See all versions of this document

PetaLinux Tools Documentation

Reference Guide

UG1144 (v2021.1) June 16, 2021

XILINX®

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
06/16/2021 Version 2021.1	
Chapter 7: Customizing the Project	Added a new section: Configuring UBIFS Boot.
Chapter 5: Booting and Packaging	Updated Steps to Boot a PetaLinux Image on Hardware with SD Card.
Appendix A: Migration	Added FPGA Manager Changes, Yocto Recipe Name Changes, Host GCC Version Upgrade.
Chapter 10: Advanced Configurations	Updated U-Boot Configuration and Image Packaging Configuration.

Table of Contents

XILINX

Chapter 1

Overview

Introduction

PetaLinux is an embedded Linux Software Development Kit (SDK) targeting FPGA-based system-on-a-chip (SoC) designs. This guide helps the reader to familiarize with the tool enabling overall usage of PetaLinux.

You are assumed to have basic Linux knowledge, such as how to run Linux commands. You should be aware of OS and host system features, such as OS version, Linux distribution, security privileges, and basic Yocto concepts.

The PetaLinux tool contains:

- Yocto Extensible SDK (eSDK)
- XSCT (Xilinx Software Command-Line Tool) and toolchains
- PetaLinux CLI tools

Note: Vitis™ unified software platform is the integrated design environment (IDE) for creating embedded applications on Xilinx® microprocessors. Refer to *Vitis Unified Software Platform Documentation: Embedded Software Development* (UG1400) for more details.

PetaLinux SDK is a Xilinx development tool that contains everything necessary to build, develop, test, and deploy embedded Linux systems.

Yocto Extensible SDK

The following table details the four extensible SDKs installed.

Table 1: **Extensible SDKs**

Path	Architecture
`$PETALINUX/components/yocto/source/aarch64`	Zynq® UltraScale+™ MPSoC and Versal™ ACAP
`$PETALINUX/components/yocto/source/arm`	Zynq-7000 devices
`$PETALINUX/components/yocto/source/microblaze_full`	MicroBlaze™ platform full designs
`$PETALINUX/components/yocto/source/microblaze_lite`	MicroBlaze platform lite designs

***Note*:** Microblaze_lite will be deprecated in future releases.

***Note*:** Earlier, the eSDKs were extracted in the specified path but now they are in self-extractable TAR files. From 2021.1, your eSDK scripts have the same name and are extracted into `<plnx-proj-root>/components/yocto` when you run the `petalinux-config` or the `petalinux-build` command in the PetaLinux project. The project extracts the corresponding eSDK, for example, if you create a Zynq UltraScale+ MPSoC project, then only the aarch64 eSDK is extracted into the `<plnx-proj-root>/components/yocto` project.

XSCT and toolchains

For all embedded software applications (non-Linux), the PetaLinux tool uses XSCT underneath. The Linux toolchain for all three architectures is from Yocto.

PetaLinux Command Line Interface (CLI) tools

This contains all the PetaLinux commands that you require. The CLI command tools are:

- `petalinux-create`
- `petalinux-config`
- `petalinux-build`
- `petalinux-util`
- `petalinux-package`
- `petalinux-upgrade`
- `petalinux-devtool`
- `petalinux-boot`

Navigating Content by Design Process

Xilinx® documentation is organized around a set of standard design processes to help you find relevant content for your current development task. All Versal™ ACAP design process Design Hubs can be found on the Xilinx.com website. This document covers the following design processes:

- **Embedded Software Development:** Creating the software platform from the hardware platform and developing the application code using the embedded CPU. Also covers XRT and Graph APIs. Topics in this document that apply to this design process include:
  - Chapter 3: Creating a Project
  - Chapter 4: Configuring and Building
  - Chapter 5: Booting and Packaging

- Chapter 7: Customizing the Project
- Chapter 8: Customizing the Root File System

XILINX

Chapter 2

Setting Up Your Environment

Installation Steps

Installation Requirements

The PetaLinux tools installation requirements are:

- Minimum workstation requirements:
  - 8 GB RAM (recommended minimum for Xilinx® tools)
  - 2 GHz CPU clock or equivalent (minimum of eight cores)
  - 100 GB free HDD space
  - Supported OS:
    - Red Hat Enterprise Workstation/Server 7.4, 7.5, 7.6, 7.7, 7.8, 7.9, 8.1, 8.2 (64-bit)
    - CentOS Workstation/Server 7.4, 7.5, 7.6, 7.7, 7.8, 7.9, 8.1, 8.2 (64-bit)
    - Ubuntu Linux Workstation/Server 16.04.5, 16.04.6, 18.04.1, 18.04.2, 18.04.3, 18.04.4, 18.04.5, 20.04, 20.04.1 (64-bit)
- You need to have root access to install the required packages mentioned in the release notes. The PetaLinux tools need to be installed as a non-root user.
- PetaLinux requires a number of standard development tools and libraries to be installed on your Linux host workstation. Install the libraries and tools listed in the release notes on the host Linux.
- PetaLinux tools require that your host system `/bin/sh` is 'bash'. If you are using Ubuntu distribution and your `/bin/sh` is 'dash', consult your system administrator to change your default system shell `/bin/sh` with the `sudo dpkg-reconfigure dash` command.

Note: For package versions, refer to the PetaLinux 2021.1 Release Notes and Master Answer Record: 76256.

CAUTION! *Consult your system administrator if you are not sure about the correct procedures for host system package management.*

IMPORTANT! *PetaLinux 2021.1 works only with hardware designs exported from Vivado® Design Suite 2021.1.*

Prerequisites

- The PetaLinux tools installation requirements are met. See the Installation Requirements for more information.
- The PetaLinux installer is downloaded. You can download PetaLinux installer from PetaLinux Downloads.

Installing the PetaLinux Tool

Without any options, the PetaLinux tool is installed into the current working directory.

```
chmod 755 ./petalinux-v<petalinux-version>-final-installer.run
./petalinux-v<petalinux-version>-final-installer.run
```

Alternatively, you can specify an installation path.

```
./petalinux-v<petalinux-version>-final-installer.run [--log <LOGFILE>] [-
d|--dir <INSTALL_DIR>] [options]
```

Table 2: **PetaLinux Installer Options**

Options	Description
--log <LOGFILE>	Specifies where the log file should be created. By default, it is `petalinux_installation_log` in your working directory.
-d|--dir [INSTALL_DIR]	Specifies the directory where you want to install the tool kit. If not specified, the tool kit is installed in your working directory.
-p|--platform <arch_name>	Specifies the architecture: aarch64: Sources for Zynq UltraScale+ MPSoC devices and Versal devices. arm: sources for Zynq devices. microblaze_lite: sources for microblaze_lite microblaze_full: sources for microblaze_full

For example: To install PetaLinux tools under `/home/<user>/petalinux/<petalinux-version>`:

```
$ mkdir -p /home/<user>/petalinux/<petalinux-version>
$ ./petalinux-v<petalinux-version>-final-installer.run --dir /home/<user>/
petalinux/<petalinux-version>
```

***Note*:** You can not install PetaLinux with root user. If you try to run PetaLinux as root, you may get a bitbake sanity check failure that prevents the build from continuing. This check is done because it is very risky to run builds as root; if any build script mistakenly tries to install files to the root path (/) instead of where it is supposed to, it must be made to fail immediately and not (in the worst case) overwrite files critical to your Linux system's operation, which means in `/bin` or `/etc`. Thus, running the build as root is not supported. The only time root access is needed is (completely outside of a build) when the `runqemu` script uses sudo to set up TAP devices for networking.

This installs the PetaLinux tool into the `/home/<user>/petalinux/<petalinux-version>` directory. By default, it installs all the four eSDKs. To install a specific eSDK as part of the PetaLinux tool, see Installing a Preferred eSDK as part of the PetaLinux Tool.

IMPORTANT! *Once installed, you cannot move or copy the installed directory. In the above example, you cannot move or copy `/home/<user>/petalinux/<petalinux-version>` because the full path is stored in the Yocto e-SDK environment file.*

***Note*:** While installing the software, ensure that `/home/<user>/petalinux` is writable for you. You can change the permissions after installation to make it globally read-execute (0755). It is not mandatory to install the tool in `/home/<user>/petalinux` directory. You can install it at any location that has the 755 permissions.

Reading and agreeing to the PetaLinux End User License Agreement (EULA) is a required and integral part of the PetaLinux tools installation process. You can read the license agreement prior to running the installation. If you wish to keep the license for your records, the licenses are available in plain ASCII text in the following files:

- **`$PETALINUX/etc/license/petalinux_EULA.txt`:** EULA specifies in detail the rights and restrictions that apply to PetaLinux.
- **`$PETALINUX/etc/license/Third_Party_Software_End_User_License_Agreement.txt`:** This third party license agreement details the licenses of the distributable and non-distributable components in PetaLinux tools.

By default, the WebTalk option is disabled to send tools usage statistics back to Xilinx. You can turn on the WebTalk feature by running the `petalinux-util --webtalk` command after the installation is complete.

IMPORTANT! *Before running the PetaLinux command, you need to source PetaLinux settings. For more information, see PetaLinux Working Environment Setup.*

```
$ petalinux-util --webtalk on
```

Installing a Preferred eSDK as part of the PetaLinux Tool

As described in Installing the PetaLinux Tool, the PetaLinux tool has four eSDKs: aarch64, arm, microblaze_full, and microblaze_lite. While installing the tool, you can specify your preferred eSDK, for example, if you are working on a Zynq platform, you can only install the arm eSDK into the PetaLinux tool. However, by default, all platform eSDKs are installed into the tool install directory. To install the desired eSDK, follow these examples:

- To install eSDKs for all Xilinx supported architectures like Zynq, Zynq UltraScale+ MPSoC, Versal, microblaze_lite, and microblaze_full:

```
 $ ./petalinux-v<petalinux-version>-final-installer.run --dir
<INSTALL_DIR>
```

- To install only the Zynq eSDK for arm architecture:

```
$ ./petalinux-v<petalinux-version>-final-installer.run --dir
<INSTALL_DIR> --platform "arm"
```

- To install the Zynq, Zynq UltraScale+ MPSoC, and Versal devices eSDKs for arm and aarch64 architecture:

```
$ ./petalinux-v<petalinux-version>-final-installer.run --dir
<INSTALL_DIR> --platform "arm aarch64"
```

- To install microblaze_lite and microblaze_full eSDKs for MicroBlaze architecture:

```
$ ./petalinux-v<petalinux-version>-final-installer.run --dir
<INSTALL_DIR> --platform "microblaze_lite microblaze_full"
```

Troubleshooting

This section describes some common issues you may experience while installing the PetaLinux tool. If the PetaLinux tool installation fails, the file `petalinux_installation_log` is generated in your PetaLinux installation directory.

Table 3: **PetaLinux Installation Troubleshooting**

Problem / Error Message	Description and Solution
`WARNING: You have less than 1 GB free space on the installation drive`	**Problem Description:** This warning message indicates that the installation drive is almost full. You might not have enough free space to develop the hardware project and/or software project after the installation. **Solution:** Clean up the installation drive to clear some more free space. Alternatively, move PetaLinux installation to another hard disk drive.
`WARNING: No tftp server found`	**Problem Description:** This warning message indicates that you do not have a TFTP service running on the workstation. Without a TFTP service, you cannot download Linux system images to the target system using the U-Boot network/TFTP capabilities. This warning can be ignored for other boot modes. **Solution:** Enable the TFTP service on your workstation. If you are unsure how to enable this service, contact your system administrator.
`ERROR: GCC is not installed - unable to continue. Please install and retry`	**Problem Description:** This error message indicates that you do not have gcc installed on the host workstation. **Solution:** Install gcc using your Linux workstation package management system. If you are unsure how to do this, contact your system administrator. See Installation Steps.

Table 3: **PetaLinux Installation Troubleshooting** *(cont'd)*

Problem / Error Message	Description and Solution

`ERROR: You are missing the following system tools required by PetaLinux: missing-tools-list`
or
`ERROR: You are missing these development libraries required by PetaLinux: missing-library-list`
	Problem Description:
	This error message indicates that you do not have the required tools or libraries listed in the "missing-tools-list" or "missing-library-list".
	Solution:
	Install the packages of the missing tools. For more information, see Installation Requirements.

`./petalinux-v<petalinux-version>-final-installer.run: line 52: petalinux_installation_log: Permission denied`	**Problem Description:** This error message indicates that PetaLinux install directory does not have writable permissions. **Solution:** Set 755 permissions to the install directory.

PetaLinux Working Environment Setup

After the installation, the remaining setup is completed automatically by sourcing the provided `settings` scripts.

Prerequisites

This section assumes that the PetaLinux tools installation is complete. For more information, see Installation Steps.

Steps to Set Up PetaLinux Working Environment

1. Source the appropriate settings script:
   - For Bash as user login shell:

     ```
     $ source <path-to-installed-PetaLinux>/settings.sh
     ```

   - For C shell as user login shell:

     ```
     $ source <path-to-installed-PetaLinux>/settings.csh
     ```

     Below is an example of the output when sourcing the setup script for the first time:

     ```
     PetaLinux environment set to '/opt/pkg/petalinux'
     INFO: Checking free disk space
     INFO: Checking installed tools
     INFO: Checking installed development libraries
     INFO: Checking network and other services
     WARNING: No tftp server found - please refer to "UG1144 <petalinux-version> PetaLinux Tools Documentation Reference Guide" for its impact and solution
     ```

2. Verify that the working environment has been set:

```
$ echo $PETALINUX
```

Example output: `/opt/pkg/petalinux`

Environment variable `$PETALINUX` should point to the installed PetaLinux path. The output may be different from this example based on the PetaLinux installation path.

Troubleshooting

This section describes some common issues that you may experience while setting up PetaLinux Working Environment.

Table 4: **PetaLinux Working Environment Troubleshooting**

Problem / Error Message	Description and Solution
`WARNING: /bin/sh is not bash`	**Problem Description:** This warning message indicates that your default shell is linked to dash. **Solution:** PetaLinux tools require your host system /bin/sh is bash. If you are using Ubuntu distribution and your /bin/sh is dash, consult your system administrator to change your default host system /bin/sh with the `sudo dpkg-reconfigure dash` command.

`Failed to open PetaLinux lib`	**Problem Description:**
	This error message indicates that a PetaLinux library failed to load. The possible reasons are:
	• The PetaLinux `settings.sh` has not been loaded.
	• The Linux Kernel that is running has SELinux configured. This can cause issues with regards to security context and loading libraries.
	Solution:
	1. Source the `settings.sh` script from the top-level PetaLinux directory. For more information, see PetaLinux Working Environment Setup.
	2. If you have SELinux enabled, determine if SELinux is in enforcing mode. If SELinux is configured in enforcing mode, either reconfigure SELinux to permissive mode (see the SELinux manual) or change the security context of the libraries to allow access.
	`$ cd $PETALINUX/tools/xsct/lib/lnx64.o`
	`$ chcon -R -t textrel_shlib_t lib`

Design Flow Overview

In general, the PetaLinux tools follow a sequential workflow model. The table below provides an example design workflow, demonstrating the order in which the tasks should be completed and the corresponding tool or workflow for that task.

Table 5: **Design Flow Overview**

Design Flow Step	Tool / Workflow
Hardware platform creation (for custom hardware only)	Vivado® design tools
Create a PetaLinux project	`petalinux-create -t project`
Initialize a PetaLinux project (for custom hardware only)	`petalinux-config --get-hw-description`
Configure system-level options	`petalinux-config`
Create user components	`petalinux-create -t COMPONENT`
Configure the Linux kernel	`petalinux-config -c kernel`
Configure the root filesystem	`petalinux-config -c rootfs`
Build the system	`petalinux-build`
Package for deploying the system	`petalinux-package`
Boot the system for testing	`petalinux-boot`
Upgrades the workspace	`petalinux-upgrade`
Use Yocto devtools command	`petalinux-devtool`
Use debug utilities	`petalinux-util`

Creating a Project

Project Creation Using PetaLinux BSP

PetaLinux board support packages (BSPs) are reference designs on supported boards for you to start working with and customizing your own projects. In addition, these designs can be used as a basis for creating your own projects on supported boards. PetaLinux BSPs are provided in the form of installable BSP files, and include all necessary design and configuration files, pre-built and tested hardware, and software images ready for downloading on your board or for booting in the QEMU system emulation environment. You can download a BSP to any location of your choice.

BSPs are not included in the PetaLinux tools installer and need to be downloaded and installed separately. PetaLinux BSP packages are available on the Xilinx.com Download Center. There is a README in each BSP which explains the details of the BSP.

Note: Download only the BSPs you need.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- PetaLinux BSP is downloaded. You can download PetaLinux BSP from PetaLinux Downloads.
- PetaLinux Working Environment Setup is completed. For more details, see PetaLinux Working Environment Setup.

Create a Project from a BSP

1. Change to the directory under which you want PetaLinux projects to be created. For example, if you want to create projects under `/home/user`:

   ```
   $ cd /home/user
   ```

2. Run `petalinux-create` command on the command console:

   ```
   petalinux-create -t project -s <path-to-bsp>
   ```

The board being referenced is based on the BSP installed. The output is similar to the following output:

```
INFO: Create project:
INFO: Projects:
INFO:    * xilinx-zcu102-v<petalinux-version>
INFO: has been successfully installed to /home/user/
INFO: New project successfully created in /home/user/
```

In the above example, when the command runs, it tells you the projects that are extracted and installed from the BSP. If the specified location is on the Network File System (NFS), it changes the TMPDIR to `/tmp/<projname-timestamp-id>`; otherwise, it is set to `$PROOT/build/tmp`

> **CAUTION!** *Do not create the symbolic link to a local file system on the NFS file system. You can not use NFS for the location of the 'tmp' directory in the build; it will fail.*

***Note*:** PetaLinux requires a minimum of 50 GB and a maximum of 100 GB /tmp space to build the project successfully.

If `/tmp/<projname_timestamp>` is also on NFS, then it throws an error. You can change TMPDIR while creating the PetaLinux project using the following command

```
petalinux-create -t project -s <PATH_TO_PETALINUX_PROJECT_BSP> --tmpdir
<TMPDIR PATH>
```

Alternatively, you can also create/modify it at any time using **petalinux-config → Yocto-settings**. Do not configure the same location as TMPDIR for two different PetaLinux projects as it can cause build errors.

Run `ls` from `/home/user` to see the created project(s). For more details on the structure of a PetaLinux project, see Appendix B: PetaLinux Project Structure.

> **CAUTION!** *Do not create PetaLinux projects in the tool install area and do not use the tool install area as a tmp build area.*

To create a project using XRT, click https://xilinx.github.io/XRT/master/html/yocto.html#create-petalinux-project-with-xrt

Troubleshooting

This section describes some common issues you may experience while installing PetaLinux BSP.

Table 6: **PetaLinux BSP Installation Troubleshooting**

Problem / Error Message	Description and Solution
`petalinux-create: command not found`	**Problem Description**: This message indicates that it is unable to find `petalinux-create` command and therefore it cannot proceed with BSP installation. **Solution**: You have to setup your environment for PetaLinux tools. For more information, see the PetaLinux Working Environment Setup.

Configuring Hardware Platform with Vivado Design Suite

This section describes how to make a hardware platform ready for PetaLinux.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- Vivado® Design Suite is installed. You can download Vivado Design Suite from Vivado Design Tool Downloads.
- You have set up the Vivado tools working environment. If you have not, source the appropriate settings scripts as follows:

  ```
  $ source <path-to-installed-Xilinx-Vivado>/settings64.sh
  ```

  ***Note*:** You can have Vivado tools set up on a different machine; it is not necessary to have PetaLinux and Vivado tools set up on the same machine.
- You are familiar with the Vivado Design Suite and the Vitis™ software development platform. For more information, see the *Vitis Unified Software Platform Documentation: Embedded Software Development* (UG1400).

Configure a Hardware Platform for Linux

You can create your own hardware platform with Vivado® tools. Regardless of how the hardware platform is created and configured, there are a small number of hardware IP and software platform configuration changes required to make the hardware platform Linux ready. These are described below:

Zynq UltraScale+ MPSoC and Versal ACAP

The following is a list of hardware requirements for a Zynq® UltraScale+™ MPSoC and a Versal™ ACAP hardware project to boot Linux:

- External memory of, at least, 64 MB (required).
- UART for serial console (required).
- Non-volatile memory, for example, QSPI flash and SD/MMC. This memory is optional but only JTAG boot can work.
- Ethernet (optional, essential for network access).

IMPORTANT! *If soft IP with interrupt or external PHY device with interrupt is used, ensure the interrupt signal is connected.*

Zynq-7000 Devices

The following is a list of hardware requirements for a Zynq-7000hardware project to boot Linux:

- One Triple Timer Counter (TTC) (required).

  IMPORTANT! *If multiple TTCs are enabled, the Zynq-7000 Linux kernel uses the first TTC block from the device tree. Please make sure the TTC is not used by others.*

- External memory controller with at least 32 MB of memory (required).
- UART for serial console (required).
- Non-volatile memory, for example, QSPI flash and SD/MMC. This memory is optional but only JTAG boot can work.
- Ethernet (optional, essential for network access).

IMPORTANT! *If soft IP is used, ensure the interrupt signal is connected. If soft IP with interrupt or external PHY device with interrupt is used, ensure the interrupt signal is connected.*

MicroBlaze processors (AXI)

The following is a list of requirements for a MicroBlaze™ hardware project to boot Linux:

- IP core check list:
  - External memory controller with at least 32 MB of memory (required)
  - Dual channel timer with interrupt connected (required)
  - UART with interrupt connected for serial console (required)
  - Non-volatile memory such as Linear Flash or SPI Flash (required)
  - Ethernet with interrupt connected (optional, but required for network access)

- MicroBlaze processor configuration:
  - MicroBlaze processors with MMU support by selecting either Linux with MMU or low-end Linux with MMU configuration template in the MicroBlaze configuration wizard.

    ***Note*:** Do not disable any instruction set related options that are enabled by the template, unless you understand the implications of such a change.
  - MicroBlaze processor initial boot loader fs-boot needs minimum 4 KB of block RAM for parallel flash and 8 KB for SPI flash when the system boots from non-volatile memory.

***Note*:** PetaLinux will only support 32-bit MicroBlaze processors.

Exporting Hardware Platform to PetaLinux Project

This section describes how to export a hardware platform to a PetaLinux project.

Prerequisites

This section assumes that a hardware platform is created with the Vivado Design Suite. For more information, see Configuring Hardware Platform with Vivado Design Suite.

Exporting Hardware Platform

After you have configured your hardware project, the PetaLinux project requires a hardware description file (`.xsa` file) with information about the processing system. You can get the hardware description file by running Export Hardware from the Vivado® Design Suite.

During project initialization (or update), PetaLinux generates a device tree source file, U-Boot configuration header files (MicroBlaze processors only), and enables the Linux kernel drivers (MicroBlaze processors only) based on the hardware description file. These details are discussed in Appendix B: PetaLinux Project Structure.

For Zynq® UltraScale+™ MPSoC platform, you need to boot with the Platform Management Unit (PMU) firmware and ATF. See Appendix C: Generating Boot Components for building PMU firmware and ATF. If you want a first stage boot loader (FSBL) built for Cortex®-R5F boot, you have to build it with the Vitis™ software platform because the FSBL built with PetaLinux tools is for Cortex-A53 boot. For details on how to build the FSBL for Cortex-R5F with the Vitis software platform, see the *Zynq UltraScale+ MPSoC: Software Developers Guide* (UG1137).

For Versal platforms, boot with PLM, PSM and ATF. See Appendix C: Generating Boot Components for building the PLM, PSM, and ATF.

Creating an Empty Project from a Template

This section describes how to create an empty project from a template. Projects created from templates must be configured to an actual hardware instance before they can be built.

Prerequisites

This section assumes that the PetaLinux working environment setup is complete. For more information, see PetaLinux Working Environment Setup.

Create New Project

The `petalinux-create` command is used to create a new PetaLinux project:

```
$ petalinux-create --type project --template <PLATFORM> --name
<PROJECT_NAME>
```

The parameters are as follows:

- `--template <PLATFORM>` - The following platform types are supported:
  - `versal` (for Versal ACAP)
  - `zynqMP` (for Zynq UltraScale+ MPSoC)
  - `zynq` (for Zynq-7000 devices)
  - `microblaze` (for MicroBlaze™ processor)

    Note: The MicroBlaze option cannot be used along with Zynq-7000 devices or Zynq UltraScale+ designs in the Programmable Logic (PL).
- `--name <PROJECT_NAME>` - The name of the project you are building.

  This command creates a new PetaLinux project folder from a default template.

For more information, see Importing Hardware Configuration.

XILINX

Chapter 4

Configuring and Building

Version Control

This section details about version management/control in PetaLinux project.

Prerequisites

This section assumes that you have created a new PetaLinux project or have an existing PetaLinux project. See Creating an Empty Project from a Template for more information on creating a PetaLinux project.

Version Control

You can have version control over your PetaLinux project directory `<plnx-proj-root>`, excluding the following:

- `<plnx-proj-root>/.petalinux`
- `<plnx-proj-root>/!.petalinux/metadata`
- `<plnx-proj-root>/build/`
- `<plnx-proj-root>/images/linux`
- `<plnx-proj-root>/pre-built/linux`
- `<plnx-proj-root>/components/plnx-workspace/`
- `<plnx-proj-root>/components/yocto/`
- `<plnx-proj-root>/*/*/config.old`
- `<plnx-proj-root>/*/*/rootfs_config.old`
- `<plnx-proj-root>/*.o`
- `<plnx-proj-root>/*.log`
- `<plnx-proj-root>/*.jou`

By default, these files are added into `.gitignore` while creating the project.

Note: A PetaLinux project should be cleaned using `petalinux-build -x mrproper` before submitting to the source control.

Note: In concurrent development, TMPDIR in `petalinux-config` should be unique for each user.

