到相应分区，并准备可引导介质。

3.2 在用户空间中集成 SPI 与 AFE79xx

AFE79xx 在 *CAFE\Afe79xxUser\Src\tiAfe79_baseFunc.c* 中对底层 SPI 读写函数进行了抽象。实现 SPIDEV 的 SPI 读写函数，以便与器件交互。

此示例使用结构体 afe79InstDeviceInfo 的 halConfig 成员来存储 SPI 句柄。该成员已集成在 ti_afe79_afeSpiRawWrite 和 ti_afe79_afeSpiRawRead 函数中。

```
/**
    @brief AFE SPI Write driver function.
    @details AFE SPI Write driver function. The contents of this function should be replaced by host SPI driver function.
    @param afeInst AFE Instance of AFE79_INST_TYPE type
    @param addr Address to be written to.
    @param data value to be written.
    @return Returns if the function execution passed or failed.
*/
TI_AFE_API_COMP uint8_t AFE79FNP(afeSpiRawWrite)(AFE79_INST_TYPE afeInst, uint16_t addr, uint8_t data)
{
    // afeLogDbg("WRITE: Addrress: 0x%X, data: 0x%X", addr, data);
    spi_write(afeInst->halConfig, addr, data);
    return TI_AFE_RET_EXEC_PASS;
}

/**
    @brief AFE SPI read driver function.
    @details AFE SPI read driver function and returns the read value as pointer. The contents of this function should be replaced by host SPI driver function.
    @param afeInst AFE Instance of AFE79_INST_TYPE type
    @param addr Address to be read from.
    @param readVal Pointer return of the value read.
    @return Returns if the function execution passed or failed.
*/
TI_AFE_API_COMP uint8_t AFE79FNP(afeSpiRawRead)(AFE79_INST_TYPE afeInst, uint16_t addr, uint8_t *readVal)
{
    spi_read(afeInst->halConfig, addr, readVal);
    // afeLogDbg("READ: Address: 0X%X, Read Val: 0X%X", addr, *readVal);
    return TI_AFE_RET_EXEC_PASS;
}
```

图 3-4. SPI 写入和读取示例

创建 spidev 句柄并与 SDK 一起编译的示例代码已包含在软件包中。

4 面向 ZCU102 FPGA 的参考设计（含处理器和 SPI）

本节介绍在 Vivado 中为 Zynq UltraScale+ MPSoC 平台创建硬件参考设计的步骤。该设计包含具有 SPI 通信接口的处理系统，并生成进行软件开发和驱动程序集成所需的硬件描述文件。生成的两个关键文件是：

- 硬件描述 (.xsa)：导出完整的硬件规格，包括 Zynq 处理系统配置、SPI 外设设置、内存寻址和中断映射等。该文件使 Linux 构建系统能自动生成器件树，并配置与您的特定硬件实现方案匹配的内核驱动程序。
- FPGA 位流 (.bit)：包含用于对 FPGA 结构进行编程的合成硬件逻辑。此文件打包到引导映像 (BOOT.BIN) 中，并在系统初始化期间加载以配置可编程逻辑。

两个文件都必须在完成硬件设计后生成，并在后续构建 Linux 系统和集成驱动程序的步骤中使用。

4.1 配置 Zynq 处理器

本节介绍如何在 Vivado 中通过添加和配置 Zynq UltraScale+ MPSoC 与 Processor System Reset IP 模块来建立基础模块设计。这些模块构成了硬件设计的基础，SPI 接口和其他外设都将以此为基础构建。