Importing Hardware Configuration

This section explains the process of updating an existing/newly created PetaLinux project with a new hardware configuration. This enables you to make the PetaLinux tools software platform ready for building a Linux system, customized to your new hardware platform.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have exported the hardware platform and .xsa file is generated. For more information, see Exporting Hardware Platform.
- You have created a new PetaLinux project or have an existing PetaLinux project. For more information on creating a PetaLinux project, see Creating an Empty Project from a Template.

Steps to Import Hardware Configuration

Steps to import hardware configuration are:

1. Change into the directory of your PetaLinux project.

   ```
   $ cd <plnx-proj-root>
   ```

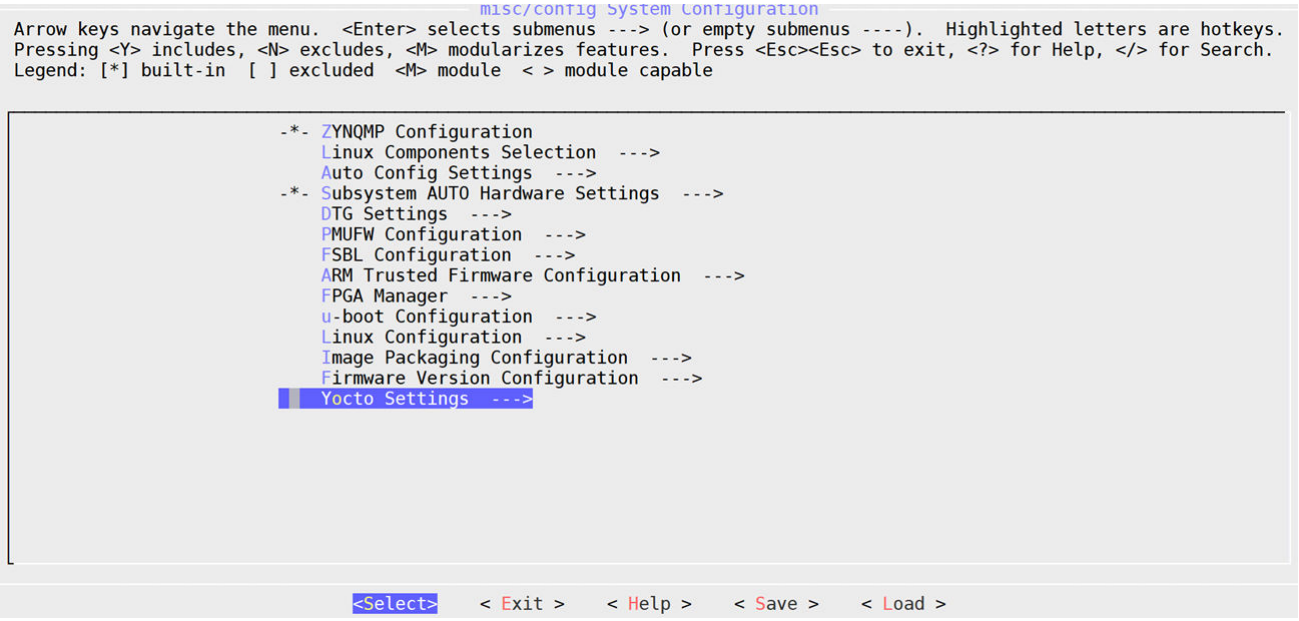
2. Import the hardware description with `petalinux-config` command by giving the path of the directory containing the .xsa file as follows:

   ```
   petalinux-config --get-hw-description <PATH-TO-XSA Directory>/<PATH-TO-XSA>
   ```

   Note: Changing the XSA file in the `<PATH-TO-XSA directory>` later gives an *INFO: Seems like your hardware design:<PATH-TO_XSA Directory>/system.xsa has changed* warning for all subsequent executions of the `petalinux-config`/`petalinux-build` commands. This means that your xsa has changed. To use the latest XSA, run `petalinux-config --get-hw-description <PATH-TO-XSA Directory>/--get-hw-description=<PATH-TO_XSA Directory>/--get-hw-description=<PATH-TO-XSA>` again.

This launches the top system configuration menu. When the `petalinux-config --get-hw-description` command runs for the PetaLinux project, the tool detects changes in the system primary hardware candidates:

Figure 1: **System Configuration Menu**



Ensure **DTG Settings → (template) MACHINE_NAME** is selected and change the template to any of the below mentioned possible values.

Table 7: **BSP and Machine Names**

BSP	Machine
ZCU102	zcu102-rev1.0
ZCU104	zcu104-revc
ZCU106	zcu106-reva
ZCU111	zcu111-reva
ZCU1275	zcu1275-revb
ZCU1285	zcu1285-reva
ZCU216	zcu216-reva
ZCU208	zcu208-reva
ZCU208-SDFEC	zcu208-reva
ULTRA96	avnet-ultra96-rev1
ZCU100	zcu100-revc
ZC702	zc702
ZC706	zc706
ZEDBOARD	zedboard
AC701	ac701-full
KC705	kc705-full
KCU105	kcu105
VCU118	vcu118-rev2.0

Table 7: **BSP and Machine Names** *(cont'd)*

BSP	Machine
SP701	sp701-rev1.0
VCK190	versal-vck190-reva-x-ebm-01-reva
VMK180	versal-vmk180-reva-x-ebm-01-reva

***Note*:** These values are applicable only to Xilinx® evaluation boards. For custom boards, do not change the configurations.

TIP: *For details on the PetaLinux project structure, see Appendix B: PetaLinux Project Structure.*

CAUTION! *When a PetaLinux project is created on NFS, `petalinux-create` automatically changes the TMPDIR to `/tmp/<projname-timestamp-id>`. If `/tmp` is on NFS, it throws an error. you can change the TMPDIR to local storage while creating the PetaLinux project by running `petalinux-create -t project -s <PATH_TO_PETALINUX_PROJECT_BSP> --tmpdir <TMPDIR PATH>`, or selecting **petalinux-config → Yocto-settings → TMPDIR**. Do not configure the same location as TMPDIR for two different PetaLinux projects. This can cause build errors. If TMPDIR is at `/tmp/.`, deleting the project does not work. To delete the project, run `petalinux-build -x mrproper`.*

Ensure **Subsystem AUTO Hardware Settings** is selected, and go into the menu which is similar to the following:

```
Subsystem AUTO Hardware Settings
System Processor (psu_cortexa53_0)  --->
Memory Settings  --->
Serial Settings  --->
Ethernet Settings  --->
Flash Settings  --->
SD/SDIO Settings  --->
RTC Settings  --->
```

The **Subsystem AUTO Hardware Settings →** menu allows customizing system wide hardware settings.

This step can take a few minutes to complete because the tool parses the hardware description file for hardware information required to update the device tree, PetaLinux U-Boot configuration files (only for Microblaze) and the kernel config files(only for Microblaze) based on the “Auto Config Settings --->” and “Subsystem AUTO Hardware Settings --->” settings.

***Note*:** For more details on the Auto Config Settings menu, see the Auto Config Settings.

The `--silentconfig` option allows you to reuse a prior configuration. Old configurations have the file name `CONFIG.old` within the directory containing the specified component for unattended updates.

Build System Image

Prerequisites

This section assumes that you have PetaLinux tools software platform ready for building a Linux system that is customized to your hardware platform. For more information, see Importing Hardware Configuration.

Steps to Build PetaLinux System Image

1. Change into the directory of your PetaLinux project.

   ```
   $ cd <plnx-proj-root>
   ```

2. Run `petalinux-build` to build the system image:

   ```
   $ petalinux-build
   ```

   This step generates a device tree DTB file, a first stage boot loader (for Zynq® devices, Zynq® UltraScale+™ MPSoC, and MicroBlaze™), PLM (for Versal™ ACAP), PSM (for Versal ACAP) and ATF (for Zynq UltraScale+ MPSoC and Versal ACAP), U-Boot, the Linux kernel, a root file system image. and the boot script (`boot.scr` which does not exist for MicroBlaze). Finally, it generates the necessary boot images.

3. The compilation progress shows on the console. Wait until the compilation finishes.

TIP: *A detailed compilation log is in `<plnx-proj-root>/build/build.log`.*

When the build finishes, the generated images are stored in the `<plnx-proj-root>/images/linux` or `/tftpboot` directories.

The console shows the compilation progress. For example:

```
$ petalinux-build
[INFO] Sourcing buildtools
[INFO] Building project
[INFO] Sourcing build environment
[INFO] Generating workspace directory
INFO: bitbake petalinux-image-minimal
NOTE: Started PRServer with DBfile: <plnx-proj-root>/xilinx-vck190-2021.1/
build/cache/prserv.sqlite3, IP: 127.0.0.1, PORT: 43231, PID: 5719
Loading cache: 100%
|
                                                                  | ETA:
--:--:--
Loaded 0 entries from dependency cache.
Parsing recipes: 100% |
##################################################################################
##########################################################| Time: 0:01:37
Parsing of 3477 .bb files complete (0 cached, 3477 parsed). 5112 targets,
```

```
243 skipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
NOTE: Fetching uninative binary shim file:/<plnx-proj-root>/xilinx-
vck190-2021.1/components/yocto/downloads/uninative/
5ec5a9276046e7eceeac749a18b175667384e1f445cd4526300a41404d985a5b/x86_64-
nativesdk-
libc.tar.xz;sha256sum=5ec5a9276046e7eceeac749a18b175667384e1f445cd4526300a41
404d985a5b (will check PREMIRRORS first)
Initialising tasks: 100% |
##################################################################################
##########################################################| Time: 0:00:14
Checking sstate mirror object availability: 100% |
##################################################################################
###################################| Time: 0:00:20
Sstate summary: Wanted 1441 Found 1118 Missed 323 Current 0 (77% match, 0%
complete)
NOTE: Executing Tasks
NOTE: Tasks Summary: Attempted 4363 tasks of which 3067 didn't need to be
rerun and all succeeded.
INFO: Failed to copy built images to tftp dir: /tftpboot
[INFO] Successfully built project
```

Default Image

When you run `petalinux-build`, it generates FIT images for Versal™ ACAP, Zynq® UltraScale+™ MPSoC, Zynq-7000 devices, and MicroBlaze™ platforms. The RAM disk image `rootfs.cpio.gz.u-boot` is also generated.

The full compilation log `build.log` is stored in the build sub-directory of your PetaLinux project. The final image, `<plnx-proj-root>/images/linux/image.ub`, is a FIT image. The kernel image (including RootFS) is `Image` for Zynq® UltraScale+™ MPSoC and Versal™ platform, `zImage` for Zynq-7000 devices, and `image.elf` for MicroBlaze processors. The build images are located in the `<plnx-proj-root>/images/linux` directory. A copy is also placed in the `/tftpboot` directory if the option is enabled in the system-level configuration for the PetaLinux project.

IMPORTANT! *By default, besides the kernel, RootFS, and U-Boot, the PetaLinux project is configured to generate and build the first stage boot loader. For more details on the auto generated first stage boot loader, see Appendix C: Generating Boot Components.*

Troubleshooting

This section describes some common issues/warnings you may experience while building a PetaLinux image.

Warnings/Errors

Warning or Error Message	Description	Solution
Skipping recipe linux-xlnx as it does not produce a package with the same name	It appears if the provided recipe name does not match with the packages provided by it, for example, if linux-xlnx provides kernel-image, kernel-base, kernel-dev, and kernel-modules packages and these does not match with the name linux-xlnx which was in workspace directory.	You can ignore this warning message.
<package-name> do_package: Could not copy license file `<plnx-proj-root>/components/yocto/layers/core/meta/files/common-licenses/` to `/opt/pkg/petalinux/build/tmp/work/<machine-name>-xilinx-linux/image/usr/share/licenses/<package-name>/COPYING.MIT` `[Errno 1] Operation not permitted:`	When the tool is installed, all license files in `<plnx-proj-root>/components/yocto//layers/core/meta/files/common-licenses/` have 644 permissions. Therefore, they are readable by others but not writable.	• Method 1: Manually modify permissions of the license files coming from the layers `$ chmod 666 <plnx-proj-root>/components/yocto/layers/core/meta/files/common-licenses/*` When creating the hard link, you have write permissions to the source of the link. • Method 2: Disable hard linking protection on the kernel `$ sysctl fs.protected_hardlinks=0` The kernel does not allows the source to be writable by the current user when creating the hard link. • Method 3: Set the following Yocto variables in `<plnx-proj-root>/meta-user/conf/petalinuxbsp.conf` `LICENSE_CREATE_PACKAGE_forcevariable = "0" SIGGEN_LOCKEDSIGS_TASKSIG_CHECK = "none"` The build system does not try to create the link and the license is not on the final image. ***Note***: It is recommended to use the troubleshooting solutions specified in method 1 and method 3 because method 2 requires all the users to have sudo access.

Generate Boot Image for Versal ACAP

This section is for Versal™ ACAP only and describes how to generate `BOOT.BIN` for Versal ACAP.

Prerequisties

This section assumes that you have built PetaLinux system image. For more information, see Build System Image.

Generate Boot Image

A boot image usually contains a PDI file (imported from hardware design), PLM, PSM firmware, Arm® trusted firmware, U-Boot, and DTB.

Execute the following command to generate the boot image in .bin format:

```
$ petalinux-package --boot --u-boot
```

This generates `BOOT.BIN`, `BOOT_bh.bin`, and `qemu_boot.img` in `images/linux` directory. The default DTB load address is `0x1000`. For more information, see *Bootgen User Guide* (UG1283).

To change the DTB load address, use this command:

```
$ petalinux-package --boot --plm --psmfw --u-boot --dtb --load
<load_address>
```

This generates a `BOOT.BIN` with a specified load address for DTB. Ensure that you have also changed the load address for DTB in `petalinux-config` and in the U-Boot menuconfig.

Note: The files `versal-qemu-multiarch-pmc.dtb` and `versal-qemu-multiarch-ps.dtb` are QEMU DTBS and required to boot multiarch QEMU.

For detailed usage, see the `--help` option and petalinux-package --boot.

Generate Boot Image for Zynq UltraScale+ MPSoC

This section is for Zynq® UltraScale+™ MPSoC only and describes how to generate `BOOT.BIN` for Zynq UltraScale+ MPSoC.

Prerequisites

This section assumes that you have built the PetaLinux system image. For more information, see Build System Image.

Generate Boot Image

The boot image can be put into Flash or SD card. When you power on the board, it can boot from the boot image. A boot image usually contains a first stage boot loader image, FPGA bitstream (optional), PMU firmware, ATF, and U-Boot.

Execute the following command to generate the boot image in `.BIN` format.

```
$ petalinux-package --boot --u-boot --format BIN
[INFO] Sourcing buildtools
INFO: Getting system flash information...
INFO: File in BOOT BIN: "<plnx-proj-root>/xilinx-zcu102-2021.1/images/linux/
zynqmp_fsbl.elf"
INFO: File in BOOT BIN: "<plnx-proj-root>/xilinx-zcu102-2021.1/images/linux/
pmufw.elf"
INFO: File in BOOT BIN: "<plnx-proj-root>/xilinx-zcu102-2021.1/project-
spec/hw-description/project_1.bit"
INFO: File in BOOT BIN: "<plnx-proj-root>/xilinx-zcu102-2021.1/images/linux/
bl31.elf"
INFO: File in BOOT BIN: "<plnx-proj-root>/xilinx-zcu102-2021.1/images/linux/
system.dtb"
INFO: File in BOOT BIN: "<plnx-proj-root>/xilinx-zcu102-2021.1/images/
linux/u-boot.elf"
INFO: Generating zynqmp binary package BOOT.BIN...


****** Xilinx Bootgen v2021.1
  **** Build date : Mar 30 2021-16:20:57
    ** Copyright 1986-2021 Xilinx, Inc. All Rights Reserved.


[INFO]   : Bootimage generated successfully

INFO: Binary is ready.
WARNING: Unable to access the TFTPBOOT folder /tftpboot!!!
WARNING: Skip file copy to TFTPBOOT folder!!!
```

For detailed usage, see the `--help` option or petalinux-package --boot.

Generate Boot Image for Zynq-7000 Devices

This section is for Zynq®-7000 devices only and describes how to generate `BOOT.BIN`.

Prerequisites

This section assumes that you have built the PetaLinux system image. For more information, see Build System Image.

Generate Boot Image

The boot image can be put into Flash or SD card. When you power on the board, it can boot from the boot image. A boot image usually contains a first stage boot loader image, FPGA bitstream (optional) and U-Boot.

Follow the step below to generate the boot image in `.BIN` format.

```
$ petalinux-package --boot --fsbl <FSBL image> --fpga <FPGA bitstream> --u-boot
```

For detailed usage, see the `--help` option or petalinux-package --boot.

Generate Boot Image for MicroBlaze Processor

This section is for MicroBlaze™ processor only and describes how to generate an MCS file for MicroBlaze processor.

Prerequisites

This section assumes that you have built the PetaLinux system image. For more information, see Build System Image.

- To generate an MCS boot file, you must install the Vivado® Design Suite. You can download the Vivado Design Suite from Vivado Design Tool Downloads.
- You have set up the Vivado tools working environment. If you have not, source the appropriate settings scripts as follows:

  ```
  $ source <installed-vivado-path>/settings64.sh
  ```

Generate Boot Image

Execute the following command to generate MCS boot file for MicroBlaze processors.

```
$ petalinux-package --boot --fpga <FPGA bitstream> --u-boot --kernel
```

It generates `boot.mcs` in your working directory and it copies it to the `<plnx-proj-root>/images/linux/` directory. With the above command, the MCS file contains FPGA bitstream, fs-boot, U-Boot, and kernel image `image.ub`.

Command to generate the MCS file with fs-boot and FPGA bitstream *only*:

```
$ petalinux-package --boot --fpga <FPGA bitstream>
```

Command to generate the MCS file with FPGA bitstream, fs-boot, and U-Boot:

```
$ petalinux-package --boot --fpga <FPGA bitstream> --u-boot
```

For detailed usage, see the `--help` option or petalinux-package --boot.

***Note*:** PetaLinux will only support 32-bit MicroBlaze processors.

Modify Bitstream File for MicroBlaze Processor

Prerequisites

This section assumes that you have built the PetaLinux system image and FSBL. For more information, see Build System Image.

Modify Bitstream

Execute the following command to modify the bitstream file for MicroBlaze™ processor.

```
$ petalinux-package --boot --fpga <FPGA bitstream> --fsbl <FSBL_ELF> --
format DOWNLOAD.BIT
```

This generates `download.bit` in the `<plnx-proj-root>images/linux/` directory. With the above command, it merges the fs-boot into the FPGA bitstream by mapping the ELF data onto the memory map information (MMI) for the block RAMs in the design. For detailed usage, see the `--help` option or see petalinux-package --boot.

***Note*:** PetaLinux only supports 32-bit MicroBlaze processors.

Build Optimizations

This section describes the build optimization techniques with the PetaLinux tools.

Deselecting Default Components

You can deselect default components, if they are not needed. To disable the FSBL and PMU firmware for Zynq® UltraScale+™ MPSoC, deselect the following options in **petalinux-config → Linux Components Selection**.

- **FSBL → [ ] First Stage Boot Loader**
- **PMUFW → [ ] PMU Firmware**

To disable the PLM and PSM firmware for Versal™, deselect the following option in **petalinux-config → Linux Components Selection**.

- **PLM → [ ] PLM**
- **PSM → [ ] PSM Firmware**

Deselecting these components removes these components from the default build flow.

***Note*:** If the FSBL, PMU firmware, PLM firmware, and PSM firmware are not built with PetaLinux, they must be built in the Vitis™ software platform.

Local Mirror Servers

You can set internal mirrors on the NFS or web server which can speed up the builds. By default, PetaLinux uses sstate-cache and download mirrors from petalinux.xilinx.com. Follow these steps to work with local, NFS, or the internal webserver copy of sstate in PetaLinux. You can download the sstate from the download area along with PetaLinux.

Table 8: **Local Mirror Servers**

Server	Description
downloads	Source of download files are available in http://petalinux.xilinx.com/sswreleases/rel-v2020/downloads
aarch64	sstate mirrors for Zynq UltraScale+ MPSoC/RFSoC and Versal™ devices
arm	sstate mirrors for Zynq-7000
mb-full	sstate mirrors for MicroBlaze™ processors (full)
mb-lite	sstate mirrors for MicroBlaze processors (lite)

Source Mirrors

You can set source mirrors through **petalinux-config → Yocto-settings → Add pre-mirror URL**. Select `file://<local downloads path>` for all projects. Save the configuration to use the download mirrors and verify the changes in `build/conf/plnxtool.conf`.

Reduce Build Time

To reduce the build time by disabling the network sstate feeds, de-select the **petalinux-config → Yocto Settings → Enable Network sstate feeds**.

Sstate Feeds

You can set sstate feeds through `petalinux-config`.

- sstate feeds on NFS: Go to **petalinux-config → Yocto Settings → Local sstate feeds settings** and enter the full path of the sstate directory. By enabling this option, you can point to your own shared state which is available at a NFS/local mount point.

  For example, to enable, use `/opt/petalinux/sstate-cache_2020/aarch64`.

- sstate feeds on webserver: Go to **petalinux-config → Yocto Settings → Enable Network sstate feeds → Network sstate feeds URL** and enter the URL for sstate feeds.

  For more information, see How to reduce build time using SSTATE CACHE

***Note*:** This is set to http://petalinux.xilinx.com/sswreleases/rel-v2021/aarch64/sstate-cache, by default.

Booting and Packaging

Packaging Prebuilt Images

This section describes how to package newly built images into a prebuilt directory.

This step is typically done when you want to distribute your project as a BSP to other users.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- For Zynq® UltraScale+™ MPSoC, Zynq-7000 devices, and Versal™ devices, you have generated the boot image. For more information, see Generate Boot Image for Zynq UltraScale+ MPSoC.
- For MicroBlaze™ processors, you have generated the system image. For more information, see Build System Image.

Steps to Package Prebuilt Image

1. Change into the root directory of your project.

```
$ cd <plnx-proj-root>
```

2. Use `petalinux-package --prebuilt` to package the prebuilt images.

```
$ petalinux-package --prebuilt --fpga <FPGA bitstream>
```

For detailed usage, see the `--help` option or the petalinux-package --prebuilt.

Using petalinux-boot Command with Prebuilt Images

You can boot a PetaLinux image using the `petalinux-boot` command. Use the `--qemu` option for software emulation (QEMU) and `--jtag` option to boot on hardware. This section describes different boot levels for prebuilt option.

Prerequisites

This section assumes that you have packaged prebuilt images. For more information, see Packaging Prebuilt Images.

Boot Levels for Prebuilt Option

`--prebuilt <BOOT_LEVEL>` boots prebuilt images (override all settings). Supported boot levels are 1 to 3. The command for JTAG boot:

```
petalinux-boot --jtag --prebuilt <BOOT_LEVEL> --hw_server-url hostname:3121
```

The command for the QEMU boot is as follows:

```
petalinux-boot --qemu --prebuilt <BOOT_LEVEL>
```

***Note*:** The QEMU boot does not support BOOT_LEVEL 1.

- Level 1: Download the prebuilt FPGA bitstream.
  - It boots FSBL and PMU firmware for Zynq® UltraScale+™ MPSoC.
  - It boots FSBL for Zynq-7000 devices.
- Level 2: Download the prebuilt FPGA bitstream and boot the prebuilt U-Boot.
  - For Zynq-7000 devices: It boots FSBL before booting U-Boot.
  - For Zynq UltraScale+ MPSoC: It boots PMU firmware, FSBL, and ATF before booting U-Boot.
  - For Versal™ devices: It loads the required cdo files from .pdi file, PLM, PSM firmware, and ATF before booting U-Boot.
- Level 3:
  - For MicroBlaze™ processors: Downloads the prebuilt FPGA bitstream and boots the prebuilt kernel image on target.
  - For Zynq-7000 devices: Downloads the prebuilt FPGA bitstream and FSBL, boots the prebuilt U-Boot, and boots the prebuilt kernel on target.

- For Zynq UltraScale+ MPSoC: Downloads PMU firmware, prebuilt FSBL, prebuilt kernel, prebuilt FPGA bitstream, linux-boot.elf, DTB, and the prebuilt ATF on target.
- For Versal devices: It loads the required cdo files from .pdi file, PLM, PSM firmware, ATF, and U-Boot before booting Kernel.

Example to show the usage of boot level for prebuilt option:

```
$ petalinux-boot --jtag --prebuilt 3
```

Booting a PetaLinux Image on QEMU

This section describes how to boot a PetaLinux image under software emulation (QEMU) environment.

For details on Xilinx® IP Models supported by QEMU, see Appendix E: Xilinx IP Models Supported by QEMU.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have a PetaLinux system image by either installing a PetaLinux BSP (see Project Creation Using PetaLinux BSP) or by building your own PetaLinux project (see Build System Image).
- If you are going to use `--prebuilt` option for QEMU boot, you need to have prebuilt images packaged. For more information, see Packaging Prebuilt Images.
- To boot QEMU for Versal™ ACAP, you should generate BOOT.BIN first. Please see Generate Boot Image for Versal ACAP for more information on how to generate BOOT.BIN.

IMPORTANT! *Unless otherwise indicated, the PetaLinux tool command must be run within a project directory (`<plnx-proj-root>`).*

Steps to Boot a PetaLinux Image on QEMU

PetaLinux provides QEMU support to enable testing of PetaLinux software image in a simulated environment without any hardware.

Use the following steps to test the PetaLinux reference design with QEMU:

1. Change to your project directory and boot the prebuilt Linux kernel image:

```
$ petalinux-boot --qemu --prebuilt 3
```

If you do not wish to use prebuilt capability for QEMU boot, see the Additional Options for Booting on QEMU.

The `--qemu` option tells `petalinux-boot` to boot QEMU instead of real hardware.

- The `--prebuilt 1` performs a Level 1 (FPGA bitstream) boot. This option is not valid for QEMU.
- A level 2 boot includes U-Boot.
- A level 3 boot includes a prebuilt Linux image.

To know more about different boot levels for prebuilt option, see Using petalinux-boot Command with Prebuilt Images.

An example of the kernel boot log messages displayed on the console during successful Linux boot is as follows:

```
  $.petalinux-boot --qemu --prebuilt 3
[INFO] Sourcing buildtools
Image resized.
INFO: No DTB has been specified, use the default one "/<plnx-proj-root>/
xilinx-zcu102-2021.1/pre-built/linux/images/system.dtb".
INFO: No DTB has been specified, use the default one "/<plnx-proj-root>/
xilinx-zcu102-2021.1/pre-built/linux/images/system.dtb".
INFO: Starting microblaze QEMU
INFO: Starting the above QEMU command in the background
INFO:  qemu-system-microblazeel -M microblaze-fdt   -serial mon:stdio -
serial /dev/null -display none -kernel //xilinx-zcu102-2021.1/pre-built/
linux/images/pmu_rom_qemu_sha3.elf -device loader,file=//xilinx-
zcu102-2021.1/pre-built/linux/images/pmufw.elf      -hw-dtb //xilinx-
zcu102-2021.1/pre-built/linux/images/zynqmp-qemu-multiarch-pmu.dtb -
machine-path /tmp/tmp.5fycBfKBph -device
loader,addr=0xfd1a0074,data=0x1011003,data-len=4 -device
loader,addr=0xfd1a007C,data=0x1010f03,data-len=4
qemu-system-microblazeel: Failed to connect socket /tmp/tmp.5fycBfKBph/
qemu-rport-_pmu@0: No such file or directory
qemu-system-microblazeel: info: QEMU waiting for connection on:
disconnected:unix:/tmp/tmp.5fycBfKBph/qemu-rport-_pmu@0,server
INFO: TCP PORT is free
INFO: Starting aarch64 QEMU
INFO:  qemu-system-aarch64 -M arm-generic-fdt   -serial mon:stdio -
serial /dev/null -display none -device loader,file=//xilinx-
zcu102-2021.1/pre-built/linux/images/bl31.elf,cpu-num=0 -device
loader,file=//xilinx-zcu102-2021.1/pre-built/linux/images/
ramdisk.cpio.gz.u-boot,addr=0x04000000,force-raw -device loader,file=//
xilinx-zcu102-2021.1/pre-built/linux/images/u-boot.elf -device
loader,file=//xilinx-zcu102-2021.1/pre-built/linux/images/
Image,addr=0x00200000,force-raw -device loader,file=//xilinx-
zcu102-2021.1/pre-built/linux/images/system.dtb,addr=0x00100000,force-
raw -device loader,file=//xilinx-zcu102-2021.1/pre-built/linux/images/
boot.scr,addr=0x20000000,force-raw -gdb tcp::9009   -net nic -net nic -
net nic -net nic,netdev=eth0 -netdev user,id=eth0,tftp=/tftpboot   -hw-
dtb //xilinx-zcu102-2021.1/pre-built/linux/images/zynqmp-qemu-multiarch-
arm.dtb -machine-path /tmp/tmp.5fycBfKBph -global xlnx,zynqmp-boot.cpu-
num=0 -global xlnx,zynqmp-boot.use-pmufw=true  -drive
if=sd,format=raw,index=1,file=//xilinx-zcu102-2021.1/pre-built/linux/
images/rootfs.ext4   -m 4G
QEMU 5.1.0 monitor - type 'help' for more information
(qemu) qemu-system-aarch64: warning: hub 0 is not connected to host
network
```

```
PMU Firmware 2021.1    May 31 2021   18:37:19
PMU_ROM Version: xpbr-v8.1.0-0
NOTICE:  ATF running on XCZUUNKN/QEMU v4/RTL0.0 at 0xfffea000
NOTICE:  BL31: v2.4(release):v1.1-7609-g851523ea2
NOTICE:  BL31: Built : 08:27:07, Apr 28 2021


U-Boot 2021.01 (May 28 2021 - 10:52:42 +0000)

Model: ZynqMP ZCU102 Rev1.0
Board: Xilinx ZynqMP
DRAM:  4 GiB
PMUFW:    v1.1
EL Level:    EL2
Chip ID:    unknown
NAND:  0 MiB
MMC:   mmc@ff170000: 0
In:    serial
Out:   serial
Err:   serial
Bootmode: JTAG_MODE
Reset reason:
Net:
ZYNQ GEM: ff0e0000, mdio bus ff0e0000, phyaddr 12, interface rgmii-id

Warning: ethernet@ff0e0000 (eth0) using random MAC address -
d6:ec:61:e0:83:3a
eth0: ethernet@ff0e0000
Hit any key to stop autoboot:  0
JTAG: Trying to boot script at 20000000
## Executing script at 20000000
Trying to load boot images from jtag
## Loading init Ramdisk from Legacy Image at 04000000 ...
   Image Name:   petalinux-initramfs-image-zynqmp
   Created:      2011-04-05  23:00:00 UTC
   Image Type:   AArch64 Linux RAMDisk Image (uncompressed)
   Data Size:    5123986 Bytes = 4.9 MiB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 00100000
   Booting using the fdt blob at 0x100000
   Loading Ramdisk to 7d82d000, end 7dd0ff92 ... OK
   Loading Device Tree to 000000007d81c000, end 000000007d82cd77 ... OK

Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0x0000000000 [0x410fd034]
[    0.000000] Linux version 5.10.0-xilinx-v2020.2 (oe-user@oe-host)
(aarch64-xilinx-linux-gcc (GCC) 10.2.0, GNU ld (GNU Binutils) 2.35.1) #1
SMP Fri May 28 08:10:27 UTC 2021
[    0.000000] Machine model: ZynqMP ZCU102 Rev1.0
[    0.000000] earlycon: cdns0 at MMIO 0x00000000ff000000 (options
'115200n8')
[    0.000000] printk: bootconsole [cdns0] enabled
[    0.000000] efi: UEFI not found.
[    0.000000] cma: Reserved 256 MiB at 0x000000006d800000
[    0.000000] Zone ranges:
[    0.000000]   DMA32    [mem 0x0000000000000000-0x00000000ffffffff]
[    0.000000]   Normal   [mem 0x0000000100000000-0x000000087fffffff]
[    0.000000] Movable zone start for each node
[    0.000000] Early memory node ranges
[    0.000000]   node   0: [mem 0x0000000000000000-0x000000007feffff]
```

```
[    0.000000]   node   0: [mem 0x0000000800000000-0x000000087fffffff]
[    0.000000] Zeroed struct page in unavailable ranges: 256 pages
[    0.000000] Initmem setup node 0 [mem
0x0000000000000000-0x000000087fffffff]
[    0.000000] psci: probing for conduit method from DT.
[    0.000000] psci: PSCIv1.1 detected in firmware.
[    0.000000] psci: Using standard PSCI v0.2 function IDs
[    0.000000] psci: MIGRATE_INFO_TYPE not supported.
[    0.000000] psci: SMC Calling Convention v1.2
[    0.000000] percpu: Embedded 22 pages/cpu s50968 r8192 d30952 u90112
[    0.000000] Detected VIPT I-cache on CPU0
[    0.000000] CPU features: detected: ARM erratum 845719
[    0.000000] CPU features: detected: ARM erratum 843419
[    0.000000] Built 1 zonelists, mobility grouping on.  Total pages:
1031940
[    0.000000] Kernel command line:  earlycon console=ttyPS0,115200
clk_ignore_unused init_fatal_sh=1
[    0.000000] Dentry cache hash table entries: 524288 (order: 10,
4194304 bytes, linear)
[    0.000000] Inode-cache hash table entries: 262144 (order: 9, 2097152
bytes, linear)
[    0.000000] mem auto-init: stack:off, heap alloc:off, heap free:off
[    0.000000] software IO TLB: mapped [mem
0x0000000069800000-0x000000006d800000] (64MB)
[    0.000000] Memory: 3758028K/4193280K available (13888K kernel code,
982K rwdata, 3920K rodata, 2112K init, 586K bss, 173108K reserved,
262144K cma-reserved)
[    0.000000] rcu: Hierarchical RCU implementation.
[    0.000000] rcu:     RCU event tracing is enabled.
[    0.000000] rcu: RCU calculated value of scheduler-enlistment delay
is 25 jiffies.
[    0.000000] NR_IRQS: 64, nr_irqs: 64, preallocated irqs: 0
[    0.000000] GIC: Adjusting CPU interface base to 0x00000000f902f000
[    0.000000] GIC: Using split EOI/Deactivate mode
[    0.000000] irq-xilinx: mismatch in kind-of-intr param
[    0.000000] irq-xilinx: /amba_pl@0/interrupt-controller@80020000:
num_irq=32, sw_irq=0, edge=0x1
[    0.000000] random: get_random_bytes called from start_kernel+0x31c/
0x524 with crng_init=0
[    0.000000] arch_timer: cp15 timer(s) running at 65.00MHz (phys).
[    0.000000] clocksource: arch_sys_counter: mask: 0xffffffffffffff
max_cycles: 0xefdb196da, max_idle_ns: 440795204367 ns
[    0.000160] sched_clock: 56 bits at 65MHz, resolution 15ns, wraps
every 2199023255550ns
[    0.010444] Console: colour dummy device 80x25
[    0.012051] Calibrating delay loop (skipped), value calculated using
timer frequency.. 130.00 BogoMIPS (lpj=260000)
[    0.012503] pid_max: default: 32768 minimum: 301
[    0.015340] Mount-cache hash table entries: 8192 (order: 4, 65536
bytes, linear)
[    0.015736] Mountpoint-cache hash table entries: 8192 (order: 4,
65536 bytes, linear)
[    0.061816] rcu: Hierarchical SRCU implementation.
[    0.067823] EFI services will not be available.
[    0.072112] smp: Bringing up secondary CPUs ...
[    0.095019] Detected VIPT I-cache on CPU1
[    0.096811] CPU1: Booted secondary processor 0x0000000001 [0x410fd034]
[    0.118827] Detected VIPT I-cache on CPU2
[    0.119379] CPU2: Booted secondary processor 0x0000000002 [0x410fd034]
[    0.150279] Detected VIPT I-cache on CPU3
[    0.150679] CPU3: Booted secondary processor 0x0000000003 [0x410fd034]
[    0.161705] smp: Brought up 1 node, 4 CPUs
[    0.170776] SMP: Total of 4 processors activated.
```

```
[    0.174416] CPU features: detected: 32-bit EL0 Support
[    0.174653] CPU features: detected: CRC32 instructions
[    0.242828] CPU: All CPU(s) started at EL2
[    0.279093] alternatives: patching kernel code
[    0.384714] devtmpfs: initialized
[    0.487081] clocksource: jiffies: mask: 0xffffffff max_cycles:
0xffffffff, max_idle_ns: 7645041785100000 ns
[    0.487700] futex hash table entries: 1024 (order: 4, 65536 bytes,
linear)
[    0.504308] pinctrl core: initialized pinctrl subsystem
[    0.534158] DMI not present or invalid.
[    0.539466] NET: Registered protocol family 16
[    0.580400] DMA: preallocated 512 KiB GFP_KERNEL pool for atomic
allocations
[    0.581088] DMA: preallocated 512 KiB GFP_KERNEL|GFP_DMA32 pool for
atomic allocations
[    0.581767] audit: initializing netlink subsys (disabled)
[    0.598871] audit: type=2000 audit(0.236:1): state=initialized
audit_enabled=0 res=1
[    0.613090] cpuidle: using governor menu
[    0.614407] hw-breakpoint: found 6 breakpoint and 4 watchpoint
registers.
[    0.619956] ASID allocator initialised with 65536 entries
[    0.622040] Serial: AMBA PL011 UART driver
[    0.804498] HugeTLB registered 1.00 GiB page size, pre-allocated 0
pages
[    0.805223] HugeTLB registered 32.0 MiB page size, pre-allocated 0
pages
[    0.807043] HugeTLB registered 2.00 MiB page size, pre-allocated 0
pages
[    0.807935] HugeTLB registered 64.0 KiB page size, pre-allocated 0
pages
[    2.794862] cryptd: max_cpu_qlen set to 1000
[    3.158448] DRBG: Continuing without Jitter RNG
[    3.339101] raid6: neonx8   gen()  2590 MB/s
[    3.471358] raid6: neonx8   xor()   572 MB/s
[    3.665883] raid6: neonx4   gen()  3648 MB/s
[    3.786737] raid6: neonx4   xor()   776 MB/s
[    3.934202] raid6: neonx2   gen()  2682 MB/s
[    4.047159] raid6: neonx2   xor()   902 MB/s
[    4.181538] raid6: neonx1   gen()  1302 MB/s
[    4.349578] raid6: neonx1   xor()   723 MB/s
[    4.466416] raid6: int64x8  gen()  1720 MB/s
[    4.619072] raid6: int64x8  xor()  1054 MB/s
[    4.772621] raid6: int64x4  gen()  1973 MB/s
[    4.919665] raid6: int64x4  xor()   979 MB/s
[    5.131932] raid6: int64x2  gen()  3194 MB/s
[    5.355130] raid6: int64x2  xor()   755 MB/s
[    5.435320] raid6: int64x1  gen()  1038 MB/s
[    5.545000] raid6: int64x1  xor()   841 MB/s
[    5.545332] raid6: using algorithm neonx4 gen() 3648 MB/s
[    5.545958] raid6: .... xor() 776 MB/s, rmw enabled
[    5.546500] raid6: using neon recovery algorithm
[    5.554844] iommu: Default domain type: Translated
[    5.559710] SCSI subsystem initialized
[    5.563501] usbcore: registered new interface driver usbfs
[    5.565578] usbcore: registered new interface driver hub
[    5.568820] usbcore: registered new device driver usb
[    5.570315] mc: Linux media interface: v0.10
[    5.571071] videodev: Linux video capture interface: v2.00
[    5.572198] EDAC MC: Ver: 3.0.0
[    5.586980] zynqmp-ipi-mbox mailbox@ff990400: Registered ZynqMP IPI
mbox with TX/RX channels.
```

```
[     5.590533] FPGA manager framework
[     5.592779] Advanced Linux Sound Architecture Driver Initialized.
[     5.606911] Bluetooth: Core ver 2.22
[     5.608185] NET: Registered protocol family 31
[     5.608515] Bluetooth: HCI device and connection manager initialized
[     5.609123] Bluetooth: HCI socket layer initialized
[     5.610491] Bluetooth: L2CAP socket layer initialized
[     5.611216] Bluetooth: SCO socket layer initialized
[     5.649686] clocksource: Switched to clocksource arch_sys_counter
[     5.652090] VFS: Disk quotas dquot_6.6.0
[     5.652962] VFS: Dquot-cache hash table entries: 512 (order 0, 4096
bytes)
[     6.047047] NET: Registered protocol family 2
[     6.069245] tcp_listen_portaddr_hash hash table entries: 2048 (order:
3, 32768 bytes, linear)
[     6.070309] TCP established hash table entries: 32768 (order: 6,
262144 bytes, linear)
[     6.071335] TCP bind hash table entries: 32768 (order: 7, 524288
bytes, linear)
[     6.072384] TCP: Hash tables configured (established 32768 bind 32768)
[     6.074332] UDP hash table entries: 2048 (order: 4, 65536 bytes,
linear)
[     6.075200] UDP-Lite hash table entries: 2048 (order: 4, 65536 bytes,
linear)
[     6.078690] NET: Registered protocol family 1
[     6.096178] RPC: Registered named UNIX socket transport module.
[     6.096667] RPC: Registered udp transport module.
[     6.096880] RPC: Registered tcp transport module.
[     6.097204] RPC: Registered tcp NFSv4.1 backchannel transport module.
[     6.102558] PCI: CLS 0 bytes, default 64
[     6.120932] Trying to unpack rootfs image as initramfs...
[     6.770036] Freeing initrd memory: 5000K
[     8.162841] Initialise system trusted keyrings
[     8.176753] workingset: timestamp_bits=46 max_order=20 bucket_order=0
[     8.296739] NFS: Registering the id_resolver key type
[     8.297476] Key type id_resolver registered
[     8.298017] Key type id_legacy registered
[     8.298783] nfs4filelayout_init: NFSv4 File Layout Driver
Registering...
[     8.299670] jffs2: version 2.2. (NAND) (SUMMARY)  © 2001-2006 Red
Hat, Inc.
[     8.375929] NET: Registered protocol family 38
[     8.376462] xor: measuring software checksum speed
[     8.383833]    8regs           :  1975 MB/sec
[     8.389349]    32regs          :  2078 MB/sec
[     8.398582]    arm64_neon      :  1835 MB/sec
[     8.398754] xor: using function: 32regs (2078 MB/sec)
[     8.398953] Key type asymmetric registered
[     8.399436] Asymmetric key parser 'x509' registered
[     8.407759] Block layer SCSI generic (bsg) driver version 0.4 loaded
(major 247)
[     8.416454] io scheduler mq-deadline registered
[     8.417151] io scheduler kyber registered
[     8.427905] ps_pcie_dma init()
[     9.587342] Serial: 8250/16550 driver, 4 ports, IRQ sharing disabled
[     9.595888] Serial: AMBA driver
[     9.775681] brd: module loaded
[     9.914787] loop: module loaded
[     9.918566] mtdoops: mtd device (mtddev=name/number) must be supplied
[     9.925498] libphy: Fixed MDIO Bus: probed
[     9.929734] tun: Universal TUN/TAP device driver, 1.6
[     9.936832] CAN device driver interface
[     9.939015] usbcore: registered new interface driver asix
```

```
[    9.951885] usbcore: registered new interface driver ax88179_178a
[    9.952098] usbcore: registered new interface driver cdc_ether
[    9.952314] usbcore: registered new interface driver net1080
[    9.952497] usbcore: registered new interface driver cdc_subset
[    9.953133] usbcore: registered new interface driver zaurus
[    9.953702] usbcore: registered new interface driver cdc_ncm
[    9.974796] usbcore: registered new interface driver uas
[    9.975219] usbcore: registered new interface driver usb-storage
[    9.993558] rtc_zynqmp ffa60000.rtc: registered as rtc0
[    9.996654] rtc_zynqmp ffa60000.rtc: setting system clock to
2021-06-04T00:53:27 UTC (1622768007)
[    9.997417] i2c /dev entries driver
[   10.002452] usbcore: registered new interface driver uvcvideo
[   10.004260] USB Video Class driver (1.1.1)
[   10.037708] Bluetooth: HCI UART driver ver 2.3
[   10.037972] Bluetooth: HCI UART protocol H4 registered
[   10.038246] Bluetooth: HCI UART protocol BCSP registered
[   10.038583] Bluetooth: HCI UART protocol LL registered
[   10.038904] Bluetooth: HCI UART protocol ATH3K registered
[   10.039226] Bluetooth: HCI UART protocol Three-wire (H5) registered
[   10.039672] Bluetooth: HCI UART protocol Intel registered
[   10.039969] Bluetooth: HCI UART protocol QCA registered
[   10.040336] usbcore: registered new interface driver bcm203x
[   10.040788] usbcore: registered new interface driver bpa10x
[   10.041231] usbcore: registered new interface driver bfusb
[   10.064174] usbcore: registered new interface driver btusb
[   10.064717] usbcore: registered new interface driver ath3k
[   10.065837] EDAC MC: ECC not enabled
[   10.066603] EDAC ZynqMP-OCM: ECC not enabled - Disabling EDAC driver
[   10.070617] sdhci: Secure Digital Host Controller Interface driver
[   10.070797] sdhci: Copyright(c) Pierre Ossman
[   10.070922] sdhci-pltfm: SDHCI platform and OF driver helper
[   10.080130] ledtrig-cpu: registered to indicate activity on CPUs
[   10.080577] SMCCC: SOC_ID: ARCH_SOC_ID not implemented, skipping ....
[   10.088194] zynqmp_firmware_probe Platform Management API v1.1
[   10.088495] zynqmp_firmware_probe Trustzone version v1.0
[   10.271775] zynqmp-pinctrl firmware:zynqmp-firmware:pinctrl: zynqmp
pinctrl initialized
[   11.998987] alg: No test for xilinx-zynqmp-aes (zynqmp-aes)
[   12.002037] zynqmp_aes firmware:zynqmp-firmware:zynqmp-aes: AES
Successfully Registered
[   12.004797] alg: No test for xilinx-keccak-384 (zynqmp-keccak-384)
[   12.019407] alg: No test for xilinx-zynqmp-rsa (zynqmp-rsa)
[   12.030505] usbcore: registered new interface driver usbhid
[   12.031614] usbhid: USB HID core driver
[   12.077030] ARM CCI_400_r1 PMU driver probed
[   12.092185] fpga_manager fpga0: Xilinx ZynqMP FPGA Manager registered
[   12.097082] usbcore: registered new interface driver snd-usb-audio
[   12.108012] pktgen: Packet Generator for packet performance testing.
Version: 2.75
[   12.153675] Initializing XFRM netlink socket
[   12.154694] NET: Registered protocol family 10
[   12.195881] Segment Routing with IPv6
[   12.241371] sit: IPv6, IPv4 and MPLS over IPv4 tunneling driver
[   12.258762] NET: Registered protocol family 17
[   12.267624] NET: Registered protocol family 15
[   12.268561] can: controller area network core
[   12.276038] NET: Registered protocol family 29
[   12.277124] can: raw protocol
[   12.277392] can: broadcast manager protocol
[   12.278961] can: netlink gateway - max_hops=1
[   12.280772] Bluetooth: RFCOMM TTY layer initialized
[   12.284294] Bluetooth: RFCOMM socket layer initialized
```

```
[    12.285120] Bluetooth: RFCOMM ver 1.11
[    12.286219] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
[    12.286549] Bluetooth: BNEP filters: protocol multicast
[    12.286814] Bluetooth: BNEP socket layer initialized
[    12.287024] Bluetooth: HIDP (Human Interface Emulation) ver 1.2
[    12.287667] Bluetooth: HIDP socket layer initialized
[    12.289552] 9pnet: Installing 9P2000 support
[    12.290079] Key type dns_resolver registered
[    12.292357] registered taskstats version 1
[    12.292732] Loading compiled-in X.509 certificates
[    12.304748] Btrfs loaded, crc32c=crc32c-generic
[    12.435845] ff000000.serial: ttyPS0 at MMIO 0xff000000 (irq = 56,
base_baud = 2169921) is a xuartps
[    12.438810] printk: console [ttyPS0] enabled
[    12.438810] printk: console [ttyPS0] enabled
[    12.439934] printk: bootconsole [cdns0] disabled
[    12.439934] printk: bootconsole [cdns0] disabled
[    12.535110] ff010000.serial: ttyPS1 at MMIO 0xff010000 (irq = 57,
base_baud = 2169921) is a xuartps
[    12.543093] of-fpga-region fpga-full: FPGA Region probed
[    12.627812] nwl-pcie fd0e0000.pcie: host bridge /axi/pcie@fd0e0000
ranges:
[    12.634296] nwl-pcie fd0e0000.pcie:      MEM
0x00e0000000..0x00efffffff -> 0x00e0000000
[    12.634843] nwl-pcie fd0e0000.pcie:      MEM
0x0600000000..0x07ffffffff -> 0x0600000000
[    12.641979] nwl-pcie fd0e0000.pcie: Link is UP
[    12.652252] nwl-pcie fd0e0000.pcie: PCI host bridge to bus 0000:00
[    12.653284] pci_bus 0000:00: root bus resource [bus 00-ff]
[    12.654291] pci_bus 0000:00: root bus resource [mem
0xe0000000-0xefffffff]
[    12.667394] pci_bus 0000:00: root bus resource [mem
0x600000000-0x7fffffff pref]
[    12.694921] xilinx-zynqmp-dma fd500000.dma: ZynqMP DMA driver Probe
success
[    12.696542] xilinx-zynqmp-dma fd510000.dma: ZynqMP DMA driver Probe
success
[    12.700628] xilinx-zynqmp-dma fd520000.dma: ZynqMP DMA driver Probe
success
[    12.701706] xilinx-zynqmp-dma fd530000.dma: ZynqMP DMA driver Probe
success
[    12.702807] xilinx-zynqmp-dma fd540000.dma: ZynqMP DMA driver Probe
success
[    12.704385] xilinx-zynqmp-dma fd550000.dma: ZynqMP DMA driver Probe
success
[    12.705848] xilinx-zynqmp-dma fd560000.dma: ZynqMP DMA driver Probe
success
[    12.707152] xilinx-zynqmp-dma fd570000.dma: ZynqMP DMA driver Probe
success
[    12.720518] xilinx-zynqmp-dma ffa80000.dma: ZynqMP DMA driver Probe
success
[    12.725104] xilinx-zynqmp-dma ffa90000.dma: ZynqMP DMA driver Probe
success
[    12.726359] xilinx-zynqmp-dma ffaa0000.dma: ZynqMP DMA driver Probe
success
[    12.743792] xilinx-zynqmp-dma ffab0000.dma: ZynqMP DMA driver Probe
success
[    12.744949] xilinx-zynqmp-dma ffac0000.dma: ZynqMP DMA driver Probe
success
[    12.746678] xilinx-zynqmp-dma ffad0000.dma: ZynqMP DMA driver Probe
success
[    12.747845] xilinx-zynqmp-dma ffae0000.dma: ZynqMP DMA driver Probe
success
```

```
[    12.748905] xilinx-zynqmp-dma ffaf0000.dma: ZynqMP DMA driver Probe
success
[    12.771931] xilinx-zynqmp-dpdma fd4c0000.dma-controller: Xilinx DPDMA
engine is probed
[    12.815841] ahci-ceva fd0c0000.ahci: supply ahci not found, using
dummy regulator
[    12.822252] ahci-ceva fd0c0000.ahci: supply phy not found, using
dummy regulator
[    12.874863] spi-nor spi0.0: found n25q512a, expected m25p80
[    12.942624] spi-nor spi0.0: trying to lock already unlocked area
[    12.943006] spi-nor spi0.0: n25q512a (131072 Kbytes)
[    12.944647] 3 fixed-partitions partitions found on MTD device spi0.0
[    12.950733] Creating 3 MTD partitions on "spi0.0":
[    12.952688] 0x000000000000-0x000001e00000 : "boot"
[    12.970234] 0x000001e00000-0x000001e40000 : "bootenv"
[    12.974283] 0x000001e40000-0x000004240000 : "kernel"
[    13.356823] macb ff0e0000.ethernet: Not enabling partial store and
forward
[    13.365850] libphy: MACB_mii_bus: probed
[    13.422289] macb ff0e0000.ethernet eth0: Cadence GEM rev 0x40070106
at 0xff0e0000 irq 39 (d6:ec:61:e0:83:3a)
[    13.430741] xilinx-axipmon ffa00000.perf-monitor: Probed Xilinx APM
[    13.436784] xilinx-axipmon fd0b0000.perf-monitor: Probed Xilinx APM
[    13.438268] xilinx-axipmon fd490000.perf-monitor: Probed Xilinx APM
[    13.450567] xilinx-axipmon ffa10000.perf-monitor: Probed Xilinx APM
[    13.887514] pca953x 0-0020: supply vcc not found, using dummy
regulator
[    13.890027] pca953x 0-0020: using no AI
[    13.909311] pca953x 0-0021: supply vcc not found, using dummy
regulator
[    13.909787] pca953x 0-0021: using no AI
[    14.066539] i2c i2c-0: Added multiplexed i2c bus 2
[    14.119985] i2c i2c-0: Added multiplexed i2c bus 3
[    14.144474] max20751 4-0072: Failed to identify chip capabilities
[    14.203282] max20751 4-0073: Failed to identify chip capabilities
[    14.204141] i2c i2c-0: Added multiplexed i2c bus 4
[    14.205317] i2c i2c-0: Added multiplexed i2c bus 5
[    14.205677] pca954x 0-0075: registered 4 multiplexed busses for I2C
mux pca9544
[    14.206952] cdns-i2c ff020000.i2c: 400 kHz mmio ff020000 irq 41
[    14.248287] at24 6-0054: supply vcc not found, using dummy regulator
[    14.258175] at24 6-0054: 1024 byte 24c08 EEPROM, writable, 1 bytes/
write
[    14.259023] i2c i2c-1: Added multiplexed i2c bus 6
[    14.336247] si5341 7-0036: Chip: 0 Grade: 0 Rev: 0
[    14.336653] si5341 7-0036: Model '0' not supported
[    14.337803] si5341: probe of 7-0036 failed with error -22
[    14.339492] i2c i2c-1: Added multiplexed i2c bus 7
[    14.416737] si570 8-005d: registered, current frequency 300000000 Hz
[    14.420191] i2c i2c-1: Added multiplexed i2c bus 8
[    14.530039] si570 9-005d: registered, current frequency 148500000 Hz
[    14.530895] i2c i2c-1: Added multiplexed i2c bus 9
[    14.533141] si5324 10-0069: si5328 probed
[    14.630968] random: fast init done
[    14.904727] si5324 10-0069: si5328 probe successful
[    14.907852] i2c i2c-1: Added multiplexed i2c bus 10
[    14.908788] i2c i2c-1: Added multiplexed i2c bus 11
[    14.909782] i2c i2c-1: Added multiplexed i2c bus 12
[    14.910725] i2c i2c-1: Added multiplexed i2c bus 13
[    14.911066] pca954x 1-0074: registered 8 multiplexed busses for I2C
switch pca9548
[    14.914222] i2c i2c-1: Added multiplexed i2c bus 14
[    14.915988] i2c i2c-1: Added multiplexed i2c bus 15
```

```
[    14.916708] i2c i2c-1: Added multiplexed i2c bus 16
[    14.919655] i2c i2c-1: Added multiplexed i2c bus 17
[    14.920944] i2c i2c-1: Added multiplexed i2c bus 18
[    14.923181] i2c i2c-1: Added multiplexed i2c bus 19
[    14.926153] i2c i2c-1: Added multiplexed i2c bus 20
[    14.926846] i2c i2c-1: Added multiplexed i2c bus 21
[    14.927060] pca954x 1-0075: registered 8 multiplexed busses for I2C
switch pca9548
[    14.927353] cdns-i2c ff030000.i2c: 400 kHz mmio ff030000 irq 42
[    14.942489] cdns-wdt fd4d0000.watchdog: Xilinx Watchdog Timer with
timeout 60s
[    14.944393] cdns-wdt ff150000.watchdog: Xilinx Watchdog Timer with
timeout 10s
[    14.974879] cpufreq: cpufreq_online: CPU0: Running at unlisted
initial frequency: 277750 KHz, changing to: 299999 KHz
[    15.269747] ahci-ceva fd0c0000.ahci: supply ahci not found, using
dummy regulator
[    15.275128] ahci-ceva fd0c0000.ahci: supply phy not found, using
dummy regulator
[    15.479977] ahci-ceva fd0c0000.ahci: supply ahci not found, using
dummy regulator
[    15.480786] ahci-ceva fd0c0000.ahci: supply phy not found, using
dummy regulator
[    15.639288] input: gpio-keys as /devices/platform/gpio-keys/input/
input0
[    15.642604] of_cfs_init
[    15.643141] of_cfs_init: OK
[    15.643965] cfg80211: Loading compiled-in X.509 certificates for
regulatory database
[    15.920288] ahci-ceva fd0c0000.ahci: supply ahci not found, using
dummy regulator
[    15.934093] mmc0: SDHCI controller on ff170000.mmc [ff170000.mmc]
using ADMA 64-bit
[    15.976580] ahci-ceva fd0c0000.ahci: supply phy not found, using
dummy regulator
[    16.275147] ahci-ceva fd0c0000.ahci: supply ahci not found, using
dummy regulator
[    16.276292] ahci-ceva fd0c0000.ahci: supply phy not found, using
dummy regulator
[    16.293485] hrtimer: interrupt took 16800800 ns
[    16.329842] cfg80211: Loaded X.509 cert 'sforshee: 00b28ddf47aef9cea7'
[    16.330461] clk: Not disabling unused clocks
[    16.499489] ALSA device list:
[    16.499718]   No soundcards found.
[    16.537749] platform regulatory.0: Direct firmware load for
regulatory.db failed with error -2
[    16.551516] cfg80211: failed to load regulatory.db
[    16.571626] Freeing unused kernel memory: 2112K
[    16.643435] mmc0: Problem switching card into high-speed mode!
[    16.647438] Run /init as init process
[    16.778270] mmc0: new SD card at address 4567
[    16.782829] mmcblk0: mmc0:4567 QEMU! 512 MiB
[    16.893747] ahci-ceva fd0c0000.ahci: supply ahci not found, using
dummy regulator
[    16.895429] ahci-ceva fd0c0000.ahci: supply phy not found, using
dummy regulator
/dev/mmcblk0: clean, 5739/86344 files, 196581/345204 blocks
[    19.164310] EXT4-fs (mmcblk0): mounted filesystem with ordered data
mode. Opts: (null)
[    19.165992] ext4 filesystem being mounted at /rootfs supports
timestamps until 2038 (0x7fffffff)
INIT: version 2.97 booting
Starting udev
```

```
[    23.256354] udevd[355]: starting version 3.2.9
[    23.420071] random: udevd: uninitialized urandom read (16 bytes read)
[    23.430067] random: udevd: uninitialized urandom read (16 bytes read)
[    23.431606] random: udevd: uninitialized urandom read (16 bytes read)
[    23.900574] udevd[356]: starting eudev-3.2.9
[    25.056278] urandom_read: 2 callbacks suppressed
[    25.056334] random: udevd: uninitialized urandom read (16 bytes read)
[    25.063557] random: udevd: uninitialized urandom read (16 bytes read)
[    25.063944] random: udevd: uninitialized urandom read (16 bytes read)
[    25.341085] zocl: loading out-of-tree module taints kernel.
[    25.452650] [drm] Probing for xlnx,zocl
[    25.457033] zocl-drm amba_pl@0:zyxclmm_drm: IRQ index 32 not found
[    25.461206] [drm] FPGA programming device pcap founded.
[    25.465205] [drm] PR Isolation addr 0x0
[    25.550428] [drm] Initialized zocl 0.0.0 00000 for
amba_pl@0:zyxclmm_drm on minor 0
[    25.638983] ahci-ceva fd0c0000.ahci: supply ahci not found, using
dummy regulator
[    25.643594] ahci-ceva fd0c0000.ahci: supply phy not found, using
dummy regulator
[    26.922113] mali-utgard: probe of fd4b0000.gpu failed with error -14
[    35.033257] EXT4-fs (mmcblk0): re-mounted. Opts: (null)
[    35.034460] ext4 filesystem being remounted at / supports timestamps
until 2038 (0x7fffffff)
[    41.711269] Unloading old XRT Linux kernel modules
[    41.931159] Loading new XRT Linux kernel modules
[    42.037923] [drm] Probing for xlnx,zocl
[    42.039120] zocl-drm amba_pl@0:zyxclmm_drm: IRQ index 32 not found
[    42.039864] [drm] FPGA programming device pcap founded.
[    42.040063] [drm] PR Isolation addr 0x0
[    42.055149] [drm] Initialized zocl 0.0.0 00000 for
amba_pl@0:zyxclmm_drm on minor 0
[    42.078482] ahci-ceva fd0c0000.ahci: supply ahci not found, using
dummy regulator
[    42.079319] ahci-ceva fd0c0000.ahci: supply phy not found, using
dummy regulator
[    47.672399] INFO: Creating ICD entry for Xilinx Platform
[    49.059330] update-alternatives: Linking /usr/lib/libMali.so.9.0
to /usr/lib/x11/libMali.so.9.0
[    50.341071] update-alternatives: Linking /usr/lib/libMali.so.9.0
to /usr/lib/x11/libMali.so.9.0
[    50.924487] Warn: update-alternatives: libmali-xlnx has multiple
providers with the same priority, please check /usr/lib/opkg/
alternatives/libmali-xlnx for details
[    51.390668] update-alternatives: Linking /usr/lib/libMali.so.9.0
to /usr/lib/x11/libMali.so.9.0
[    52.436957] update-alternatives: Linking /usr/lib/libMali.so.9.0
to /usr/lib/x11/libMali.so.9.0
[    55.111814] macb ff0e0000.ethernet eth0: PHY [ff0e0000.ethernet-
ffffffff:0c] driver [Marvell 88E1118] (irq=POLL)
[    55.112497] macb ff0e0000.ethernet eth0: configuring for phy/rgmii-id
link mode
[    55.235486] macb ff0e0000.ethernet eth0: Link is Up - 1Gbps/Full -
flow control tx
[    55.236207] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[    60.898444] random: crng init done
[    60.898871] random: 2 urandom warning(s) missed due to ratelimiting
Starting tcf-agent: OK

root@xilinx-zcu102-2021_1:~#
```