第 1 步：打开 Vivado 并创建一个新项目。选择所有默认设置，然后选择所需的板卡。示例中使用的是 ZCU102。

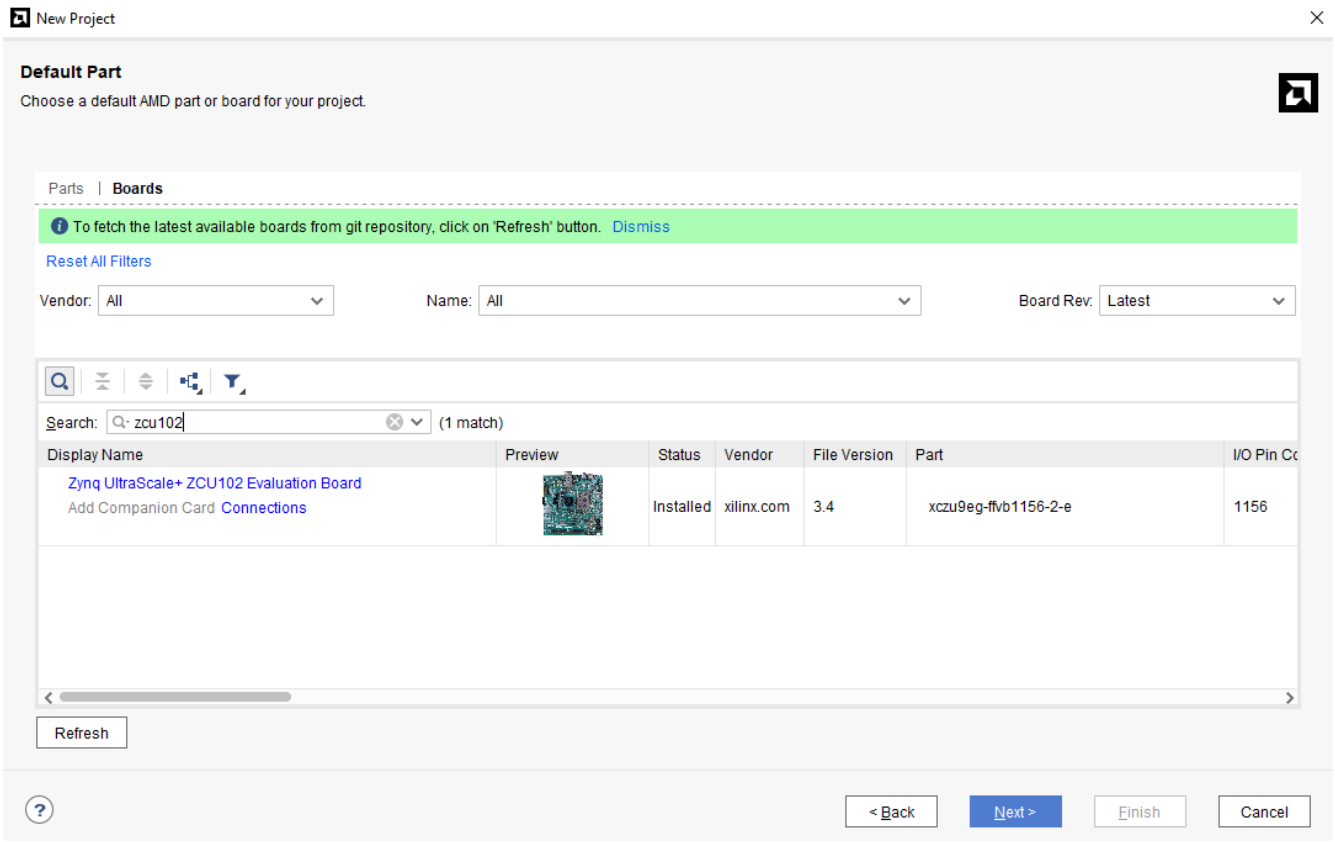

图 4-1. 项目的板卡选择

第 2 步：单击 *Create Block Design* 创建新的模块设计

第 3 步：单击 Add IP 或按 Ctrl + I。搜索 Zynq UltraScale+ MPSoC IP 模块并通过选择它进行添加。

图 4-2. 将 IP 添加到项目中

第 4 步：单击 Add IP 或按 Ctrl + I。搜索 Processor System Reset IP 模块并通过选择它进行添加