TIP: *To exit the emulator when you are finished, press **Ctrl + A**, release, and then press **X**.*

Additional Options for Booting on QEMU

- To download the newly built `<plnx-proj-root>/images/linux/u-boot.elf` with QEMU:

```
$ petalinux-boot --qemu --u-boot
```

  - For Zynq® UltraScale+™ MPSoC and Versal ACAP, it loads `<plnx-proj-root>/images/linux/u-boot.elf` and boots the ATF image `<plnx-proj-root>/images/linux/bl31.elf` with QEMU. The ATF then boots the loaded U-Boot image.
  - For MicroBlaze™ CPUs and Zynq-7000 devices, it boots `<plnx-proj-root>/images/linux/u-boot.elf` with QEMU.

- To download the newly built kernel with QEMU:

```
$ petalinux-boot --qemu --kernel
```

  - For MicroBlaze processors, it boots `<plnx-proj-root>/images/linux/image.elf` with QEMU.
  - For Zynq-7000 devices, it boots `<plnx-proj-root>/images/linux/zImage` with QEMU.
  - For Zynq UltraScale+ MPSoC, it loads the kernel image `<plnx-proj-root>/images/linux/Image` and boots the ATF image `<plnx-proj-root>/images/linux/bl31.elf` with QEMU, and the ATF then boots the loaded kernel image, with PMU firmware running in the background.
  - For Versal™ ACAP, it loads the kernel image `<plnx-proj-root>/images/linux/Image` and boots the ATF image `<plnx-proj-root>/images/linux/bl31.elf` with QEMU, and the ATF then boots the loaded kernel image with PLM and PSM firmware running in the background.

Note: For Zynq UltraScale+ MPSoC kernel boot, create a `pre-built/linux/images/` folder and copy `pmu_rom_qemu_sha3.elf` from any Zynq UltraScale+ MPSoC BSP project. You can also pass `pmu_rom_qemu_sha3.elf` using `--pmu-qemu-args`.

```
$ cd <plnx-proj-root>
$ mkdir -p pre-built/linux/images
$ cp <zynq UltraScale+ bsp project directory>/pre-built/linux/images/
pmu_rom_qemu_sha3.elf pre-built/linux/images/
```

or

```
$ petalinux-boot --qemu --u-boot --pmu-qemu-args" -kernel
pmu_rom_qemu_sha3.elf"
```

During startup, the normal Linux boot process ending with a login prompt is displayed as shown below:

```
$ petalinux-boot --qemu --kernel
[INFO] Sourcing buildtools
Image resized.
INFO: No DTB has been specified, use the default one "<plnx-proj-root>/
xilinx-zcu102-2021.1/images/linux/system.dtb".
INFO: No DTB has been specified, use the default one "<plnx-proj-root>/
xilinx-zcu102-2021.1/images/linux/system.dtb".
INFO: Starting microblaze QEMU
INFO: Starting the above QEMU command in the background
INFO:  qemu-system-microblazeel -M microblaze-fdt   -serial mon:stdio -
serial /dev/null -display none -kernel /xilinx-zcu102-2021.1/pre-built/
linux/images/pmu_rom_qemu_sha3.elf -device loader,file=/xilinx-
zcu102-2021.1/images/linux/pmufw.elf      -hw-dtb /xilinx-zcu102-2021.1/
images/linux/zynqmp-qemu-multiarch-pmu.dtb -machine-path /tmp/
tmp.smx5yp8FUo -device loader,addr=0xfd1a0074,data=0x1011003,data-len=4 -
device loader,addr=0xfd1a007C,data=0x1010f03,data-len=4
qemu-system-microblazeel: Failed to connect socket /tmp/tmp.smx5yp8FUo/qemu-
rport-_pmu@0: No such file or directory
qemu-system-microblazeel: info: QEMU waiting for connection on:
disconnected:unix:/tmp/tmp.smx5yp8FUo/qemu-rport-_pmu@0,server
INFO: TCP PORT is free
INFO: Starting aarch64 QEMU
INFO:  qemu-system-aarch64 -M arm-generic-fdt   -serial mon:stdio -
serial /dev/null -display none -device loader,file=/xilinx-zcu102-2021.1/
images/linux/bl31.elf,cpu-num=0 -device loader,file=/xilinx-zcu102-2021.1/
images/linux/ramdisk.cpio.gz.u-boot,addr=0x04000000,force-raw -device
loader,file=/xilinx-zcu102-2021.1/images/linux/u-boot.elf -device
loader,file=/xilinx-zcu102-2021.1/images/linux/Image,addr=0x00200000,force-
raw -device loader,file=/xilinx-zcu102-2021.1/images/linux/
system.dtb,addr=0x00100000,force-raw -device loader,file=/xilinx-
zcu102-2021.1/images/linux/boot.scr,addr=0x20000000,force-raw -gdb
tcp::9003   -net nic -net nic -net nic -net nic,netdev=eth0 -netdev
user,id=eth0,tftp=/tftpboot   -hw-dtb /xilinx-zcu102-2021.1/images/linux/
zynqmp-qemu-multiarch-arm.dtb -machine-path /tmp/tmp.smx5yp8FUo -global
xlnx,zynqmp-boot.cpu-num=0 -global xlnx,zynqmp-boot.use-pmufw=true  -drive
if=sd,format=raw,index=1,file=/xilinx-zcu102-2021.1/images/linux/
rootfs.ext4   -m 4G
QEMU 5.1.0 monitor - type 'help' for more information
(qemu) qemu-system-aarch64: warning: hub 0 is not connected to host network
PMU Firmware 2021.1    Jun  1 2021   15:25:35
PMU_ROM Version: xpbr-v8.1.0-0
NOTICE:  ATF running on XCZUUNKN/QEMU v4/RTL0.0 at 0xfffea000
NOTICE:  BL31: v2.4(release):v1.1-7609-g851523ea2
NOTICE:  BL31: Built : 08:27:07, Apr 28 2021


U-Boot 2021.01 (May 28 2021 - 10:52:42 +0000)

Model: ZynqMP ZCU102 Rev1.0
Board: Xilinx ZynqMP
DRAM:  4 GiB
PMUFW:    v1.1
EL Level:    EL2
Chip ID:    unknown
NAND:  0 MiB
MMC:   mmc@ff170000: 0
In:    serial
Out:   serial
Err:   serial
```

```
Bootmode: JTAG_MODE
Reset reason:
Net:
ZYNQ GEM: ff0e0000, mdio bus ff0e0000, phyaddr 12, interface rgmii-id

Warning: ethernet@ff0e0000 (eth0) using random MAC address -
12:96:40:91:d0:8d
eth0: ethernet@ff0e0000
Hit any key to stop autoboot:  0
JTAG: Trying to boot script at 20000000
## Executing script at 20000000
Trying to load boot images from jtag
## Loading init Ramdisk from Legacy Image at 04000000 ...
   Image Name:   petalinux-initramfs-image-zynqmp
   Created:      2011-04-05  23:00:00 UTC
   Image Type:   AArch64 Linux RAMDisk Image (uncompressed)
   Data Size:    5123993 Bytes = 4.9 MiB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 00100000
   Booting using the fdt blob at 0x100000
   Loading Ramdisk to 7d82d000, end 7dd0ff99 ... OK
   Loading Device Tree to 000000007d81c000, end 000000007d82cd77 ... OK

Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0x0000000000 [0x410fd034]
[    0.000000] Linux version 5.10.0-xilinx-v2020.2 (oe-user@oe-host)
(aarch64-xilinx-linux-gcc (GCC) 10.2.0, GNU ld (GNU Binutils) 2.35.1) #1
SMP Fri May 28 08:10:27 UTC 2021
[    0.000000] Machine model: ZynqMP ZCU102 Rev1.0
[    0.000000] earlycon: cdns0 at MMIO 0x00000000ff000000 (options
'115200n8')
[    0.000000] printk: bootconsole [cdns0] enabled
[    0.000000] efi: UEFI not found.
[    0.000000] cma: Reserved 256 MiB at 0x000000006d800000
[    0.000000] Zone ranges:
[    0.000000]   DMA32    [mem 0x0000000000000000-0x00000000ffffffff]
[    0.000000]   Normal   [mem 0x0000000100000000-0x000000087fffffff]
[    0.000000] Movable zone start for each node
[    0.000000] Early memory node ranges
[    0.000000]   node   0: [mem 0x0000000000000000-0x000000007effffff]
[    0.000000]   node   0: [mem 0x0000000800000000-0x000000087fffffff]
[    0.000000] Zeroed struct page in unavailable ranges: 256 pages
[    0.000000] Initmem setup node 0 [mem
0x0000000000000000-0x000000087fffffff]
[    0.000000] psci: probing for conduit method from DT.
[    0.000000] psci: PSCIv1.1 detected in firmware.
[    0.000000] psci: Using standard PSCI v0.2 function IDs
[    0.000000] psci: MIGRATE_INFO_TYPE not supported.
[    0.000000] psci: SMC Calling Convention v1.2
[    0.000000] percpu: Embedded 22 pages/cpu s50968 r8192 d30952 u90112
[    0.000000] Detected VIPT I-cache on CPU0
[    0.000000] CPU features: detected: ARM erratum 845719
[    0.000000] CPU features: detected: ARM erratum 843419
[    0.000000] Built 1 zonelists, mobility grouping on.  Total pages:
1031940
[    0.000000] Kernel command line:  earlycon console=ttyPS0,115200
clk_ignore_unused init_fatal_sh=1
[    0.000000] Dentry cache hash table entries: 524288 (order: 10, 4194304
bytes, linear)
[    0.000000] Inode-cache hash table entries: 262144 (order: 9, 2097152
```

```
bytes, linear)
[    0.000000] mem auto-init: stack:off, heap alloc:off, heap free:off
[    0.000000] software IO TLB: mapped [mem
0x0000000069800000-0x000000006d800000] (64MB)
[    0.000000] Memory: 3758028K/4193280K available (13888K kernel code,
982K rwdata, 3920K rodata, 2112K init, 586K bss, 173108K reserved, 262144K
cma-reserved)
[    0.000000] rcu: Hierarchical RCU implementation.
[    0.000000] rcu:     RCU event tracing is enabled.
[    0.000000] rcu: RCU calculated value of scheduler-enlistment delay is
25 jiffies.
[    0.000000] NR_IRQS: 64, nr_irqs: 64, preallocated irqs: 0
[    0.000000] GIC: Adjusting CPU interface base to 0x00000000f902f000
[    0.000000] GIC: Using split EOI/Deactivate mode
[    0.000000] irq-xilinx: mismatch in kind-of-intr param
[    0.000000] irq-xilinx: /amba_pl@0/interrupt-controller@80020000:
num_irq=32, sw_irq=0, edge=0x1
[    0.000000] random: get_random_bytes called from start_kernel+0x31c/
0x524 with crng_init=0
[    0.000000] arch_timer: cp15 timer(s) running at 65.00MHz (phys).
[    0.000000] clocksource: arch_sys_counter: mask: 0xffffffffffffff
max_cycles: 0xefdb196da, max_idle_ns: 440795204367 ns
[    0.000181] sched_clock: 56 bits at 65MHz, resolution 15ns, wraps every
2199023255550ns
[    0.009890] Console: colour dummy device 80x25
[    0.011499] Calibrating delay loop (skipped), value calculated using
timer frequency.. 130.00 BogoMIPS (lpj=260000)
[    0.011905] pid_max: default: 32768 minimum: 301
[    0.014908] Mount-cache hash table entries: 8192 (order: 4, 65536 bytes,
linear)
[    0.015232] Mountpoint-cache hash table entries: 8192 (order: 4, 65536
bytes, linear)
[    0.061391] rcu: Hierarchical SRCU implementation.
[    0.066624] EFI services will not be available.
[    0.070650] smp: Bringing up secondary CPUs ...
[    0.092443] Detected VIPT I-cache on CPU1
[    0.094307] CPU1: Booted secondary processor 0x0000000001 [0x410fd034]
[    0.136665] Detected VIPT I-cache on CPU2
[    0.137319] CPU2: Booted secondary processor 0x0000000002 [0x410fd034]
[    0.203651] Detected VIPT I-cache on CPU3
[    0.204614] CPU3: Booted secondary processor 0x0000000003 [0x410fd034]
[    0.206047] smp: Brought up 1 node, 4 CPUs
[    0.212596] SMP: Total of 4 processors activated.
[    0.214108] CPU features: detected: 32-bit EL0 Support
[    0.214955] CPU features: detected: CRC32 instructions
[    0.262828] CPU: All CPU(s) started at EL2
[    0.313779] alternatives: patching kernel code
[    0.372530] devtmpfs: initialized
[    0.411012] clocksource: jiffies: mask: 0xffffffff max_cycles:
0xffffffff, max_idle_ns: 7645041785100000 ns
[    0.412691] futex hash table entries: 1024 (order: 4, 65536 bytes,
linear)
[    0.448030] pinctrl core: initialized pinctrl subsystem
[    0.460629] DMI not present or invalid.
[    0.463588] NET: Registered protocol family 16
[    0.480238] DMA: preallocated 512 KiB GFP_KERNEL pool for atomic
allocations
[    0.481254] DMA: preallocated 512 KiB GFP_KERNEL|GFP_DMA32 pool for
atomic allocations
[    0.482661] audit: initializing netlink subsys (disabled)
[    0.491529] audit: type=2000 audit(0.168:1): state=initialized
audit_enabled=0 res=1
[    0.494429] cpuidle: using governor menu
```

```
[    0.495783] hw-breakpoint: found 6 breakpoint and 4 watchpoint registers.
[    0.506601] ASID allocator initialised with 65536 entries
[    0.508650] Serial: AMBA PL011 UART driver
[    0.644015] HugeTLB registered 1.00 GiB page size, pre-allocated 0 pages
[    0.644604] HugeTLB registered 32.0 MiB page size, pre-allocated 0 pages
[    0.645124] HugeTLB registered 2.00 MiB page size, pre-allocated 0 pages
[    0.646160] HugeTLB registered 64.0 KiB page size, pre-allocated 0 pages
[    2.969884] cryptd: max_cpu_qlen set to 1000
[    3.435765] DRBG: Continuing without Jitter RNG
[    3.601406] raid6: neonx8   gen()  2919 MB/s
[    3.739342] raid6: neonx8   xor()  1100 MB/s
[    3.944022] raid6: neonx4   gen()  4621 MB/s
[    4.107715] raid6: neonx4   xor()  1504 MB/s
[    4.310124] raid6: neonx2   gen()  4412 MB/s
[    4.399904] raid6: neonx2   xor()   743 MB/s
[    4.514868] raid6: neonx1   gen()  1364 MB/s
[    4.753240] raid6: neonx1   xor()   958 MB/s
[    4.958094] raid6: int64x8  gen()  1762 MB/s
[    5.181490] raid6: int64x8  xor()  1563 MB/s
[    5.354368] raid6: int64x4  gen()  3410 MB/s
[    5.518304] raid6: int64x4  xor()  1257 MB/s
[    5.692287] raid6: int64x2  gen()  1732 MB/s
[    5.812285] raid6: int64x2  xor()   380 MB/s
[    5.962757] raid6: int64x1  gen()  1517 MB/s
[    6.082049] raid6: int64x1  xor()   876 MB/s
[    6.082317] raid6: using algorithm neonx4 gen() 4621 MB/s
[    6.082746] raid6: .... xor() 1504 MB/s, rmw enabled
[    6.083246] raid6: using neon recovery algorithm
[    6.093701] iommu: Default domain type: Translated
[    6.096230] SCSI subsystem initialized
[    6.098801] usbcore: registered new interface driver usbfs
[    6.100366] usbcore: registered new interface driver hub
[    6.102231] usbcore: registered new device driver usb
[    6.195715] mc: Linux media interface: v0.10
[    6.196714] videodev: Linux video capture interface: v2.00
[    6.198718] EDAC MC: Ver: 3.0.0
[    6.225158] zynqmp-ipi-mbox mailbox@ff990400: Registered ZynqMP IPI mbox
with TX/RX channels.
[    6.232143] FPGA manager framework
[    6.237036] Advanced Linux Sound Architecture Driver Initialized.
[    6.293287] Bluetooth: Core ver 2.22
[    6.295619] NET: Registered protocol family 31
[    6.296471] Bluetooth: HCI device and connection manager initialized
[    6.309264] Bluetooth: HCI socket layer initialized
[    6.310578] Bluetooth: L2CAP socket layer initialized
[    6.311865] Bluetooth: SCO socket layer initialized
[    6.363596] clocksource: Switched to clocksource arch_sys_counter
[    6.365509] VFS: Disk quotas dquot_6.6.0
[    6.370976] VFS: Dquot-cache hash table entries: 512 (order 0, 4096
bytes)
[    6.393634] NET: Registered protocol family 2
[    6.401474] tcp_listen_portaddr_hash hash table entries: 2048 (order: 3,
32768 bytes, linear)
[    6.406160] TCP established hash table entries: 32768 (order: 6, 262144
bytes, linear)
[    6.407132] TCP bind hash table entries: 32768 (order: 7, 524288 bytes,
linear)
[    6.415077] TCP: Hash tables configured (established 32768 bind 32768)
[    6.428242] UDP hash table entries: 2048 (order: 4, 65536 bytes, linear)
[    6.428823] UDP-Lite hash table entries: 2048 (order: 4, 65536 bytes,
linear)
[    6.431680] NET: Registered protocol family 1
[    6.460938] RPC: Registered named UNIX socket transport module.
```

```
[    6.461277] RPC: Registered udp transport module.
[    6.461475] RPC: Registered tcp transport module.
[    6.461684] RPC: Registered tcp NFSv4.1 backchannel transport module.
[    6.497860] PCI: CLS 0 bytes, default 64
[    6.507696] Trying to unpack rootfs image as initramfs...
[    7.241585] Freeing initrd memory: 5000K
[    8.729493] Initialise system trusted keyrings
[    8.766031] workingset: timestamp_bits=46 max_order=20 bucket_order=0
[    8.819683] NFS: Registering the id_resolver key type
[    8.820687] Key type id_resolver registered
[    8.820932] Key type id_legacy registered
[    8.822685] nfs4filelayout_init: NFSv4 File Layout Driver Registering...
[    8.831550] jffs2: version 2.2. (NAND) (SUMMARY)  © 2001-2006 Red Hat,
Inc.
[    8.888697] NET: Registered protocol family 38
[    8.889275] xor: measuring software checksum speed
[    8.894161]    8regs           :  2195 MB/sec
[    8.899672]    32regs          :  2050 MB/sec
[    8.906388]    arm64_neon      :  1867 MB/sec
[    8.906565] xor: using function: 8regs (2195 MB/sec)
[    8.906752] Key type asymmetric registered
[    8.906976] Asymmetric key parser 'x509' registered
[    8.908150] Block layer SCSI generic (bsg) driver version 0.4 loaded
(major 247)
[    8.909029] io scheduler mq-deadline registered
[    8.909409] io scheduler kyber registered
[    8.921301] ps_pcie_dma init()
[    9.623214] Serial: 8250/16550 driver, 4 ports, IRQ sharing disabled
[    9.644919] Serial: AMBA driver
[    9.812471] brd: module loaded
[    9.845615] loop: module loaded
[    9.851865] mtdoops: mtd device (mtddev=name/number) must be supplied
[    9.858806] libphy: Fixed MDIO Bus: probed
[    9.873102] tun: Universal TUN/TAP device driver, 1.6
[    9.891666] CAN device driver interface
[    9.894521] usbcore: registered new interface driver asix
[    9.895503] usbcore: registered new interface driver ax88179_178a
[    9.896427] usbcore: registered new interface driver cdc_ether
[    9.897177] usbcore: registered new interface driver net1080
[    9.897807] usbcore: registered new interface driver cdc_subset
[    9.901073] usbcore: registered new interface driver zaurus
[    9.909227] usbcore: registered new interface driver cdc_ncm
[    9.916106] usbcore: registered new interface driver uas
[    9.916811] usbcore: registered new interface driver usb-storage
[    9.940888] rtc_zynqmp ffa60000.rtc: registered as rtc0
[    9.960974] rtc_zynqmp ffa60000.rtc: setting system clock to
2021-06-04T00:57:57 UTC (1622768277)
[    9.965354] i2c /dev entries driver
[    9.976467] usbcore: registered new interface driver uvcvideo
[    9.981777] USB Video Class driver (1.1.1)
[    9.990117] Bluetooth: HCI UART driver ver 2.3
[    9.995640] Bluetooth: HCI UART protocol H4 registered
[    9.995949] Bluetooth: HCI UART protocol BCSP registered
[    9.999144] Bluetooth: HCI UART protocol LL registered
[    9.999367] Bluetooth: HCI UART protocol ATH3K registered
[   10.001284] Bluetooth: HCI UART protocol Three-wire (H5) registered
[   10.001705] Bluetooth: HCI UART protocol Intel registered
[   10.003139] Bluetooth: HCI UART protocol QCA registered
[   10.003621] usbcore: registered new interface driver bcm203x
[   10.004081] usbcore: registered new interface driver bpa10x
[   10.007436] usbcore: registered new interface driver bfusb
[   10.009083] usbcore: registered new interface driver btusb
[   10.009673] usbcore: registered new interface driver ath3k
```

```
[    10.010813] EDAC MC: ECC not enabled
[    10.011731] EDAC ZynqMP-OCM: ECC not enabled - Disabling EDAC driver
[    10.020168] sdhci: Secure Digital Host Controller Interface driver
[    10.020502] sdhci: Copyright(c) Pierre Ossman
[    10.020647] sdhci-pltfm: SDHCI platform and OF driver helper
[    10.035351] ledtrig-cpu: registered to indicate activity on CPUs
[    10.036024] SMCCC: SOC_ID: ARCH_SOC_ID not implemented, skipping ....
[    10.046397] zynqmp_firmware_probe Platform Management API v1.1
[    10.052419] zynqmp_firmware_probe Trustzone version v1.0
[    10.153709] zynqmp-pinctrl firmware:zynqmp-firmware:pinctrl: zynqmp
pinctrl initialized
[    11.969974] alg: No test for xilinx-zynqmp-aes (zynqmp-aes)
[    11.972005] zynqmp_aes firmware:zynqmp-firmware:zynqmp-aes: AES
Successfully Registered
[    11.987790] alg: No test for xilinx-keccak-384 (zynqmp-keccak-384)
[    11.990656] alg: No test for xilinx-zynqmp-rsa (zynqmp-rsa)
[    11.993015] usbcore: registered new interface driver usbhid
[    11.993286] usbhid: USB HID core driver
[    12.086498] ARM CCI_400_r1 PMU driver probed
[    12.093586] fpga_manager fpga0: Xilinx ZynqMP FPGA Manager registered
[    12.101308] usbcore: registered new interface driver snd-usb-audio
[    12.113997] pktgen: Packet Generator for packet performance testing.
Version: 2.75
[    12.133751] Initializing XFRM netlink socket
[    12.135270] NET: Registered protocol family 10
[    12.162424] Segment Routing with IPv6
[    12.185071] sit: IPv6, IPv4 and MPLS over IPv4 tunneling driver
[    12.189433] NET: Registered protocol family 17
[    12.190102] NET: Registered protocol family 15
[    12.191289] can: controller area network core
[    12.213060] NET: Registered protocol family 29
[    12.213488] can: raw protocol
[    12.213892] can: broadcast manager protocol
[    12.214390] can: netlink gateway - max_hops=1
[    12.216406] Bluetooth: RFCOMM TTY layer initialized
[    12.217324] Bluetooth: RFCOMM socket layer initialized
[    12.217676] Bluetooth: RFCOMM ver 1.11
[    12.217936] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
[    12.218135] Bluetooth: BNEP filters: protocol multicast
[    12.218670] Bluetooth: BNEP socket layer initialized
[    12.219498] Bluetooth: HIDP (Human Interface Emulation) ver 1.2
[    12.220176] Bluetooth: HIDP socket layer initialized
[    12.222552] 9pnet: Installing 9P2000 support
[    12.223559] Key type dns_resolver registered
[    12.225965] registered taskstats version 1
[    12.226371] Loading compiled-in X.509 certificates
[    12.240521] Btrfs loaded, crc32c=crc32c-generic
[    12.459671] ff000000.serial: ttyPS0 at MMIO 0xff000000 (irq = 56,
base_baud = 2169921) is a xuartps
[    12.462616] printk: console [ttyPS0] enabled
[    12.462616] printk: console [ttyPS0] enabled
[    12.463668] printk: bootconsole [cdns0] disabled
[    12.463668] printk: bootconsole [cdns0] disabled
[    12.522126] ff010000.serial: ttyPS1 at MMIO 0xff010000 (irq = 57,
base_baud = 2169921) is a xuartps
[    12.534452] of-fpga-region fpga-full: FPGA Region probed
[    12.726227] nwl-pcie fd0e0000.pcie: host bridge /axi/pcie@fd0e0000
ranges:
[    12.738830] nwl-pcie fd0e0000.pcie:      MEM 0x00e0000000..0x00efffffff -
> 0x00e0000000
[    12.739342] nwl-pcie fd0e0000.pcie:      MEM 0x0600000000..0x07ffffffff -
> 0x0600000000
[    12.766164] nwl-pcie fd0e0000.pcie: Link is UP
```

```
[   12.794353] nwl-pcie fd0e0000.pcie: PCI host bridge to bus 0000:00
[   12.799593] pci_bus 0000:00: root bus resource [bus 00-ff]
[   12.799887] pci_bus 0000:00: root bus resource [mem
0xe0000000-0xefffffff]
[   12.800132] pci_bus 0000:00: root bus resource [mem
0x600000000-0x7fffffff pref]
[   12.818891] xilinx-zynqmp-dma fd500000.dma: ZynqMP DMA driver Probe
success
[   12.829017] xilinx-zynqmp-dma fd510000.dma: ZynqMP DMA driver Probe
success
[   12.834315] xilinx-zynqmp-dma fd520000.dma: ZynqMP DMA driver Probe
success
[   12.837065] xilinx-zynqmp-dma fd530000.dma: ZynqMP DMA driver Probe
success
[   12.839491] xilinx-zynqmp-dma fd540000.dma: ZynqMP DMA driver Probe
success
[   12.840849] xilinx-zynqmp-dma fd550000.dma: ZynqMP DMA driver Probe
success
[   12.842422] xilinx-zynqmp-dma fd560000.dma: ZynqMP DMA driver Probe
success
[   12.844403] xilinx-zynqmp-dma fd570000.dma: ZynqMP DMA driver Probe
success
[   12.852918] xilinx-zynqmp-dma ffa80000.dma: ZynqMP DMA driver Probe
success
[   12.854963] xilinx-zynqmp-dma ffa90000.dma: ZynqMP DMA driver Probe
success
[   12.856854] xilinx-zynqmp-dma ffaa0000.dma: ZynqMP DMA driver Probe
success
[   12.858478] xilinx-zynqmp-dma ffab0000.dma: ZynqMP DMA driver Probe
success
[   12.860298] xilinx-zynqmp-dma ffac0000.dma: ZynqMP DMA driver Probe
success
[   12.861509] xilinx-zynqmp-dma ffad0000.dma: ZynqMP DMA driver Probe
success
[   12.862615] xilinx-zynqmp-dma ffae0000.dma: ZynqMP DMA driver Probe
success
[   12.863732] xilinx-zynqmp-dma ffaf0000.dma: ZynqMP DMA driver Probe
success
[   12.878455] xilinx-zynqmp-dpdma fd4c0000.dma-controller: Xilinx DPDMA
engine is probed
[   12.887242] ahci-ceva fd0c0000.ahci: supply ahci not found, using dummy
regulator
[   12.891337] ahci-ceva fd0c0000.ahci: supply phy not found, using dummy
regulator
[   12.928067] spi-nor spi0.0: found n25q512a, expected m25p80
[   13.059569] spi-nor spi0.0: trying to lock already unlocked area
[   13.059918] spi-nor spi0.0: n25q512a (131072 Kbytes)
[   13.061384] 3 fixed-partitions partitions found on MTD device spi0.0
[   13.061652] Creating 3 MTD partitions on "spi0.0":
[   13.062065] 0x000000000000-0x000001e00000 : "boot"
[   13.073209] 0x000001e00000-0x000001e40000 : "bootenv"
[   13.089310] 0x000001e40000-0x000004240000 : "kernel"
[   13.409680] macb ff0e0000.ethernet: Not enabling partial store and
forward
[   13.420670] libphy: MACB_mii_bus: probed
[   13.444119] macb ff0e0000.ethernet eth0: Cadence GEM rev 0x40070106 at
0xff0e0000 irq 39 (12:96:40:91:d0:8d)
[   13.448152] xilinx-axipmon ffa00000.perf-monitor: Probed Xilinx APM
[   13.463888] xilinx-axipmon fd0b0000.perf-monitor: Probed Xilinx APM
[   13.470143] xilinx-axipmon fd490000.perf-monitor: Probed Xilinx APM
[   13.473945] xilinx-axipmon ffa10000.perf-monitor: Probed Xilinx APM
[   13.918964] pca953x 0-0020: supply vcc not found, using dummy regulator
[   13.920348] pca953x 0-0020: using no AI
```

```
[   13.947533] pca953x 0-0021: supply vcc not found, using dummy regulator
[   13.948072] pca953x 0-0021: using no AI
[   14.075065] i2c i2c-0: Added multiplexed i2c bus 2
[   14.143051] i2c i2c-0: Added multiplexed i2c bus 3
[   14.184002] max20751 4-0072: Failed to identify chip capabilities
[   14.219650] max20751 4-0073: Failed to identify chip capabilities
[   14.220304] i2c i2c-0: Added multiplexed i2c bus 4
[   14.221709] i2c i2c-0: Added multiplexed i2c bus 5
[   14.222124] pca954x 0-0075: registered 4 multiplexed busses for I2C mux
pca9544
[   14.222669] cdns-i2c ff020000.i2c: 400 kHz mmio ff020000 irq 41
[   14.308818] at24 6-0054: supply vcc not found, using dummy regulator
[   14.322038] at24 6-0054: 1024 byte 24c08 EEPROM, writable, 1 bytes/write
[   14.332234] i2c i2c-1: Added multiplexed i2c bus 6
[   14.381660] si5341 7-0036: Chip: 0 Grade: 0 Rev: 0
[   14.391200] si5341 7-0036: Model '0' not supported
[   14.392263] si5341: probe of 7-0036 failed with error -22
[   14.392700] i2c i2c-1: Added multiplexed i2c bus 7
[   14.419521] si570 8-005d: registered, current frequency 300000000 Hz
[   14.420420] i2c i2c-1: Added multiplexed i2c bus 8
[   14.628914] si570 9-005d: registered, current frequency 148500000 Hz
[   14.634435] i2c i2c-1: Added multiplexed i2c bus 9
[   14.637686] si5324 10-0069: si5328 probed
[   14.719590] random: fast init done
[   14.915923] si5324 10-0069: si5328 probe successful
[   14.917020] i2c i2c-1: Added multiplexed i2c bus 10
[   14.918674] i2c i2c-1: Added multiplexed i2c bus 11
[   14.920344] i2c i2c-1: Added multiplexed i2c bus 12
[   14.923241] i2c i2c-1: Added multiplexed i2c bus 13
[   14.924941] pca954x 1-0074: registered 8 multiplexed busses for I2C
switch pca9548
[   14.928297] i2c i2c-1: Added multiplexed i2c bus 14
[   14.942005] i2c i2c-1: Added multiplexed i2c bus 15
[   14.943714] i2c i2c-1: Added multiplexed i2c bus 16
[   14.945175] i2c i2c-1: Added multiplexed i2c bus 17
[   14.948532] i2c i2c-1: Added multiplexed i2c bus 18
[   14.949847] i2c i2c-1: Added multiplexed i2c bus 19
[   14.951807] i2c i2c-1: Added multiplexed i2c bus 20
[   14.953092] i2c i2c-1: Added multiplexed i2c bus 21
[   14.961224] pca954x 1-0075: registered 8 multiplexed busses for I2C
switch pca9548
[   14.961562] cdns-i2c ff030000.i2c: 400 kHz mmio ff030000 irq 42
[   14.977652] cdns-wdt fd4d0000.watchdog: Xilinx Watchdog Timer with
timeout 60s
[   14.980354] cdns-wdt ff150000.watchdog: Xilinx Watchdog Timer with
timeout 10s
[   14.991651] cpufreq: cpufreq_online: CPU0: Running at unlisted initial
frequency: 277750 KHz, changing to: 299999 KHz
[   15.010354] ahci-ceva fd0c0000.ahci: supply ahci not found, using dummy
regulator
[   15.012736] ahci-ceva fd0c0000.ahci: supply phy not found, using dummy
regulator
[   15.023312] ahci-ceva fd0c0000.ahci: supply ahci not found, using dummy
regulator
[   15.024508] ahci-ceva fd0c0000.ahci: supply phy not found, using dummy
regulator
[   15.043274] input: gpio-keys as /devices/platform/gpio-keys/input/input0
[   15.065525] of_cfs_init
[   15.066383] of_cfs_init: OK
[   15.082471] cfg80211: Loading compiled-in X.509 certificates for
regulatory database
[   15.101616] ahci-ceva fd0c0000.ahci: supply ahci not found, using dummy
regulator
```

```
[   15.105163] ahci-ceva fd0c0000.ahci: supply phy not found, using dummy
regulator
[   15.431939] cfg80211: Loaded X.509 cert 'sforshee: 00b28ddf47aef9cea7'
[   15.437922] clk: Not disabling unused clocks
[   15.701089] ALSA device list:
[   15.708300]   No soundcards found.
[   15.727406] platform regulatory.0: Direct firmware load for
regulatory.db failed with error -2
[   15.728009] cfg80211: failed to load regulatory.db
[   15.900565] mmc0: SDHCI controller on ff170000.mmc [ff170000.mmc] using
ADMA 64-bit
[   15.939464] Freeing unused kernel memory: 2112K
[   15.993732] Run /init as init process
[   16.049773] ahci-ceva fd0c0000.ahci: supply ahci not found, using dummy
regulator
[   16.050531] ahci-ceva fd0c0000.ahci: supply phy not found, using dummy
regulator
[   16.248338] mmc0: Problem switching card into high-speed mode!
[   16.419420] mmc0: new SD card at address 4567
[   16.426585] mmcblk0: mmc0:4567 QEMU! 128 MiB
[   16.499880] ahci-ceva fd0c0000.ahci: supply ahci not found, using dummy
regulator
[   16.500511] ahci-ceva fd0c0000.ahci: supply phy not found, using dummy
regulator
/dev/mmcblk0: recovering journal
/dev/mmcblk0: clean, 1614/23616 files, 77238/94188 blocks
[   18.880085] EXT4-fs (mmcblk0): mounted filesystem with ordered data
mode. Opts: (null)
[   18.880870] ext4 filesystem being mounted at /rootfs supports timestamps
until 2038 (0x7fffffff)
INIT: version 2.97 booting
Starting udev
[   22.533844] udevd[353]: starting version 3.2.9
[   22.663585] random: udevd: uninitialized urandom read (16 bytes read)
[   22.683407] random: udevd: uninitialized urandom read (16 bytes read)
[   22.687224] random: udevd: uninitialized urandom read (16 bytes read)
[   23.218318] udevd[354]: starting eudev-3.2.9
[   23.869930] urandom_read: 5 callbacks suppressed
[   23.869971] random: udevd: uninitialized urandom read (16 bytes read)
[   23.942711] zocl: loading out-of-tree module taints kernel.
[   24.054921] [drm] Probing for xlnx,zocl
[   24.057874] zocl-drm amba_pl@0:zyxclmm_drm: IRQ index 32 not found
[   24.060622] [drm] FPGA programming device pcap founded.
[   24.060880] [drm] PR Isolation addr 0x0
[   24.109576] [drm] Initialized zocl 0.0.0 00000 for amba_pl@0:zyxclmm_drm
on minor 0
[   24.144595] ahci-ceva fd0c0000.ahci: supply ahci not found, using dummy
regulator
[   24.145923] ahci-ceva fd0c0000.ahci: supply phy not found, using dummy
regulator
[   24.320519] random: udevd: uninitialized urandom read (16 bytes read)
[   26.163910] hrtimer: interrupt took 14407231 ns
[   33.600065] EXT4-fs (mmcblk0): re-mounted. Opts: (null)
[   33.602141] ext4 filesystem being remounted at / supports timestamps
until 2038 (0x7fffffff)
[   40.450208] Unloading old XRT Linux kernel modules
[   40.638124] Loading new XRT Linux kernel modules
[   40.862210] [drm] Probing for xlnx,zocl
[   40.867504] zocl-drm amba_pl@0:zyxclmm_drm: IRQ index 32 not found
[   40.868946] [drm] FPGA programming device pcap founded.
[   40.869138] [drm] PR Isolation addr 0x0
[   40.911848] [drm] Initialized zocl 0.0.0 00000 for amba_pl@0:zyxclmm_drm
on minor 0
```

```
[   41.064209] ahci-ceva fd0c0000.ahci: supply ahci not found, using dummy
regulator
[   41.066190] ahci-ceva fd0c0000.ahci: supply phy not found, using dummy
regulator
[   43.888491] INFO: Creating ICD entry for Xilinx Platform
[   46.531832] macb ff0e0000.ethernet eth0: PHY [ff0e0000.ethernet-
ffffffff:0c] driver [Marvell 88E1118] (irq=POLL)
[   46.533367] macb ff0e0000.ethernet eth0: configuring for phy/rgmii-id
link mode
[   46.630552] macb ff0e0000.ethernet eth0: Link is Up - 1Gbps/Full - flow
control tx
[   46.631483] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[   52.181990] random: crng init done
Starting tcf-agent: OK

root@xilinx-zcu102-2021_1:~#
```

You may see slightly different output from the above example depending on the Linux image you test and its configuration.

Try Linux commands such as `ls`, `ifconfig`, `cat/proc/cpuinfo` and so on. They behave the same as on real hardware. To exit the emulator when you are finished, press **Ctrl + A**, release, and then press **X**

- To download a customized U-Boot image with `--uboot/--u-boot` option:

  ```
  $ petalinux-boot --qemu --u-boot/--uboot <specify custom u-boot.elf path>
  ```

- To download a customized kernel image with --kernel option:
  - For Zynq UltraScale+ MPSoC and Versal ACAP, use `image`:

    ```
    petalinux-boot --qemu --kernel <specify custom Image path>
    ```

  - For Zynq-7000 devices, use `zImage`:

    ```
    $ petalinux-boot --qemu --kernel <specify custom zimage path>
    ```

  - For MicroBlaze processors, use `Image.elf`:

    ```
    $ petalinux-boot --qemu --kernel <specify custom Image.elf path>
    ```

- To download a customized dtb image with `--kernel` option:

  ```
  $ petalinux-boot --qemu --kernel <specify custom kernel path> --dtb
  <specify custom dtb path>
  ```

- To download a customized dtb image with `--uboot/--u-boot` option:

  ```
  $ petalinux-boot --qemu --u-boot/--uboot <specify custom u-boot path> --
  dtb <specify custom dtb path>
  ```

  This is not supported by Versal ACAPs.

- To download a customized pmufw image with the `--kernel` option:

```
$ petalinux-boot --qemu --kernel <specify custom kernel path> --pmufw
<specify custom pmufw.elf path>
```

- To download a customized pmufw image with `--uboot/--u-boot` option:

```
$ petalinux-boot --qemu --uboot/--u-boot <specify custom u-boot path> --
pmufw <specify custom pmufw.elf path>
```

***Note*:** QEMU version has been upgraded to 5.1. The old options are deprecated in the new version but remain functionally operational. Warning messages are displayed because PetaLinux tools still use the old options. You can ignore them.

Boot a PetaLinux Image on Hardware with an SD Card

This section describes how to boot a PetaLinux image on hardware with an SD Card.

IMPORTANT! *This section is only for devices that allow booting from SD cards. This includes Versal™ ACAPs, Zynq® UltraScale+™ MPSoCs, and Zynq-7000 devices.*

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have installed PetaLinux Tools on the Linux workstation. If you have not installed, see the Installation Steps.
- You have installed PetaLinux BSP on the Linux workstation. If you have not installed, see the Project Creation Using PetaLinux BSP.
- A serial communication program such as minicom/kermit/gtkterm has been installed; the baud rate of the serial communication program has been set to 115200 bps.

Steps to Boot a PetaLinux Image on Hardware with SD Card

You can boot a PetaLinux image on hardware using an SD card or pre-built wic image after mounting the SD card.