第 5 步：运行模块自动化。此设计将如下所示。

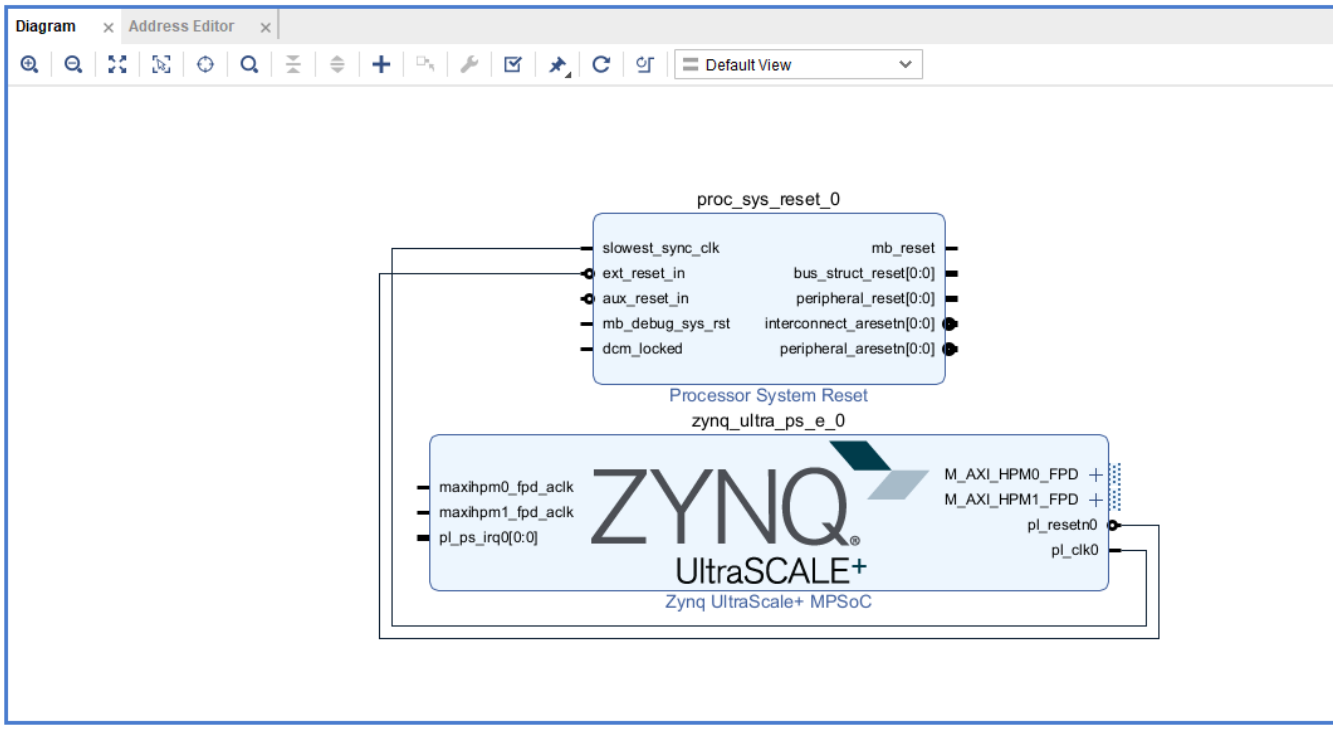

图 4-3. 将处理器添加到项目中

4.2 配置 SPI 接口

本节介绍如何启用并配置 Zynq 处理系统中的集成 SPI 控制器。

请按照以下步骤操作：

第 1 步：双击 Zynq UltraScale+ MPSoC，转到 I/O Configuration > Low Speed > I/O Peripherals，在 Zynq UltraScale+ MPSoC 中启用 SPI（启用所需数量的片选信号），然后单击 OK。