Steps to Flash and Boot the PetaLinux Images Manually

1. Copy the following files from `<plnx-proj-root>/pre-built/linux/images/` into the root directory of the first partition, which is in FAT32 format in the SD card:

- `BOOT.BIN`
- `image.ub`
- `boot.scr`

2. Extract the `rootfs.tar.gz` folder into the ext4 partition of the SD card.
3. Connect the serial port on the board to your workstation.
4. Open a console on the workstation and start the preferred serial communication program (For example: kermit, minicom, gtkterm) with the baud rate set to 115200 on that console.
5. Power off the board.
6. Set the boot mode of the board to SD boot. Refer to the board documentation for details.
7. Plug the SD card into the board.
8. Power on the board.
9. A boot message displays on the serial console.

Steps to Flash and Boot PetaLinux Image Using Prebuilt WIC Image

1. Go to the `<plnx-proj-root>/pre-built/linux/images/` directory. For example, `cd prebuilt/linux/images`.
2. Extract the `petalinux-sdimage.wic.xz` file with the `xz` command. For example, `xz -d petalinux-sdimage.wic.xz`

   ***Note*:** If xz package is not installed in your build machine, use the prebuilt xz binary from `$PETALINUX/components/yocto/buildtools/sysroots/x86_64-petalinux-linux/usr/bin/xz`
3. Flash the extracted `petalinux-sdimage.wic` image into the SD card.
   - **Flash the wic image in Linux:** To flash the wic image to SD card in Linux machines, connect the SD card to the host system and use the `dd` command:

     ```
     dd if=petalinux-sdimage.wic of=/dev/sd<X> conv=fsync
     ```

     ***Note*:** You need sudo access to do this.
   - **Flash the wic image in Windows:** To flash the wic image to the SD card in Windows, you can use any of the following:
     - BalenaEtcher tool
     - Win32DiskImager
4. Insert the SD card into the board and boot the system.
5. Connect the serial port on the board to your workstation.
6. Open a console on the workstation and start the preferred serial communication program (For example: kermit, minicom, gtkterm) with the baud rate set to 115200 on that console.
7. Power off the board.

8. Set the boot mode of the board to SD boot. Refer to the board documentation for details.
9. Plug the SD card into the board.
10. Power on the board. A boot message similar to the following displays on the serial console:

TIP: *To stop auto-boot, hit any key when you see a similar message on the console:* `Hit any key to stop autoboot`.

```
[430.184]Xilinx Versal Platform Loader and Manager
[434.702]Release 2021.1   May 31 2021  -  18:37:19
[439.133]Platform Version: v2.0 PMC: v2.0, PS: v2.0
[443.650]BOOTMODE: 0xE, MULTIBOOT: 0xF0000000
[447.656]****************************************
[452.043] 50.978 ms for Partition#: 0x1, Size: 2288 Bytes
[457.030]---Loading Partition#: 0x2, Id: 0x0
[463.646] 2.662 ms for Partition#: 0x2, Size: 48 Bytes
[465.726]---Loading Partition#: 0x3, Id: 0x0
[476.923] 7.243 ms for Partition#: 0x3, Size: 59280 Bytes
[479.168]---Loading Partition#: 0x4, Id: 0x0
[483.785] 0.663 ms for Partition#: 0x4, Size: 2640 Bytes
[488.032]---Loading Partition#: 0x5, Id: 0x0
[494.890] 2.905 ms for Partition#: 0x5, Size: 3440 Bytes
[497.050]---Loading Partition#: 0x6, Id: 0x0
[501.020] 0.018 ms for Partition#: 0x6, Size: 80 Bytes
PSM Firmware version: 2021.1 [Build: May 31 2021 18:37:19 ]
[511.042]+++Loading Image#: 0x2, Name: pl_cfi, Id: 0x18700000
[516.395]---Loading Partition#: 0x7, Id: 0x3
[719.051] 198.699 ms for Partition#: 0x7, Size: 1304480 Bytes
[721.640]---Loading Partition#: 0x8, Id: 0x5
[876.102] 150.506 ms for Partition#: 0x8, Size: 933344 Bytes
[878.647]+++Loading Image#: 0x3, Name: aie_subsys, Id: 0x0421C005
[884.320]---Loading Partition#: 0x9, Id: 0x7
[893.552] 5.276 ms for Partition#: 0x9, Size: 352 Bytes
[895.671]+++Loading Image#: 0x4, Name: fpd, Id: 0x0420C003
[900.745]---Loading Partition#: 0xA, Id: 0x8
[907.520] 2.819 ms for Partition#: 0xA, Size: 1040 Bytes
[909.944]+++Loading Image#: 0x5, Name: apu_subsystem, Id: 0x1C000000
[915.656]---Loading Partition#: 0xB, Id: 0x0
[925.484] 5.870 ms for Partition#: 0xB, Size: 29520 Bytes
[927.734]---Loading Partition#: 0xC, Id: 0x0
[938.507] 6.813 ms for Partition#: 0xC, Size: 59344 Bytes
[940.756]---Loading Partition#: 0xD, Id: 0x0
[1057.431] 112.718 ms for Partition#: 0xD, Size: 1001184 Bytes
***T**:1 7]****n**** *Booi PDI Loada Donl****
I[1:67.443]Totnl PLu Booo Timt 0
000000
NOTICE:  BL31: v2.4(debug):v1.1-7609-g851523ea2
NOTICE:  BL31: Built : 08:27:07, Apr 28 2021
INFO:    GICv3 with legacy support detected.
INFO:    ARM GICv3 driver initialized in EL3
INFO:    BL31: Initializing runtime services
WARNING: BL31: cortex_a72: CPU workaround for 859971 was missing!
WARNING: BL31: cortex_a72: CPU workaround for 1319367 was missing!
INFO:    BL31: cortex_a72: CPU workaround for cve_2017_5715 was applied
INFO:    BL31: cortex_a72: CPU workaround for cve_2018_3639 was applied
INFO:    BL31: Preparing for EL3 exit to normal world
INFO:    Entry point address = 0x8000000
INFO:    SPSR = 0x3c9
```

```
U-Boot 2021.01 (May 28 2021 - 10:52:42 +0000)

Model: Xilinx Versal vck190 Eval board revA (QSPI)
DRAM:  16 GiB
EL Level:    EL2
MMC:   sdhci@f1050000: 0
Loading Environment from FAT... *** Warning - bad CRC, using default
environment

In:    serial@ff000000
Out:   serial@ff000000
Err:   serial@ff000000
Bootmode: LVL_SHFT_SD_MODE1
Net:
ZYNQ GEM: ff0c0000, mdio bus ff0c0000, phyaddr 1, interface rgmii-id

Warning: ethernet@ff0c0000 (eth0) using random MAC address -
1e:e1:11:40:ef:d6
eth0: ethernet@ff0c0000
ZYNQ GEM: ff0d0000, mdio bus ff0c0000, phyaddr 2, interface rgmii-id

Warning: ethernet@ff0d0000 (eth1) using random MAC address -
f2:85:0f:5f:a3:5f
, eth1: ethernet@ff0d0000
Hit any key to stop autoboot:  0
switch to partitions #0, OK
mmc0 is current device
Scanning mmc 0:1...
Found U-Boot script /boot.scr
2594 bytes read in 13 ms (194.3 KiB/s)
## Executing script at 20000000
Trying to load boot images from mmc0
14374372 bytes read in 1058 ms (13 MiB/s)
## Loading kernel from FIT Image at 10000000 ...
   Using 'conf-system-top.dtb' configuration
   Trying 'kernel-1' kernel subimage
     Description:  Linux kernel
     Type:         Kernel Image
     Compression:  gzip compressed
     Data Start:   0x100000f8
     Data Size:    9234816 Bytes = 8.8 MiB
     Architecture: AArch64
     OS:           Linux
     Load Address: 0x00080000
     Entry Point:  0x00080000
     Hash algo:    sha256
     Hash value:
5b46d9864f6d5cb91e6feb01293ccbe2e0a126afe9e55ac6bc124be877a2a758
   Verifying Hash Integrity ... sha256+ OK
## Loading ramdisk from FIT Image at 10000000 ...
   Using 'conf-system-top.dtb' configuration
   Trying 'ramdisk-1' ramdisk subimage
     Description:  petalinux-initramfs-image
     Type:         RAMDisk Image
     Compression:  uncompressed
     Data Start:   0x108d5fa0
     Data Size:    5107915 Bytes = 4.9 MiB
     Architecture: AArch64
     OS:           Linux
     Load Address: unavailable
     Entry Point:  unavailable
     Hash algo:    sha256
```

```
     Hash value:
095c3689da7cc27094ee64069f1a9419a6920ecf5f888dbdb8a8641e74e404fa
   Verifying Hash Integrity ... sha256+ OK
## Loading fdt from FIT Image at 10000000 ...
   Using 'conf-system-top.dtb' configuration
   Trying 'fdt-system-top.dtb' fdt subimage
     Description:  Flattened Device Tree blob
     Type:         Flat Device Tree
     Compression:  uncompressed
     Data Start:   0x108ceb88
     Data Size:    29505 Bytes = 28.8 KiB
     Architecture: AArch64
     Hash algo:    sha256
     Hash value:
4ea39107d5bca89d4c3efd0984bf2191b0ada397a25efa31d340c747b5d66046
   Verifying Hash Integrity ... sha256+ OK
   Booting using the fdt blob at 0x108ceb88
   Uncompressing Kernel Image
   Loading Ramdisk to 7d9f9000, end 7ded80cb ... OK
   Loading Device Tree to 000000007d9ee000, end 000000007d9f8340 ... OK

Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0x0000000000 [0x410fd083]
[    0.000000] Linux version 5.10.0-xilinx-v2020.2 (oe-user@oe-host)
(aarch64-xilinx-linux-gcc (GCC) 10.2.0, GNU ld (GNU Binutils) 2.35.1) #1
SMP Fri May 28 08:10:27 UTC 2021
[    0.000000] Machine model: Xilinx Versal vck190 Eval board revA (QSPI)
[    0.000000] earlycon: pl11 at MMIO32 0x00000000ff000000 (options
'115200n8')
[    0.000000] printk: bootconsole [pl11] enabled
[    0.000000] efi: UEFI not found.
[    0.000000] [Firmware Bug]: Kernel image misaligned at boot, please fix
your bootloader!
[    0.000000] cma: Reserved 256 MiB at 0x000000006d800000
[    0.000000] Zone ranges:
[    0.000000]   DMA32    [mem 0x0000000000000000-0x00000000ffffffff]
[    0.000000]   Normal   [mem 0x0000000100000000-0x00000501ffffffff]
[    0.000000] Movable zone start for each node
[    0.000000] Early memory node ranges
[    0.000000]   node   0: [mem 0x0000000000000000-0x000000007fffffff]
[    0.000000]   node   0: [mem 0x0000000800000000-0x000000097fffffff]
[    0.000000]   node   0: [mem 0x0000050000000000-0x00000501ffffffff]
[    0.000000] Initmem setup node 0 [mem
0x0000000000000000-0x00000501ffffffff]
[    0.000000] psci: probing for conduit method from DT.
[    0.000000] psci: PSCIv1.1 detected in firmware.
[    0.000000] psci: Using standard PSCI v0.2 function IDs
[    0.000000] psci: MIGRATE_INFO_TYPE not supported.
[    0.000000] psci: SMC Calling Convention v1.2
[    0.000000] percpu: Embedded 22 pages/cpu s50968 r8192 d30952 u90112
[    0.000000] Detected PIPT I-cache on CPU0
[    0.000000] CPU features: detected: GIC system register CPU interface
[    0.000000] CPU features: detected: Spectre-v2
[    0.000000] CPU features: detected: ARM errata 1165522, 1319367, or
1530923
[    0.000000] Built 1 zonelists, mobility grouping on.  Total pages:
4128768
[    0.000000] Kernel command line: console=ttyAMA0
earlycon=pl011,mmio32,0xFF000000,115200n8 clk_ignore_unused init_fatal_sh=1
[    0.000000] Dentry cache hash table entries: 2097152 (order: 12,
16777216 bytes, linear)
[    0.000000] Inode-cache hash table entries: 1048576 (order: 11, 8388608
```

```
bytes, linear)
[    0.000000] mem auto-init: stack:off, heap alloc:off, heap free:off
[    0.000000] software IO TLB: mapped [mem
0x0000000069800000-0x000000006d800000] (64MB)
[    0.000000] Memory: 16102944K/16777216K available (13760K kernel code,
972K rwdata, 3820K rodata, 2048K init, 588K bss, 412128K reserved, 262144K
cma-reserved)
[    0.000000] rcu: Hierarchical RCU implementation.
[    0.000000] rcu:     RCU event tracing is enabled.
[    0.000000] rcu:     RCU restricting CPUs from NR_CPUS=4 to nr_cpu_ids=2.
[    0.000000] rcu: RCU calculated value of scheduler-enlistment delay is
25 jiffies.
[    0.000000] rcu: Adjusting geometry for rcu_fanout_leaf=16, nr_cpu_ids=2
[    0.000000] NR_IRQS: 64, nr_irqs: 64, preallocated irqs: 0
[    0.000000] GICv3: GIC: Using split EOI/Deactivate mode
[    0.000000] GICv3: 160 SPIs implemented
[    0.000000] GICv3: 0 Extended SPIs implemented
[    0.000000] GICv3: Distributor has no Range Selector support
[    0.000000] GICv3: 16 PPIs implemented
[    0.000000] GICv3: CPU0: found redistributor 0 region
0:0x00000000f9080000
[    0.000000] ITS [mem 0xf9020000-0xf903ffff]
[    0.000000] ITS@0x00000000f9020000: allocated 65536 Devices @800080000
(flat, esz 8, psz 64K, shr 0)
[    0.000000] ITS: using cache flushing for cmd queue
[    0.000000] GICv3: using LPI property table @0x0000000800030000
[    0.000000] GIC: using cache flushing for LPI property table
[    0.000000] GICv3: CPU0: using allocated LPI pending table
@0x0000000800040000
[    0.000000] irq-xilinx: mismatch in kind-of-intr param
[    0.000000] irq-xilinx: /amba_pl@0/interrupt-controller@a5000000:
num_irq=32, sw_irq=0, edge=0x7fffffff
[    0.000000] irq-xilinx: mismatch in kind-of-intr param
[    0.000000] irq-xilinx: /amba_pl@0/interrupt-controller@a4000000:
num_irq=32, sw_irq=0, edge=0xffffffff
[    0.000000] random: get_random_bytes called from start_kernel+0x31c/
0x524 with crng_init=0
[    0.000000] arch_timer: cp15 timer(s) running at 100.00MHz (phys).
[    0.000000] clocksource: arch_sys_counter: mask: 0xffffffffffffff
max_cycles: 0x171024e7e0, max_idle_ns: 440795205315 ns
[    0.000002] sched_clock: 56 bits at 100MHz, resolution 10ns, wraps every
4398046511100ns
[    0.008347] Console: colour dummy device 80x25
[    0.012837] Calibrating delay loop (skipped), value calculated using
timer frequency.. 200.00 BogoMIPS (lpj=400000)
[    0.023359] pid_max: default: 32768 minimum: 301
[    0.028147] Mount-cache hash table entries: 32768 (order: 6, 262144
bytes, linear)
[    0.035831] Mountpoint-cache hash table entries: 32768 (order: 6, 262144
bytes, linear)
[    0.044631] rcu: Hierarchical SRCU implementation.
[    0.049574] Platform MSI: gic-its@f9020000 domain created
[    0.055074] PCI/MSI: /axi/interrupt-controller@f9000000/gic-its@f9020000
domain created
[    0.063187] EFI services will not be available.
[    0.067826] smp: Bringing up secondary CPUs ...
[    0.072702] Detected PIPT I-cache on CPU1
[    0.072724] GICv3: CPU1: found redistributor 1 region
0:0x00000000f90a0000
[    0.072731] GICv3: CPU1: using allocated LPI pending table
@0x0000000800050000
[    0.072753] CPU1: Booted secondary processor 0x0000000001 [0x410fd083]
[    0.072803] smp: Brought up 1 node, 2 CPUs
```

```
[    0.101734] SMP: Total of 2 processors activated.
[    0.106472] CPU features: detected: 32-bit EL0 Support
[    0.111649] CPU features: detected: CRC32 instructions
[    0.116857] CPU: All CPU(s) started at EL2
[    0.120994] alternatives: patching kernel code
[    0.126141] devtmpfs: initialized
[    0.131959] clocksource: jiffies: mask: 0xffffffff max_cycles:
0xffffffff, max_idle_ns: 7645041785100000 ns
[    0.141793] futex hash table entries: 512 (order: 3, 32768 bytes, linear)
[    0.151510] pinctrl core: initialized pinctrl subsystem
[    0.157087] DMI not present or invalid.
[    0.161062] NET: Registered protocol family 16
[    0.166141] DMA: preallocated 2048 KiB GFP_KERNEL pool for atomic
allocations
[    0.173534] DMA: preallocated 2048 KiB GFP_KERNEL|GFP_DMA32 pool for
atomic allocations
[    0.181632] audit: initializing netlink subsys (disabled)
[    0.187153] audit: type=2000 audit(0.120:1): state=initialized
audit_enabled=0 res=1
[    0.194967] cpuidle: using governor menu
[    0.198958] hw-breakpoint: found 6 breakpoint and 4 watchpoint registers.
[    0.205820] ASID allocator initialised with 65536 entries
[    0.211339] Serial: AMBA PL011 UART driver
[    0.226033] HugeTLB registered 1.00 GiB page size, pre-allocated 0 pages
[    0.232797] HugeTLB registered 32.0 MiB page size, pre-allocated 0 pages
[    0.239557] HugeTLB registered 2.00 MiB page size, pre-allocated 0 pages
[    0.246312] HugeTLB registered 64.0 KiB page size, pre-allocated 0 pages
[    0.804400] cryptd: max_cpu_qlen set to 1000
[    0.823105] DRBG: Continuing without Jitter RNG
[    0.900781] raid6: neonx8   gen()  3793 MB/s
[    0.973123] raid6: neonx8   xor()  2882 MB/s
[    1.045464] raid6: neonx4   gen()  3834 MB/s
[    1.117804] raid6: neonx4   xor()  3002 MB/s
[    1.190144] raid6: neonx2   gen()  3403 MB/s
[    1.262485] raid6: neonx2   xor()  2759 MB/s
[    1.334827] raid6: neonx1   gen()  2642 MB/s
[    1.407167] raid6: neonx1   xor()  2151 MB/s
[    1.479514] raid6: int64x8  gen()  2099 MB/s
[    1.551856] raid6: int64x8  xor()  1282 MB/s
[    1.624198] raid6: int64x4  gen()  2248 MB/s
[    1.696538] raid6: int64x4  xor()  1295 MB/s
[    1.768878] raid6: int64x2  gen()  2113 MB/s
[    1.841219] raid6: int64x2  xor()  1162 MB/s
[    1.913567] raid6: int64x1  gen()  1641 MB/s
[    1.985908] raid6: int64x1  xor()   865 MB/s
[    1.990206] raid6: using algorithm neonx4 gen() 3834 MB/s
[    1.995644] raid6: .... xor() 3002 MB/s, rmw enabled
[    2.000646] raid6: using neon recovery algorithm
[    2.005580] iommu: Default domain type: Translated
[    2.010648] SCSI subsystem initialized
[    2.014520] usbcore: registered new interface driver usbfs
[    2.020065] usbcore: registered new interface driver hub
[    2.025430] usbcore: registered new device driver usb
[    2.030554] mc: Linux media interface: v0.10
[    2.034865] videodev: Linux video capture interface: v2.00
[    2.040417] EDAC MC: Ver: 3.0.0
[    2.043892] zynqmp-ipi-mbox mailbox@ff3f0440: Registered ZynqMP IPI mbox
with TX/RX channels.
[    2.052602] FPGA manager framework
[    2.056107] Advanced Linux Sound Architecture Driver Initialized.
[    2.062452] Bluetooth: Core ver 2.22
[    2.066060] NET: Registered protocol family 31
[    2.070536] Bluetooth: HCI device and connection manager initialized
```

```
[    2.076940] Bluetooth: HCI socket layer initialized
[    2.081854] Bluetooth: L2CAP socket layer initialized
[    2.086949] Bluetooth: SCO socket layer initialized
[    2.092051] clocksource: Switched to clocksource arch_sys_counter
[    2.098305] VFS: Disk quotas dquot_6.6.0
[    2.102288] VFS: Dquot-cache hash table entries: 512 (order 0, 4096
bytes)
[    2.111809] NET: Registered protocol family 2
[    2.116488] tcp_listen_portaddr_hash hash table entries: 8192 (order: 5,
131072 bytes, linear)
[    2.125288] TCP established hash table entries: 131072 (order: 8,
1048576 bytes, linear)
[    2.134036] TCP bind hash table entries: 65536 (order: 8, 1048576 bytes,
linear)
[    2.142085] TCP: Hash tables configured (established 131072 bind 65536)
[    2.148840] UDP hash table entries: 8192 (order: 6, 262144 bytes, linear)
[    2.155925] UDP-Lite hash table entries: 8192 (order: 6, 262144 bytes,
linear)
[    2.163520] NET: Registered protocol family 1
[    2.168181] RPC: Registered named UNIX socket transport module.
[    2.174153] RPC: Registered udp transport module.
[    2.178890] RPC: Registered tcp transport module.
[    2.183626] RPC: Registered tcp NFSv4.1 backchannel transport module.
[    2.190422] PCI: CLS 0 bytes, default 64
[    2.194450] Trying to unpack rootfs image as initramfs...
[    2.374713] Freeing initrd memory: 4988K
[    2.406038] Initialise system trusted keyrings
[    2.410591] workingset: timestamp_bits=46 max_order=22 bucket_order=0
[    2.417679] NFS: Registering the id_resolver key type
[    2.422844] Key type id_resolver registered
[    2.427062] Key type id_legacy registered
[    2.431120] nfs4filelayout_init: NFSv4 File Layout Driver Registering...
[    2.437888] jffs2: version 2.2. (NAND) (SUMMARY)  © 2001-2006 Red Hat,
Inc.
[    2.473759] NET: Registered protocol family 38
[    2.478246] xor: measuring software checksum speed
[    2.484823]    8regs           :  5654 MB/sec
[    2.490718]    32regs          :  6520 MB/sec
[    2.496794]    arm64_neon      :  5834 MB/sec
[    2.501182] xor: using function: 32regs (6520 MB/sec)
[    2.506270] Key type asymmetric registered
[    2.510394] Asymmetric key parser 'x509' registered
[    2.515325] Block layer SCSI generic (bsg) driver version 0.4 loaded
(major 247)
[    2.522779] io scheduler mq-deadline registered
[    2.527341] io scheduler kyber registered
[    2.532244] ps_pcie_dma init()
[    2.550780] Serial: 8250/16550 driver, 4 ports, IRQ sharing disabled
[    2.557821] Serial: AMBA driver
[    2.561933] cacheinfo: Unable to detect cache hierarchy for CPU 0
[    2.571071] brd: module loaded
[    2.577160] loop: module loaded
[    2.580774] mtdoops: mtd device (mtddev=name/number) must be supplied
[    2.588148] libphy: Fixed MDIO Bus: probed
[    2.593058] tun: Universal TUN/TAP device driver, 1.6
[    2.598228] CAN device driver interface
[    2.602484] usbcore: registered new interface driver asix
[    2.607950] usbcore: registered new interface driver ax88179_178a
[    2.614102] usbcore: registered new interface driver cdc_ether
[    2.619992] usbcore: registered new interface driver net1080
[    2.625706] usbcore: registered new interface driver cdc_subset
[    2.631684] usbcore: registered new interface driver zaurus
[    2.637322] usbcore: registered new interface driver cdc_ncm
```

```
[    2.643489] usbcore: registered new interface driver uas
[    2.648866] usbcore: registered new interface driver usb-storage
[    2.655101] i2c /dev entries driver
[    2.659456] usbcore: registered new interface driver uvcvideo
[    2.665246] USB Video Class driver (1.1.1)
[    2.669606] Bluetooth: HCI UART driver ver 2.3
[    2.674088] Bluetooth: HCI UART protocol H4 registered
[    2.679266] Bluetooth: HCI UART protocol BCSP registered
[    2.684630] Bluetooth: HCI UART protocol LL registered
[    2.689804] Bluetooth: HCI UART protocol ATH3K registered
[    2.695247] Bluetooth: HCI UART protocol Three-wire (H5) registered
[    2.701585] Bluetooth: HCI UART protocol Intel registered
[    2.707031] Bluetooth: HCI UART protocol QCA registered
[    2.712313] usbcore: registered new interface driver bcm203x
[    2.718027] usbcore: registered new interface driver bpa10x
[    2.723654] usbcore: registered new interface driver bfusb
[    2.729197] usbcore: registered new interface driver btusb
[    2.734745] usbcore: registered new interface driver ath3k
[    2.740538] sdhci: Secure Digital Host Controller Interface driver
[    2.746769] sdhci: Copyright(c) Pierre Ossman
[    2.751157] sdhci-pltfm: SDHCI platform and OF driver helper
[    2.757006] ledtrig-cpu: registered to indicate activity on CPUs
[    2.763071] SMCCC: SOC_ID: ARCH_SOC_ID not implemented, skipping ....
[    2.769667] zynqmp_firmware_probe Platform Management API v1.0
[    2.775553] zynqmp_firmware_probe Trustzone version v1.0
[    2.781389] xlnx_event_manager xlnx_event_manager: Xilinx Event
Management driver probed
[    2.832884] usbcore: registered new interface driver usbhid
[    2.838507] usbhid: USB HID core driver
[    2.843004] sysmon f1270000.sysmon: Successfully registered Versal Sysmon
[    2.850142] ARM CCI_500 PMU driver probed
[    2.850348] fpga_manager fpga0: Xilinx Versal FPGA Manager registered
[    2.861230] pktgen: Packet Generator for packet performance testing.
Version: 2.75
[    2.869407] Initializing XFRM netlink socket
[    2.873761] NET: Registered protocol family 10
[    2.878537] Segment Routing with IPv6
[    2.882353] sit: IPv6, IPv4 and MPLS over IPv4 tunneling driver
[    2.888566] NET: Registered protocol family 17
[    2.893050] NET: Registered protocol family 15
[    2.897535] can: controller area network core
[    2.901939] NET: Registered protocol family 29
[    2.906417] can: raw protocol
[    2.909403] can: broadcast manager protocol
[    2.913617] can: netlink gateway - max_hops=1
[    2.918072] Bluetooth: RFCOMM TTY layer initialized
[    2.922996] Bluetooth: RFCOMM socket layer initialized
[    2.928185] Bluetooth: RFCOMM ver 1.11
[    2.931962] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
[    2.937313] Bluetooth: BNEP filters: protocol multicast
[    2.942581] Bluetooth: BNEP socket layer initialized
[    2.947583] Bluetooth: HIDP (Human Interface Emulation) ver 1.2
[    2.953552] Bluetooth: HIDP socket layer initialized
[    2.958657] 9pnet: Installing 9P2000 support
[    2.962970] Key type dns_resolver registered
[    2.967417] registered taskstats version 1
[    2.971550] Loading compiled-in X.509 certificates
[    2.977197] Btrfs loaded, crc32c=crc32c-generic
[    2.988043] ff000000.serial: ttyAMA0 at MMIO 0xff000000 (irq = 32,
base_baud = 0) is a SBSA
[    2.996499] printk: console [ttyAMA0] enabled
[    2.996499] printk: console [ttyAMA0] enabled
[    3.005241] printk: bootconsole [pl11] disabled
```

```
[    3.005241] printk: bootconsole [pl11] disabled
[    3.022481] xilinx-ai-engine aiepart_0_50: fpga bridge [xlnx-aie-
bridge-0-0] registered
[    3.030498] aie aie0: AI engine part(0,0),(50,9), id 1 is probed
successfully.
[    3.037714] xilinx-ai-engine 20000000000.ai_engine: Xilinx AI Engine
device(cols=50) probed
[    3.046321] of-fpga-region fpga: FPGA Region probed
[    3.051813] xilinx-zynqmp-dma ffa80000.dma: ZynqMP DMA driver Probe
success
[    3.058992] xilinx-zynqmp-dma ffa90000.dma: ZynqMP DMA driver Probe
success
[    3.066165] xilinx-zynqmp-dma ffaa0000.dma: ZynqMP DMA driver Probe
success
[    3.073334] xilinx-zynqmp-dma ffab0000.dma: ZynqMP DMA driver Probe
success
[    3.080500] xilinx-zynqmp-dma ffac0000.dma: ZynqMP DMA driver Probe
success
[    3.087669] xilinx-zynqmp-dma ffad0000.dma: ZynqMP DMA driver Probe
success
[    3.094834] xilinx-zynqmp-dma ffae0000.dma: ZynqMP DMA driver Probe
success
[    3.101999] xilinx-zynqmp-dma ffaf0000.dma: ZynqMP DMA driver Probe
success
[    3.109451] spi-nor spi0.0: found n25q00a, expected m25p80
[    3.115228] spi-nor spi0.0: trying to lock already unlocked area
[    3.121235] spi-nor spi0.0: n25q00a (262144 Kbytes)
[    3.126134] 4 fixed-partitions partitions found on MTD device spi0.0
[    3.132479] Creating 4 MTD partitions on "spi0.0":
[    3.137262] 0x000000000000-0x000000040000 : "boot"
[    3.142596] 0x000000040000-0x000000060000 : "bootenv"
[    3.148124] 0x000000060000-0x000000080000 : "config"
[    3.153531] 0x000000080000-0x000000c80000 : "kernel"
[    3.159745] macb ff0c0000.ethernet: Not enabling partial store and
forward
[    3.167127] libphy: MACB_mii_bus: probed
[    3.172700] macb ff0c0000.ethernet eth0: Cadence GEM rev 0x0107010b at
0xff0c0000 irq 24 (1e:e1:11:40:ef:d6)
[    3.182861] macb ff0d0000.ethernet: Not enabling partial store and
forward
[    3.207930] libphy: MACB_mii_bus: probed
[    3.211952] macb ff0d0000.ethernet eth1: Cadence GEM rev 0x0107010b at
0xff0d0000 irq 25 (f2:85:0f:5f:a3:5f)
[    3.222682] xhci-hcd xhci-hcd.0.auto: xHCI Host Controller
[    3.228179] xhci-hcd xhci-hcd.0.auto: new USB bus registered, assigned
bus number 1
[    3.235913] xhci-hcd xhci-hcd.0.auto: hcc params 0x0238fe65 hci version
0x110 quirks 0x0000000000010810
[    3.245323] xhci-hcd xhci-hcd.0.auto: irq 42, io mem 0xfe200000
[    3.251495] usb usb1: New USB device found, idVendor=1d6b,
idProduct=0002, bcdDevice= 5.10
[    3.259761] usb usb1: New USB device strings: Mfr=3, Product=2,
SerialNumber=1
[    3.266978] usb usb1: Product: xHCI Host Controller
[    3.271847] usb usb1: Manufacturer: Linux 5.10.0-xilinx-v2020.2 xhci-hcd
[    3.278539] usb usb1: SerialNumber: xhci-hcd.0.auto
[    3.283638] hub 1-0:1.0: USB hub found
[    3.287396] hub 1-0:1.0: 1 port detected
[    3.291542] xhci-hcd xhci-hcd.0.auto: xHCI Host Controller
[    3.297031] xhci-hcd xhci-hcd.0.auto: new USB bus registered, assigned
bus number 2
[    3.304690] xhci-hcd xhci-hcd.0.auto: Host supports USB 3.0 SuperSpeed
[    3.311376] usb usb2: New USB device found, idVendor=1d6b,
```

```
idProduct=0003, bcdDevice= 5.10
[    3.319639] usb usb2: New USB device strings: Mfr=3, Product=2,
SerialNumber=1
[    3.326861] usb usb2: Product: xHCI Host Controller
[    3.331731] usb usb2: Manufacturer: Linux 5.10.0-xilinx-v2020.2 xhci-hcd
[    3.338422] usb usb2: SerialNumber: xhci-hcd.0.auto
[    3.343480] hub 2-0:1.0: USB hub found
[    3.347365] hub 2-0:1.0: config failed, hub doesn't have any ports! (err
-19)
[    3.355252] rtc_zynqmp f12a0000.rtc: registered as rtc0
[    3.360497] rtc_zynqmp f12a0000.rtc: setting system clock to
2020-04-16T18:26:53 UTC (1587061613)
[    3.369735] cdns-i2c ff030000.i2c: 400 kHz mmio ff030000 irq 27
[    3.375993] cpu cpu0: dev_pm_opp_set_rate: failed to find current OPP
for freq 1349999987 (-34)
[    3.384745] cpufreq: cpufreq_online: CPU0: Running at unlisted initial
frequency: 1349999 KHz, changing to: 1199999 KHz
[    3.395534] cpu cpu0: dev_pm_opp_set_rate: failed to find current OPP
for freq 1349999987 (-34)
[    3.406407] of_cfs_init
[    3.408886] of_cfs_init: OK
[    3.411741] cfg80211: Loading compiled-in X.509 certificates for
regulatory database
[    3.436060] mmc0: SDHCI controller on f1050000.sdhci [f1050000.sdhci]
using ADMA 64-bit
[    3.478801] mmc0: new high speed SDHC card at address aaaa
[    3.484605] mmcblk0: mmc0:aaaa SC16G 14.8 GiB
[    3.487445] cfg80211: Loaded X.509 cert 'sforshee: 00b28ddf47aef9cea7'
[    3.495584] clk: Not disabling unused clocks
[    3.499708]  mmcblk0: p1
[    3.500407] ALSA device list:
[    3.505322]   No soundcards found.
[    3.508990] platform regulatory.0: Direct firmware load for
regulatory.db failed with error -2
[    3.517604] cfg80211: failed to load regulatory.db
[    3.522975] Freeing unused kernel memory: 2048K
[    3.536107] Run /init as init process
udhcpc: started, v1.32.0
[    3.604931] macb ff0c0000.ethernet eth0: PHY [ff0c0000.ethernet-
ffffffff:01] driver [TI DP83867] (irq=POLL)
[    3.614678] macb ff0c0000.ethernet eth0: configuring for phy/rgmii-id
link mode
udhcpc: sending discover
[    4.504060] usb 1-1: new high-speed USB device number 2 using xhci-hcd
[    4.656335] usb 1-1: New USB device found, idVendor=0781,
idProduct=5567, bcdDevice= 1.00
[    4.664508] usb 1-1: New USB device strings: Mfr=1, Product=2,
SerialNumber=3
[    4.671635] usb 1-1: Product: Cruzer Blade
[    4.675723] usb 1-1: Manufacturer: SanDisk
[    4.679811] usb 1-1: SerialNumber: 4C530000220130209065
[    4.685530] usb-storage 1-1:1.0: USB Mass Storage device detected
[    4.691807] scsi host0: usb-storage 1-1:1.0
[    5.712756] scsi 0:0:0:0: Direct-Access     SanDisk  Cruzer Blade
1.00 PQ: 0 ANSI: 6
[    5.723337] sd 0:0:0:0: [sda] 30031872 512-byte logical blocks: (15.4 GB/
14.3 GiB)
[    5.731610] sd 0:0:0:0: [sda] Write Protect is off
[    5.736611] sd 0:0:0:0: [sda] Write cache: disabled, read cache:
enabled, doesn't support DPO or FUA
[    5.768842]  sda:
[    5.774428] sd 0:0:0:0: [sda] Attached SCSI removable disk
udhcpc: sending discover
```

```
[    6.673314] macb ff0c0000.ethernet eth0: Link is Up - 1Gbps/Full - flow
control off
[    6.681082] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
udhcpc: sending discover
udhcpc: sending select for 10.10.71.3
udhcpc: lease of 10.10.71.3 obtained, lease time 600
ERROR: There's no '/dev' on rootfs.

sh: can't access tty; job control turned off
/ #
```