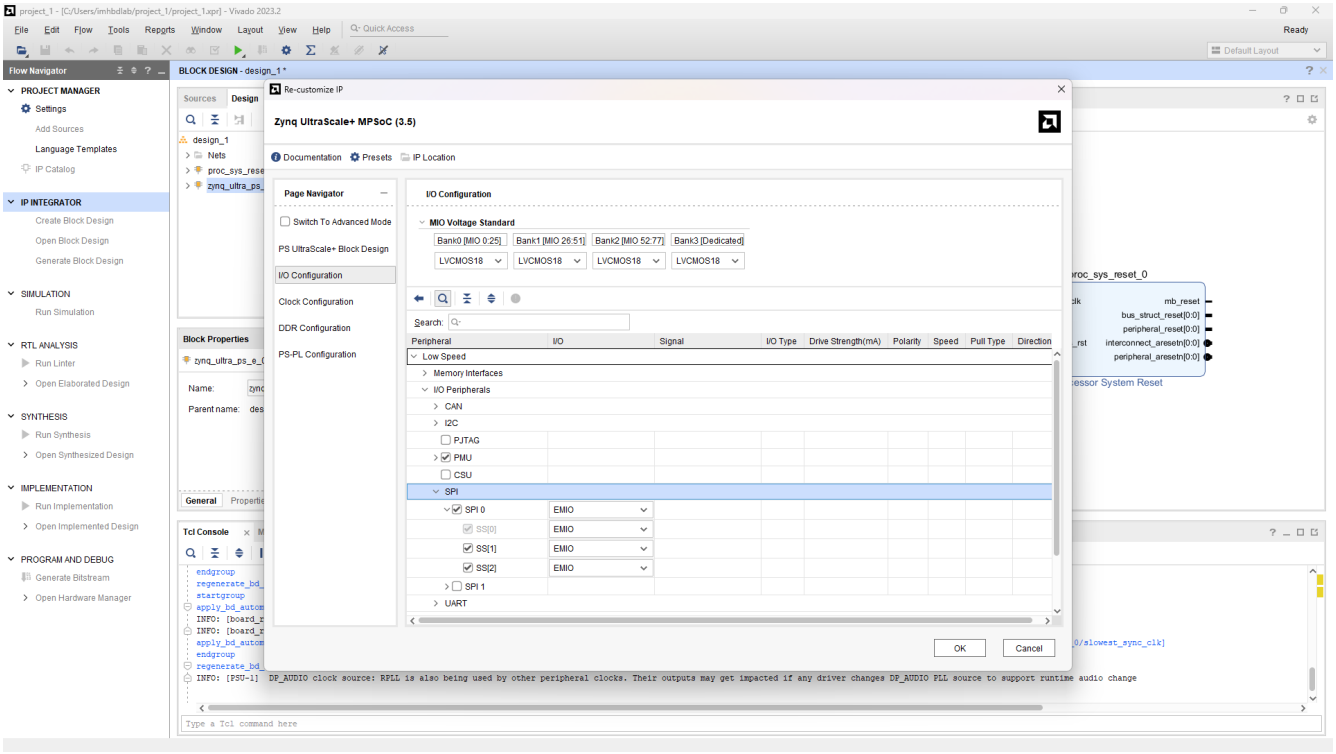

图 4-4. SPI 引脚配置

第 2 步：单击 SPI0 前面的 + 号。依次右键单击 emio_spi0_sclk_o、emio_spi0_m_o、emio_spi0_m_i 和 emio_spi0_ss_o_n，选择 *Make External* 或选中 SPI 端口后按 *Ctrl+T*。

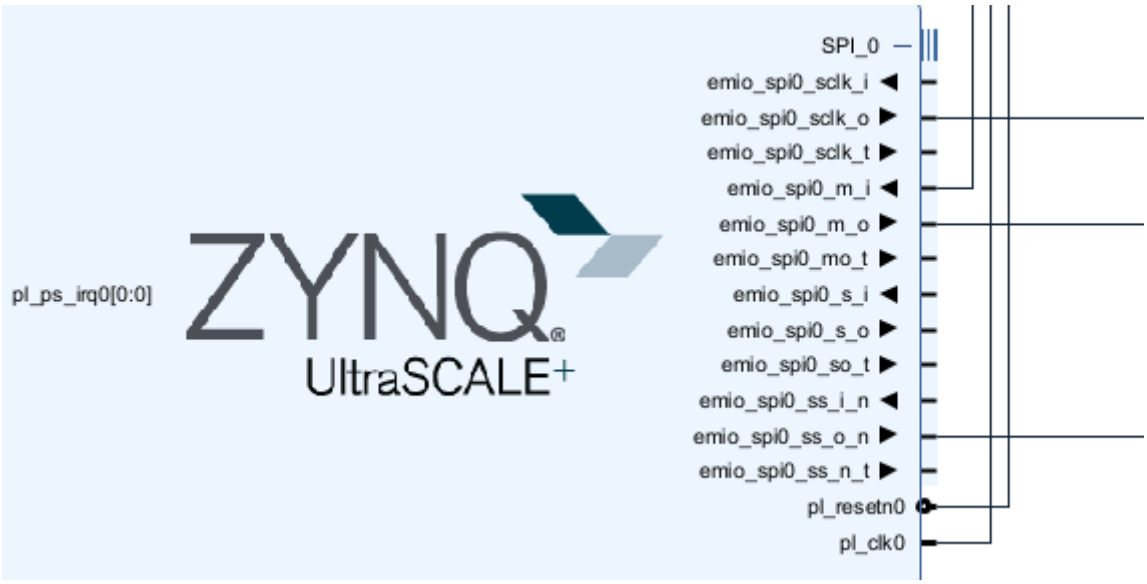

图 4-5. 选择 I/O 端口

第 3 步：双击 Zynq 核心，转到 PS/PL Configuration > PS-PL Interfaces > Master Interface，禁用 AXI HPM0 FPD 和 AXI HPM1 FPD，然后单击 *OK*。

第 4 步：保存并验证设计。按 Ctrl + S 保存设计。按 F6 验证设计。最终设计应类似于 图 4-6

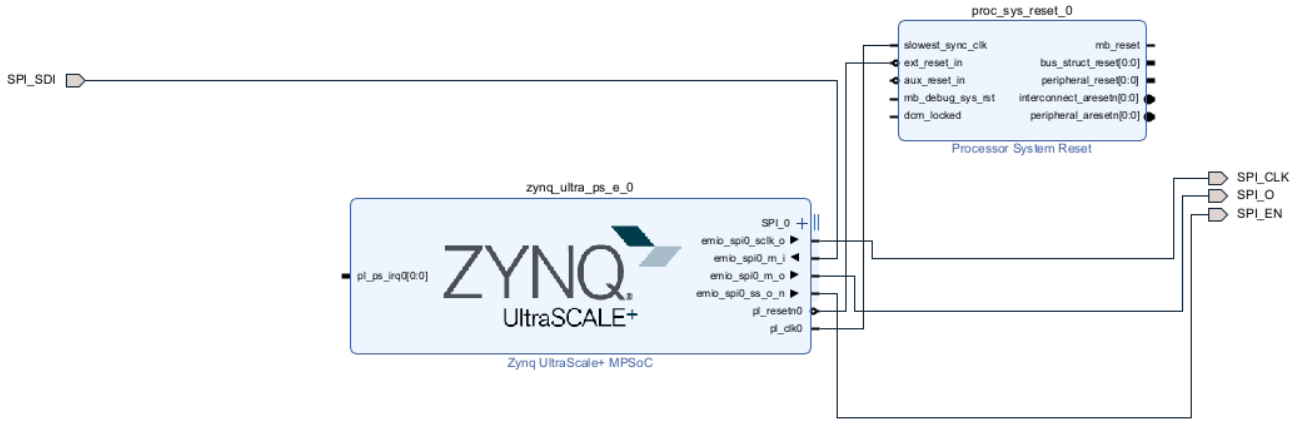

图 4-6. 示例项目

第 5 步：在 Sources 下，右键单击模块设计并选择 *Create HDL Wrapper*，然后选择默认设置。

4.3 生成硬件描述

本节介绍如何从已完成的 Vivado 设计生成并导出硬件描述文件 (.xsa) 和 FPGA 位流 (.bit)。

请按照以下步骤操作：

第 1 步：运行合成。完成后打开合成设计。

第 2 步：转到 Window -> IO Ports，为 SPI 线路分配所需的 IO 端口引脚。

SPI_0_14396 (4)	(Multiple)					✓	65	LVCMOS18	1.800		(Multiple)	
Scalar ports (4)												
SPI_CLK	OUT				AG3	✓	65	LVCMOS18	1.800		12	
SPI_EN	OUT				AJ5	✓	65	LVCMOS18	1.800		12	
SPI_O	OUT				AH3	✓	65	LVCMOS18	1.800		12	
SPI_SDI	IN				AJ6	✓	65	LVCMOS18	1.800			

图 4-7. SPI I/O 配置

AFE79xx EVM 的 I/O 映射如下（适用于 ZCU102 上的 J4 HPC1 FMC 连接器）。

第 3 步：单击 Generate bitstream。

第 4 步：单击 *File* > *Export* 并选择 *Export Hardware*。这将生成 .xsa 文件。

第 5 步：单击 *File* > *Export* 并选择 *Export bitstream file*。这将生成 .bit 文件。

硬件设计阶段到此结束。带有 SPI 接口的 Zynq 参考设计已从 Vivado 导出，生成了后续构建 Linux 内核和集成驱动程序所需的硬件描述 (.xsa) 和位流 (.bit) 文件。