Troubleshooting

This section describes some common issues you may experience while booting a PetaLinux image on hardware with SD card.

Table 9: **PetaLinux Image on Hardware Troubleshooting**

Problem / Error Message	Description and Solution

`Wrong Image Format for boot command.`
`ERROR: Can't get kernel image!`
	Problem Description:
	This error message indicates that the U-Boot boot loader is unable to find kernel image. This is likely because `bootcmd` environment variable is not set properly.
	Solution:
	To see the default boot device, print `bootcmd` environment variable using the following command in U-Boot console.
	`U-Boot-PetaLinux> print bootcmd`
	If it is not run using sdboot flow, there are a few options as follows:
	• Without rebuild PetaLinux, set `bootcmd` to boot from your desired media, use `setenv` command. For SD card boot, set the environment variable as follows.
	`U-Boot-PetaLinux> setenv bootcmd 'run sdboot' ; saveenv`
	• Run `petalinux-config` to set to load kernel image from SD card. For more information, see the Boot Images Storage Configuration. Rebuild PetaLinux and regenerate `BOOT.BIN` with the rebuilt U-Boot, and then use the new `BOOT.BIN` to boot the board. See Generate Boot Image for Zynq UltraScale+ MPSoC on how to generate `BOOT.BIN`.

TIP: *To know more about U-Boot options, use the command:* `$ U-Boot-PetaLinux> printenv`.

Boot a PetaLinux Image on Hardware with JTAG

This section describes how to boot a PetaLinux image on hardware with JTAG.

The JTAG boot communicates with XSDB which in turn communicates with the hw_server. The TCP port used is 3121; ensure that the firewall is disabled for this port.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have a PetaLinux system image by either installing a PetaLinux BSP (see Project Creation Using PetaLinux BSP) or by building your own PetaLinux project (see Build System Image).
- If you wish to make use of prebuilt capability for JTAG boot. You need to have packaged prebuilt images (see Packaging Prebuilt Images).
- A serial communication program such as minicom/kermit/gtkterm has been installed; the baud rate of the serial communication program has been set to 115200 bps.
- Appropriate JTAG cable drivers have been installed.

Steps to Boot a PetaLinux Image on Hardware with JTAG

1. Power off the board.
2. Connect the JTAG port on the board with the JTAG cable to your workstation.
3. Connect the serial port on the board to your workstation.
4. If your system has Ethernet, also connect the Ethernet port on the board to your local network.
5. Ensure that the mode switches are set to JTAG mode. Refer to the board documentation for details.
6. Power on the board.
7. Open a console on your workstation and start with preferred serial communication program (For example, kermit, minicom) with the baud rate set to 115200 on that console.
8. Run the `petalinux-boot` command as follows on your workstation:

   ```
   $ petalinux-boot --jtag --prebuilt 3 --hw_server-url
   <hostname:3121>
   ```

   Note: If you wish not to use prebuilt capability for JTAG boot, refer to Additional Options for Booting with JTAG.

   The `--jtag` option tells `petalinux-boot` to boot on hardware via JTAG, and the `--prebuilt 3` option boots the Linux kernel. Wait for the appearance of the shell prompt on the command console to indicate completion of the command.

   Note: To know more about different boot levels for prebuilt option, see Using petalinux-boot Command with Prebuilt Images.