4.4 使用 SD 卡引导 ZCU102 板

本节介绍使用准备好的 SD 卡引导 ZCU102 板并确认系统成功初始化的过程。引导序列包括从 SD 卡加载第一阶段引导加载程序、U-Boot、Linux 内核、器件树和根文件系统。配置板卡的引导模式设置，连接串行控制台以监控引导过程，并确认系统成功引导进入 Linux 命令提示符。

请按照以下步骤操作：

第 1 步：配置引导模式

设置 ZCU102 引导模式开关以从 SD 卡引导：

- SW6[4:1] = OFF-OFF-OFF-ON

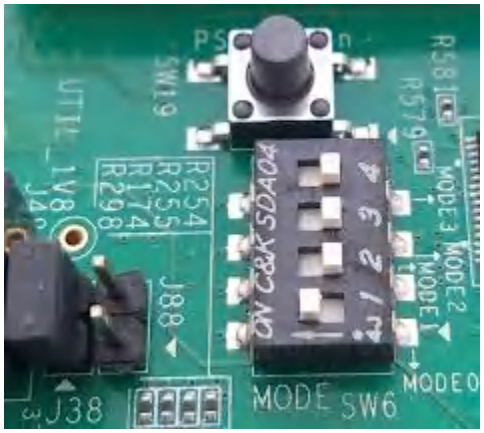

图 4-8. SD 卡引导对应的开关状态

第 2 步：将准备好的 SD 卡插入 ZCU102 板上的 SD 卡槽 (J100)。

第 3 步：连接串行控制台

连接 USB-UART 线缆：

- 将 micro-USB 线缆从 ZCU102 的 UART 端口 (J83) 连接到 PC
- PC 会识别出 FTDI USB 转 UART 器件

第 4 步：打开终端软件

启动串行终端应用程序（如 PuTTY、Tera Term 或 Minicom）：

- **端口：**选择分配给 ZCU102 的 COM 端口（例如，Windows 上为 COM3，Linux 上为 /dev/ttyUSB0）
- **波特率：**115200

第 5 步：为板卡上电

打开 ZCU102 电源开关 (SW1)。

第 6 步：监控引导过程

观察终端中的引导消息。系统会依次经历以下阶段：

- 第一阶段引导加载程序 (FSBL)
- U-Boot
- Linux 内核加载
- 根文件系统挂载
- 登录提示符

第 7 步：登录

在登录提示符下输入：

- **用户名：**petalinux
- **密码：**设置新密码。

5 总结

本应用手册提供了使用两种不同方法将 AFE79xx SDK 集成到 Zynq UltraScale+ ZCU102 平台上的 Linux 系统中的完整步骤。内核树内驱动程序方法演示了如何将 AFE79xx 驱动程序源代码添加到 Linux 内核树中，通过修改 Kconfig 和 Makefile 来配置内核构建系统，并编译具有集成驱动程序支持的完整 Linux 系统。该方法提供了优异的性能和紧密的系统集成，适用于生产部署。用户空间 SDK 方法展示了如何将 AFE79xx 功能构建为应用层软件，以通过通用 spidev 接口与硬件通信，从而在不依赖内核的情况下提供开发灵活性。两种方法具有相同的基础步骤，包括创建含 SPI 接口配置的 Vivado 硬件设计，生成硬件描述文件以及设置 PetaLinux 项目。本文最后介绍了准备可引导 SD 卡介质和启动 ZCU102 板以确认系统成功初始化的步骤。开发人员现在拥有了在 Zynq 平台上实施 AFE79xx 集成的经过验证的方法，以及根据具体项目需求、性能约束条件和维护考量因素在内核空间与用户空间方法之间进行选择的明确指导。生成的系统提供了完整的 AFE79xx 器件控制能力，可随时用于应用程序开发和部署。

6 参考资料

1. AMD，*PetaLinux Tools 文档：参考指南*，参考指南。
2. Yocto Project，*Yocto Project 文档*，网页。

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、与某特定用途的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他安全、安保法规或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

最后更新日期：2025 年 10 月