The example of the message on the workstation command console for successful `petalinux-boot` is:

```
[3.958]PLM Initialization Time
[4.021]***********Boot PDI Load: Started***********
[4.099]Loading PDI from JTAG
[4.159]Monolithic/Master Device
[4.285]0.153 ms: PDI initialization time
[4.359]+++Loading Image#: 0x1, Name: lpd, Id: 0x04210002
[4.444]---Loading Partition#: 0x1, Id: 0xC
[27.674]****************************************
[31.900]Xilinx Versal Platform Loader and Manager
[36.295]Release 2021.1   Jun  1 2021  -  15:25:35
[40.607]Platform Version: v2.0 PMC: v2.0, PS: v2.0
[45.002]BOOTMODE: 0x0, MULTIBOOT: 0x0
[48.299]****************************************
[52.564] 48.010 ms for Partition#: 0x1, Size: 2288 Bytes
[57.426]---Loading Partition#: 0x2, Id: 0x0
[61.780] 0.516 ms for Partition#: 0x2, Size: 48 Bytes
[65.881]---Loading Partition#: 0x3, Id: 0x0
[106.053] 36.331 ms for Partition#: 0x3, Size: 59280 Bytes
[108.364]---Loading Partition#: 0x4, Id: 0x0
[112.297] 0.012 ms for Partition#: 0x4, Size: 2640 Bytes
[117.154]---Loading Partition#: 0x5, Id: 0x0
[121.089] 0.014 ms for Partition#: 0x5, Size: 3440 Bytes
[125.944]---Loading Partition#: 0x6, Id: 0x0
[129.871] 0.007 ms for Partition#: 0x6, Size: 80 Bytes
PSM Firmware version: 2021.1 [Build: Jun  1 2021 15:25:35 ]
[139.818]+++Loading Image#: 0x2, Name: pl_cfi, Id: 0x18700000
[145.126]---Loading Partition#: 0x7, Id: 0x3
[985.682] 836.632 ms for Partition#: 0x7, Size: 1304480 Bytes
[988.250]---Loading Partition#: 0x8, Id: 0x5
[1604.539] 612.366 ms for Partition#: 0x8, Size: 933344 Bytes
[1607.150]+++Loading Image#: 0x3, Name: aie_subsys, Id: 0x0421C005
[1612.862]---Loading Partition#: 0x9, Id: 0x7
[1619.981] 3.108 ms for Partition#: 0x9, Size: 352 Bytes
[1622.170]+++Loading Image#: 0x4, Name: fpd, Id: 0x0420C003
[1627.291]---Loading Partition#: 0xA, Id: 0x8
[1631.722] 0.422 ms for Partition#: 0xA, Size: 1040 Bytes
[1636.526]+++Loading Image#: 0x5, Name: apu_subsystem, Id: 0x1C000000
[1642.264]---Loading Partition#: 0xB, Id: 0x0
[1660.095] 13.821 ms for Partition#: 0xB, Size: 29520 Bytes
[1662.496]---Loading Partition#: 0xC, Id: 0x0
[1702.715] 36.209 ms for Partition#: 0xC, Size: 59344 Bytes
[1705.118]---Loading Partition#: 0xD, Id: 0x0
[2352.939] 643.811 ms for Partition#: 0xD, Size: 1001184 Bytes
N[2I55.6 3]****n******BootlPDI Load:  Sne***********
 [2360: 11]316211.9 0 ms: ROM Time
[2333.210] stal P MoBoot Tixe
000
NOTICE:  BL31: v2.4(debug):v1.1-7609-g851523ea2
NOTICE:  BL31: Built : 08:27:07, Apr 28 2021
INFO:    GICv3 with legacy support detected.
INFO:    ARM GICv3 driver initialized in EL3
INFO:    BL31: Initializing runtime services
WARNING: BL31: cortex_a72: CPU workaround for 859971 was missing!
WARNING: BL31: cortex_a72: CPU workaround for 1319367 was missing!
INFO:    BL31: cortex_a72: CPU workaround for cve_2017_5715 was applied
INFO:    BL31: cortex_a72: CPU workaround for cve_2018_3639 was applied
INFO:    BL31: Preparing for EL3 exit to normal world
INFO:    Entry point address = 0x8000000
INFO:    SPSR = 0x3c9
```

```
U-Boot 2021.01 (May 28 2021 - 10:52:42 +0000)

Model: Xilinx Versal vck190 Eval board revA (QSPI)
DRAM:  16 GiB
EL Level:    EL2
MMC:   sdhci@f1050000: 0
In:    serial@ff000000
Out:   serial@ff000000
Err:   serial@ff000000
Bootmode: JTAG_MODE
Net:
ZYNQ GEM: ff0c0000, mdio bus ff0c0000, phyaddr 1, interface rgmii-id
Could not get PHY for eth0: addr 1

ZYNQ GEM: ff0d0000, mdio bus ff0c0000, phyaddr 2, interface rgmii-id

Warning: ethernet@ff0d0000 (eth1) using random MAC address -
3a:7f:e2:ba:77:6b
eth1: ethernet@ff0d0000
Hit any key to stop autoboot:  0
JTAG: Trying to boot script at 20000000
## Executing script at 20000000
Trying to load boot images from jtag
## Loading init Ramdisk from Legacy Image at 04000000 ...
   Image Name:   petalinux-initramfs-image-versal
   Image Type:   AArch64 Linux RAMDisk Image (uncompressed)
   Data Size:    5107909 Bytes = 4.9 MiB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 00001000
   Booting using the fdt blob at 0x001000

ZYNQ GEM: ff0c0000, mdio bus ff0c0000, phyaddr 1, interface rgmii-id
Could not get PHY for eth0: addr 1
   Loading Ramdisk to 7d9fa000, end 7ded90c5 ... OK
   Loading Device Tree to 000000007d9ef000, end 000000007d9f9340 ... OK

Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0x0000000000 [0x410fd083]
[    0.000000] Linux version 5.10.0-xilinx-v2020.2 (oe-user@oe-host)
(aarch64-xilinx-linux-gcc (GCC) 10.2.0, GNU ld (GNU Binutils) 2.35.1) #1
SMP Fri May 28 08:10:27 UTC 2021
[    0.000000] Machine model: Xilinx Versal vck190 Eval board revA (QSPI)
[    0.000000] earlycon: pl11 at MMIO32 0x00000000ff000000 (options
'115200n8')
[    0.000000] printk: bootconsole [pl11] enabled
[    0.000000] efi: UEFI not found.
[    0.000000] cma: Reserved 256 MiB at 0x000000006d800000
[    0.000000] Zone ranges:
[    0.000000]   DMA32    [mem 0x0000000000000000-0x00000000ffffffff]
[    0.000000]   Normal   [mem 0x0000000100000000-0x00000501ffffffff]
[    0.000000] Movable zone start for each node
[    0.000000] Early memory node ranges
[    0.000000]   node   0: [mem 0x0000000000000000-0x000000007fffffff]
[    0.000000]   node   0: [mem 0x0000000800000000-0x000000097fffffff]
[    0.000000]   node   0: [mem 0x0000050000000000-0x00000501ffffffff]
[    0.000000] Initmem setup node 0 [mem
0x0000000000000000-0x00000501ffffffff]
[    0.000000] psci: probing for conduit method from DT.
```

```
[    0.000000] psci: PSCIv1.1 detected in firmware.
[    0.000000] psci: Using standard PSCI v0.2 function IDs
[    0.000000] psci: MIGRATE_INFO_TYPE not supported.
[    0.000000] psci: SMC Calling Convention v1.2
[    0.000000] percpu: Embedded 22 pages/cpu s50968 r8192 d30952 u90112
[    0.000000] Detected PIPT I-cache on CPU0
[    0.000000] CPU features: detected: GIC system register CPU interface
[    0.000000] CPU features: detected: Spectre-v2
[    0.000000] CPU features: detected: ARM errata 1165522, 1319367, or
1530923
[    0.000000] Built 1 zonelists, mobility grouping on.  Total pages:
4128768
[    0.000000] Kernel command line: console=ttyAMA0
earlycon=pl011,mmio32,0xFF000000,115200n8 clk_ignore_unused
init_fatal_sh=1
[    0.000000] Dentry cache hash table entries: 2097152 (order: 12,
16777216 bytes, linear)
[    0.000000] Inode-cache hash table entries: 1048576 (order: 11,
8388608 bytes, linear)
[    0.000000] mem auto-init: stack:off, heap alloc:off, heap free:off
[    0.000000] software IO TLB: mapped [mem
0x0000000069800000-0x000000006d800000] (64MB)
[    0.000000] Memory: 16102948K/16777216K available (13760K kernel
code, 972K rwdata, 3820K rodata, 2048K init, 588K bss, 412124K reserved,
262144K cma-reserved)
[    0.000000] rcu: Hierarchical RCU implementation.
[    0.000000] rcu:     RCU event tracing is enabled.
[    0.000000] rcu:     RCU restricting CPUs from NR_CPUS=4 to
nr_cpu_ids=2.
[    0.000000] rcu: RCU calculated value of scheduler-enlistment delay
is 25 jiffies.
[    0.000000] rcu: Adjusting geometry for rcu_fanout_leaf=16,
nr_cpu_ids=2
[    0.000000] NR_IRQS: 64, nr_irqs: 64, preallocated irqs: 0
[    0.000000] GICv3: GIC: Using split EOI/Deactivate mode
[    0.000000] GICv3: 160 SPIs implemented
[    0.000000] GICv3: 0 Extended SPIs implemented
[    0.000000] GICv3: Distributor has no Range Selector support
[    0.000000] GICv3: 16 PPIs implemented
[    0.000000] GICv3: CPU0: found redistributor 0 region
0:0x00000000f9080000
[    0.000000] ITS [mem 0xf9020000-0xf903ffff]
[    0.000000] ITS@0x00000000f9020000: allocated 65536 Devices
@800080000 (flat, esz 8, psz 64K, shr 0)
[    0.000000] ITS: using cache flushing for cmd queue
[    0.000000] GICv3: using LPI property table @0x0000000800030000
[    0.000000] GIC: using cache flushing for LPI property table
[    0.000000] GICv3: CPU0: using allocated LPI pending table
@0x0000000800040000
[    0.000000] irq-xilinx: mismatch in kind-of-intr param
[    0.000000] irq-xilinx: /amba_pl@0/interrupt-controller@a5000000:
num_irq=32, sw_irq=0, edge=0x7fffffff
[    0.000000] irq-xilinx: mismatch in kind-of-intr param
[    0.000000] irq-xilinx: /amba_pl@0/interrupt-controller@a4000000:
num_irq=32, sw_irq=0, edge=0xffffffff
[    0.000000] random: get_random_bytes called from start_kernel+0x31c/
0x524 with crng_init=0
[    0.000000] arch_timer: cp15 timer(s) running at 100.00MHz (phys).
[    0.000000] clocksource: arch_sys_counter: mask: 0xffffffffffffff
max_cycles: 0x171024e7e0, max_idle_ns: 440795205315 ns
[    0.000002] sched_clock: 56 bits at 100MHz, resolution 10ns, wraps
every 4398046511100ns
[    0.008349] Console: colour dummy device 80x25
```

```
[    0.012837] Calibrating delay loop (skipped), value calculated using
timer frequency.. 200.00 BogoMIPS (lpj=400000)
[    0.023367] pid_max: default: 32768 minimum: 301
[    0.028155] Mount-cache hash table entries: 32768 (order: 6, 262144
bytes, linear)
[    0.035841] Mountpoint-cache hash table entries: 32768 (order: 6,
262144 bytes, linear)
[    0.044643] rcu: Hierarchical SRCU implementation.
[    0.049586] Platform MSI: gic-its@f9020000 domain created
[    0.055088] PCI/MSI: /axi/interrupt-controller@f9000000/gic-
its@f9020000 domain created
[    0.063207] EFI services will not be available.
[    0.067846] smp: Bringing up secondary CPUs ...
[    0.072717] Detected PIPT I-cache on CPU1
[    0.072738] GICv3: CPU1: found redistributor 1 region
0:0x00000000f90a0000
[    0.072744] GICv3: CPU1: using allocated LPI pending table
@0x0000000800050000
[    0.072768] CPU1: Booted secondary processor 0x0000000001 [0x410fd083]
[    0.072819] smp: Brought up 1 node, 2 CPUs
[    0.101767] SMP: Total of 2 processors activated.
[    0.106507] CPU features: detected: 32-bit EL0 Support
[    0.111684] CPU features: detected: CRC32 instructions
[    0.116890] CPU: All CPU(s) started at EL2
[    0.121029] alternatives: patching kernel code
[    0.126157] devtmpfs: initialized
[    0.131979] clocksource: jiffies: mask: 0xffffffff max_cycles:
0xffffffff, max_idle_ns: 7645041785100000 ns
[    0.141816] futex hash table entries: 512 (order: 3, 32768 bytes,
linear)
[    0.151529] pinctrl core: initialized pinctrl subsystem
[    0.157106] DMI not present or invalid.
[    0.161080] NET: Registered protocol family 16
[    0.166168] DMA: preallocated 2048 KiB GFP_KERNEL pool for atomic
allocations
[    0.173560] DMA: preallocated 2048 KiB GFP_KERNEL|GFP_DMA32 pool for
atomic allocations
[    0.181659] audit: initializing netlink subsys (disabled)
[    0.187178] audit: type=2000 audit(0.120:1): state=initialized
audit_enabled=0 res=1
[    0.194998] cpuidle: using governor menu
[    0.198994] hw-breakpoint: found 6 breakpoint and 4 watchpoint
registers.
[    0.205857] ASID allocator initialised with 65536 entries
[    0.211374] Serial: AMBA PL011 UART driver
[    0.226066] HugeTLB registered 1.00 GiB page size, pre-allocated 0
pages
[    0.232834] HugeTLB registered 32.0 MiB page size, pre-allocated 0
pages
[    0.239591] HugeTLB registered 2.00 MiB page size, pre-allocated 0
pages
[    0.246348] HugeTLB registered 64.0 KiB page size, pre-allocated 0
pages
[    0.804516] cryptd: max_cpu_qlen set to 1000
[    0.823256] DRBG: Continuing without Jitter RNG
[    0.900908] raid6: neonx8   gen()  3799 MB/s
[    0.973252] raid6: neonx8   xor()  2882 MB/s
[    1.045598] raid6: neonx4   gen()  3833 MB/s
[    1.117938] raid6: neonx4   xor()  3002 MB/s
[    1.190284] raid6: neonx2   gen()  3392 MB/s
[    1.262627] raid6: neonx2   xor()  2759 MB/s
[    1.334972] raid6: neonx1   gen()  2639 MB/s
[    1.407310] raid6: neonx1   xor()  2151 MB/s
```

```
[    1.479661] raid6: int64x8  gen()  2099 MB/s
[    1.552004] raid6: int64x8  xor()  1282 MB/s
[    1.624350] raid6: int64x4  gen()  2248 MB/s
[    1.696690] raid6: int64x4  xor()  1295 MB/s
[    1.769041] raid6: int64x2  gen()  2119 MB/s
[    1.841383] raid6: int64x2  xor()  1162 MB/s
[    1.913730] raid6: int64x1  gen()  1640 MB/s
[    1.986072] raid6: int64x1  xor()   865 MB/s
[    1.990373] raid6: using algorithm neonx4 gen() 3833 MB/s
[    1.995814] raid6: .... xor() 3002 MB/s, rmw enabled
[    2.000815] raid6: using neon recovery algorithm
[    2.005751] iommu: Default domain type: Translated
[    2.010832] SCSI subsystem initialized
[    2.014705] usbcore: registered new interface driver usbfs
[    2.020251] usbcore: registered new interface driver hub
[    2.025621] usbcore: registered new device driver usb
[    2.030746] mc: Linux media interface: v0.10
[    2.035057] videodev: Linux video capture interface: v2.00
[    2.040612] EDAC MC: Ver: 3.0.0
[    2.044090] zynqmp-ipi-mbox mailbox@ff3f0440: Registered ZynqMP IPI
mbox with TX/RX channels.
[    2.052800] FPGA manager framework
[    2.056303] Advanced Linux Sound Architecture Driver Initialized.
[    2.062642] Bluetooth: Core ver 2.22
[    2.066253] NET: Registered protocol family 31
[    2.070732] Bluetooth: HCI device and connection manager initialized
[    2.077135] Bluetooth: HCI socket layer initialized
[    2.082049] Bluetooth: L2CAP socket layer initialized
[    2.087145] Bluetooth: SCO socket layer initialized
[    2.092235] clocksource: Switched to clocksource arch_sys_counter
[    2.098491] VFS: Disk quotas dquot_6.6.0
[    2.102479] VFS: Dquot-cache hash table entries: 512 (order 0, 4096
bytes)
[    2.112003] NET: Registered protocol family 2
[    2.116682] tcp_listen_portaddr_hash hash table entries: 8192 (order:
5, 131072 bytes, linear)
[    2.125489] TCP established hash table entries: 131072 (order: 8,
1048576 bytes, linear)
[    2.134244] TCP bind hash table entries: 65536 (order: 8, 1048576
bytes, linear)
[    2.142295] TCP: Hash tables configured (established 131072 bind
65536)
[    2.149058] UDP hash table entries: 8192 (order: 6, 262144 bytes,
linear)
[    2.156134] UDP-Lite hash table entries: 8192 (order: 6, 262144
bytes, linear)
[    2.163730] NET: Registered protocol family 1
[    2.168405] RPC: Registered named UNIX socket transport module.
[    2.174379] RPC: Registered udp transport module.
[    2.179124] RPC: Registered tcp transport module.
[    2.183862] RPC: Registered tcp NFSv4.1 backchannel transport module.
[    2.190661] PCI: CLS 0 bytes, default 64
[    2.194690] Trying to unpack rootfs image as initramfs...
[    2.375100] Freeing initrd memory: 4988K
[    2.405447] Initialise system trusted keyrings
[    2.410022] workingset: timestamp_bits=46 max_order=22 bucket_order=0
[    2.417049] NFS: Registering the id_resolver key type
[    2.422159] Key type id_resolver registered
[    2.426383] Key type id_legacy registered
[    2.430438] nfs4filelayout_init: NFSv4 File Layout Driver
Registering...
[    2.437208] jffs2: version 2.2. (NAND) (SUMMARY)  © 2001-2006 Red
Hat, Inc.
```

```
[     2.473425] NET: Registered protocol family 38
[     2.477913] xor: measuring software checksum speed
[     2.484487]    8regs           :  5654 MB/sec
[     2.490382]    32regs          :  6519 MB/sec
[     2.496458]    arm64_neon      :  5834 MB/sec
[     2.500847] xor: using function: 32regs (6519 MB/sec)
[     2.505939] Key type asymmetric registered
[     2.510066] Asymmetric key parser 'x509' registered
[     2.514996] Block layer SCSI generic (bsg) driver version 0.4 loaded
(major 247)
[     2.522453] io scheduler mq-deadline registered
[     2.527019] io scheduler kyber registered
[     2.531904] ps_pcie_dma init()
[     2.550411] Serial: 8250/16550 driver, 4 ports, IRQ sharing disabled
[     2.557454] Serial: AMBA driver
[     2.561564] cacheinfo: Unable to detect cache hierarchy for CPU 0
[     2.570707] brd: module loaded
[     2.576795] loop: module loaded
[     2.580414] mtdoops: mtd device (mtddev=name/number) must be supplied
[     2.587788] libphy: Fixed MDIO Bus: probed
[     2.592721] tun: Universal TUN/TAP device driver, 1.6
[     2.597895] CAN device driver interface
[     2.602146] usbcore: registered new interface driver asix
[     2.607615] usbcore: registered new interface driver ax88179_178a
[     2.613772] usbcore: registered new interface driver cdc_ether
[     2.619665] usbcore: registered new interface driver net1080
[     2.625381] usbcore: registered new interface driver cdc_subset
[     2.631360] usbcore: registered new interface driver zaurus
[     2.636994] usbcore: registered new interface driver cdc_ncm
[     2.643161] usbcore: registered new interface driver uas
[     2.648537] usbcore: registered new interface driver usb-storage
[     2.654772] i2c /dev entries driver
[     2.659126] usbcore: registered new interface driver uvcvideo
[     2.664918] USB Video Class driver (1.1.1)
[     2.669277] Bluetooth: HCI UART driver ver 2.3
[     2.673761] Bluetooth: HCI UART protocol H4 registered
[     2.678939] Bluetooth: HCI UART protocol BCSP registered
[     2.684305] Bluetooth: HCI UART protocol LL registered
[     2.689484] Bluetooth: HCI UART protocol ATH3K registered
[     2.694931] Bluetooth: HCI UART protocol Three-wire (H5) registered
[     2.701276] Bluetooth: HCI UART protocol Intel registered
[     2.706725] Bluetooth: HCI UART protocol QCA registered
[     2.712007] usbcore: registered new interface driver bcm203x
[     2.717725] usbcore: registered new interface driver bpa10x
[     2.723356] usbcore: registered new interface driver bfusb
[     2.728900] usbcore: registered new interface driver btusb
[     2.734450] usbcore: registered new interface driver ath3k
[     2.740252] sdhci: Secure Digital Host Controller Interface driver
[     2.746480] sdhci: Copyright(c) Pierre Ossman
[     2.750868] sdhci-pltfm: SDHCI platform and OF driver helper
[     2.756720] ledtrig-cpu: registered to indicate activity on CPUs
[     2.762786] SMCCC: SOC_ID: ARCH_SOC_ID not implemented, skipping ....
[     2.769382] zynqmp_firmware_probe Platform Management API v1.0
[     2.775266] zynqmp_firmware_probe Trustzone version v1.0
[     2.781109] xlnx_event_manager xlnx_event_manager: Xilinx Event
Management driver probed
[     2.832727] usbcore: registered new interface driver usbhid
[     2.838354] usbhid: USB HID core driver
[     2.842855] sysmon f1270000.sysmon: Successfully registered Versal
Sysmon
[     2.849990] ARM CCI_500 PMU driver probed
[     2.850191] fpga_manager fpga0: Xilinx Versal FPGA Manager registered
[     2.861082] pktgen: Packet Generator for packet performance testing.
```

```
Version: 2.75
[    2.869262] Initializing XFRM netlink socket
[    2.873614] NET: Registered protocol family 10
[    2.878396] Segment Routing with IPv6
[    2.882217] sit: IPv6, IPv4 and MPLS over IPv4 tunneling driver
[    2.888452] NET: Registered protocol family 17
[    2.892938] NET: Registered protocol family 15
[    2.897426] can: controller area network core
[    2.901836] NET: Registered protocol family 29
[    2.906314] can: raw protocol
[    2.909302] can: broadcast manager protocol
[    2.913516] can: netlink gateway - max_hops=1
[    2.917984] Bluetooth: RFCOMM TTY layer initialized
[    2.922907] Bluetooth: RFCOMM socket layer initialized
[    2.928096] Bluetooth: RFCOMM ver 1.11
[    2.931880] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
[    2.937233] Bluetooth: BNEP filters: protocol multicast
[    2.942503] Bluetooth: BNEP socket layer initialized
[    2.947504] Bluetooth: HIDP (Human Interface Emulation) ver 1.2
[    2.953475] Bluetooth: HIDP socket layer initialized
[    2.958582] 9pnet: Installing 9P2000 support
[    2.962895] Key type dns_resolver registered
[    2.967331] registered taskstats version 1
[    2.971465] Loading compiled-in X.509 certificates
[    2.977123] Btrfs loaded, crc32c=crc32c-generic
[    2.987963] ff000000.serial: ttyAMA0 at MMIO 0xff000000 (irq = 32,
base_baud = 0) is a SBSA
[    2.996422] printk: console [ttyAMA0] enabled
[    2.996422] printk: console [ttyAMA0] enabled
[    3.005166] printk: bootconsole [pl11] disabled
[    3.005166] printk: bootconsole [pl11] disabled
[    3.022407] xilinx-ai-engine aiepart_0_50: fpga bridge [xlnx-aie-
bridge-0-0] registered
[    3.030423] aie aie0: AI engine part(0,0),(50,9), id 1 is probed
successfully.
[    3.037639] xilinx-ai-engine 20000000000.ai_engine: Xilinx AI Engine
device(cols=50) probed
[    3.046261] of-fpga-region fpga: FPGA Region probed
[    3.051742] xilinx-zynqmp-dma ffa80000.dma: ZynqMP DMA driver Probe
success
[    3.058924] xilinx-zynqmp-dma ffa90000.dma: ZynqMP DMA driver Probe
success
[    3.066098] xilinx-zynqmp-dma ffaa0000.dma: ZynqMP DMA driver Probe
success
[    3.073263] xilinx-zynqmp-dma ffab0000.dma: ZynqMP DMA driver Probe
success
[    3.080432] xilinx-zynqmp-dma ffac0000.dma: ZynqMP DMA driver Probe
success
[    3.087611] xilinx-zynqmp-dma ffad0000.dma: ZynqMP DMA driver Probe
success
[    3.094778] xilinx-zynqmp-dma ffae0000.dma: ZynqMP DMA driver Probe
success
[    3.101940] xilinx-zynqmp-dma ffaf0000.dma: ZynqMP DMA driver Probe
success
[    3.109390] spi-nor spi0.0: found n25q00a, expected m25p80
[    3.115168] spi-nor spi0.0: trying to lock already unlocked area
[    3.121175] spi-nor spi0.0: n25q00a (262144 Kbytes)
[    3.126071] 4 fixed-partitions partitions found on MTD device spi0.0
[    3.132417] Creating 4 MTD partitions on "spi0.0":
[    3.137200] 0x000000000000-0x000000040000 : "boot"
[    3.142542] 0x000000040000-0x000000060000 : "bootenv"
[    3.148062] 0x000000060000-0x000000080000 : "config"
[    3.153474] 0x000000080000-0x000000c80000 : "kernel"
```

```
[    3.159720] macb ff0c0000.ethernet: Not enabling partial store and
forward
[    3.166627] macb ff0c0000.ethernet: invalid hw address, using random
[    3.173439] libphy: MACB_mii_bus: probed
[    3.177508] mdio_bus ff0c0000.ethernet-ffffffff: MDIO device at
address 1 is missing.
[    3.186233] macb ff0c0000.ethernet eth0: Cadence GEM rev 0x0107010b
at 0xff0c0000 irq 24 (1e:f8:0a:1a:3a:2b)
[    3.196390] macb ff0d0000.ethernet: Not enabling partial store and
forward
[    3.221435] libphy: MACB_mii_bus: probed
[    3.225503] macb ff0d0000.ethernet eth1: Cadence GEM rev 0x0107010b
at 0xff0d0000 irq 25 (3a:7f:e2:ba:77:6b)
[    3.236239] xhci-hcd xhci-hcd.0.auto: xHCI Host Controller
[    3.241728] xhci-hcd xhci-hcd.0.auto: new USB bus registered,
assigned bus number 1
[    3.249459] xhci-hcd xhci-hcd.0.auto: hcc params 0x0238fe65 hci
version 0x110 quirks 0x0000000000010810
[    3.258867] xhci-hcd xhci-hcd.0.auto: irq 42, io mem 0xfe200000
[    3.264989] usb usb1: New USB device found, idVendor=1d6b,
idProduct=0002, bcdDevice= 5.10
[    3.273258] usb usb1: New USB device strings: Mfr=3, Product=2,
SerialNumber=1
[    3.280593] usb usb1: Product: xHCI Host Controller
[    3.285463] usb usb1: Manufacturer: Linux 5.10.0-xilinx-v2020.2 xhci-
hcd
[    3.292154] usb usb1: SerialNumber: xhci-hcd.0.auto
[    3.297253] hub 1-0:1.0: USB hub found
[    3.301012] hub 1-0:1.0: 1 port detected
[    3.305113] xhci-hcd xhci-hcd.0.auto: xHCI Host Controller
[    3.310599] xhci-hcd xhci-hcd.0.auto: new USB bus registered,
assigned bus number 2
[    3.318261] xhci-hcd xhci-hcd.0.auto: Host supports USB 3.0 SuperSpeed
[    3.324961] usb usb2: New USB device found, idVendor=1d6b,
idProduct=0003, bcdDevice= 5.10
[    3.333225] usb usb2: New USB device strings: Mfr=3, Product=2,
SerialNumber=1
[    3.340439] usb usb2: Product: xHCI Host Controller
[    3.345308] usb usb2: Manufacturer: Linux 5.10.0-xilinx-v2020.2 xhci-
hcd
[    3.352001] usb usb2: SerialNumber: xhci-hcd.0.auto
[    3.357060] hub 2-0:1.0: USB hub found
[    3.360945] hub 2-0:1.0: config failed, hub doesn't have any ports!
(err -19)
[    3.368836] rtc_zynqmp f12a0000.rtc: registered as rtc0
[    3.374070] rtc_zynqmp f12a0000.rtc: setting system clock to
2021-06-04T00:38:57 UTC (1622767137)
[    3.383293] cdns-i2c ff030000.i2c: 400 kHz mmio ff030000 irq 27
[    3.389557] cpu cpu0: dev_pm_opp_set_rate: failed to find current OPP
for freq 1349999987 (-34)
[    3.398309] cpufreq: cpufreq_online: CPU0: Running at unlisted
initial frequency: 1349999 KHz, changing to: 1199999 KHz
[    3.409099] cpu cpu0: dev_pm_opp_set_rate: failed to find current OPP
for freq 1349999987 (-34)
[    3.420019] of_cfs_init
[    3.422493] of_cfs_init: OK
[    3.425353] cfg80211: Loading compiled-in X.509 certificates for
regulatory database
[    3.448251] mmc0: SDHCI controller on f1050000.sdhci [f1050000.sdhci]
using ADMA 64-bit
[    3.501399] cfg80211: Loaded X.509 cert 'sforshee: 00b28ddf47aef9cea7'
[    3.507931] clk: Not disabling unused clocks
[    3.512756] ALSA device list:
```

```
[     3.515714]   No soundcards found.
[     3.519394] platform regulatory.0: Direct firmware load for
regulatory.db failed with error -2
[     3.528010] cfg80211: failed to load regulatory.db
[     3.533372] Freeing unused kernel memory: 2048K
[     3.552298] Run /init as init process
udhcpc: started, v1.32.0
[     3.603160] macb ff0c0000.ethernet eth0: Could not attach PHY (-19)
ip: SIOCSIFFLAGS: No such device
[     3.622207] mmc0: new high speed SDHC card at address aaaa
[     3.628096] mmcblk0: mmc0:aaaa SC32G 29.7 GiB
udhcpc: sending discover
[     3.637220]  mmcblk0: p1
udhcpc: sendto: Network is down
udhcpc: read error: Network is down, reopening socket
[     4.428244] usb 1-1: new high-speed USB device number 2 using xhci-hcd
[     4.580526] usb 1-1: New USB device found, idVendor=0781,
idProduct=5567, bcdDevice= 1.00
[     4.588698] usb 1-1: New USB device strings: Mfr=1, Product=2,
SerialNumber=3
[     4.595825] usb 1-1: Product: Cruzer Blade
[     4.599913] usb 1-1: Manufacturer: SanDisk
[     4.604002] usb 1-1: SerialNumber: 4C530000320522104494
[     4.609708] usb-storage 1-1:1.0: USB Mass Storage device detected
[     4.615994] scsi host0: usb-storage 1-1:1.0
[     5.648939] scsi 0:0:0:0: Direct-Access     SanDisk  Cruzer Blade
1.00 PQ: 0 ANSI: 6
[     5.659473] sd 0:0:0:0: [sda] 30629376 512-byte logical blocks: (15.7
GB/14.6 GiB)
[     5.667704] sd 0:0:0:0: [sda] Write Protect is off
[     5.672704] sd 0:0:0:0: [sda] Write cache: disabled, read cache:
enabled, doesn't support DPO or FUA
[     5.715642]  sda:
[     5.720816] sd 0:0:0:0: [sda] Attached SCSI removable disk
udhcpc: sending discover
udhcpc: sendto: Network is down
udhcpc: read error: Network is down, reopening socket
udhcpc: sending discover
udhcpc: sendto: Network is down
udhcpc: read error: Network is down, reopening socket
udhcpc: no lease, failing
ERROR: There's no '/dev' on rootfs.

sh: can't access tty; job control turned off
/ #
```

By default, network settings for PetaLinux reference designs are configured using DHCP. The output you see may be slightly different from the above example, depending on the PetaLinux reference design being tested.

9. Determine the IP address of the PetaLinux system by running ifconfig on the system console.

Additional Options for Booting with JTAG

- To download a bitstream to target board:

```
$ petalinux-boot --jtag --fpga --bitstream <BITSTREAM> --hw_server-url
<hostname:3121>
```

- To download newly built `<plnx-proj-root>/images/linux/u-boot.elf` to target board:

```
$ petalinux-boot --jtag --u-boot --hw_server-url <hostname:3121>
```

- To download newly built kernel to target board:

```
$ petalinux-boot --jtag --kernel --hw_server-url <hostname:3121>
```

  - For MicroBlaze™ processors, this boots `<plnx-proj-root>/images/linux/system.bit`, `u-boot.elf`, `linux.bin.ub`, `system.dtb`, and `rootfs.cpio.gz.u-boot` on target board.

    Note: If using a MicroBlaze processor, you need to add `--fpga` to the `petalinux-boot` command as shown in the following example:

```
petalinux-boot --jtag --fpga --kernel --hw_server-url <hostname:3121>
```

  - For Zynq® UltraScale+™ MPSoC, this boots `<plnx-proj-root>/images/linux/pmufw.elf`, `zynqmp_fsbl.elf`, `u-boot.elf`, `Image`, `system.dtb`, and `ramdisk.cpio.gz.u-boot` on target board.
  - For Zynq-7000 devices, this boots `<plnx-proj-root>/images/linux/zynq_fsbl.elf`, `u-boot.elf`, `uImage`, `system.dtb`, and `rootfs.cpio.gz.u-boot` on target board.
  - For Versal™ ACAP, this boots `<plnx-proj-root>/images/linux/BOOT.BIN`, `<plnx-proj-root>/images/linux/Image`, `<plnx-proj-root>/images/linux/ramdisk.cpio.gz.u-boot` , and `<plnx-proj-root>/images/linux/boot.scr` on target board.

- To download a image with a bitstream with `--fpga --bitstream <BITSTREAM>` option:

```
$ petalinux-boot --jtag --u-boot --fpga --bitstream <BITSTREAM>
```

  The above command downloads the bitstream and then download the U-Boot image.

- To see the verbose output of JTAG boot with `-v` option:

```
$ petalinux-boot --jtag --u-boot -v
```

- To download a customized U-Boot image with the `--u-boot`/`--uboot` option:

```
$ petalinux-boot --jtag --u-boot/--uboot <specify custom u-boot.elf path>
```

- To download a customized kernel image with `--kernel`:
  - **For Zynq UltraScale+ MPSoC and Versal ACAP, use Image:**

```
$ petalinux-boot --jtag --kernel <specify custom Image path>
```

  - **For Zynq-7000 devices, use uImage:**

```
$ petalinux-boot --jtag --kernel <specify custom uImage path>
```

- **For MicroBlaze processors, use `linux.bin.ub`:**

```
$ petalinux-boot --jtag --kernel <specify custom linux.bin.ub path>
```

- To download a customized dtb image with `--kernel`:

```
$ petalinux-boot --jtag --kernel <specify custom kernel path> --dtb
<specify custom dtb path>
```

- To download a customized dtb image with `--uboot/--u-boot`:

```
$ petalinux-boot --jtag --u-boot/--uboot <specify custom u-boot path> --
dtb <specify custom dtb path>
```

- To download a customized pmufw image with kernel:

```
$ petalinux-boot --jtag --kernel <specify custom kernel path> --pmufw
<specify custom pmufw.elf path>
```

Logging Tcl/XSDB for JTAG Boot

Use the following command to take log of XSDB commands used during JTAG boot. It dumps Tcl script (which in turn invokes the XSDB commands) data to `test.txt`.

```
$ cd <plnx-proj-root>
$ petalinux-boot --jtag --prebuilt 3 --tcl test.txt
```

Troubleshooting

This section describes some common issues you may experience while booting a PetaLinux image on hardware with JTAG.

Table 10: **PetaLinux Image on Hardware with JTAG Troubleshooting**

Problem / Error Message	Description and Solution
Cannot see any console output when trying to boot U-Boot or kernel on hardware but boots correctly on QEMU.	**Problem Description:** This problem is usually caused by one or more of the following: • The serial communication terminal application is set with the wrong baud rate. • Mismatch between hardware and software platforms. **Solution:** • Ensure your terminal application baud rate is correct and matches your hardware configuration. • Ensure the PetaLinux project is built with the right hardware platform. ◦ Import hardware configuration properly (see the Importing Hardware Configuration). ◦ Check the "Subsystem AUTO Hardware Settings →" submenu to ensure that it matches the hardware platform. ◦ Check the "Serial settings →" submenu under "Subsystem AUTO Hardware Settings →" to ensure stdout, stdin are set to the correct UART IP core. ◦ Rebuild system images (see Build System Image).

Boot a PetaLinux Image on Hardware with TFTP

This section describes how to boot a PetaLinux image using Trivial File Transfer Protocol (TFTP).

TFTP boot saves a lot of time because it is much faster than booting through JTAG and you do not have to flash the image for every change in kernel source.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- Host environment with TFTP server is setup and PetaLinux Image is built for TFTP boot. For more information, see Configure TFTP Boot.
- You have packaged prebuilt images. For more information, see Packaging Prebuilt Images.
- A serial communication program such as minicom/kermit/gtkterm has been installed; the baud rate of the serial communication program has been set to 115200 bps.
- Ethernet connection is setup properly between Host and Linux Target.
- Appropriate JTAG cable drivers have been installed.

Steps to Boot a PetaLinux Image on Hardware with TFTP

1. Power off the board.
2. Connect the JTAG port on the board to the workstation using a JTAG cable.
3. Connect the serial port on the board to your workstation.
4. Connect the Ethernet port on the board to the local network via a network switch.
5. For Zynq®-7000 devices and Zynq UltraScale+ MPSoC and Versal™ device boards, ensure that the mode switches are set to JTAG mode. Refer to the board documentation for details.
6. Power on the board.
7. Open a console on your workstation and start with preferred serial communication program (for example, kermit, minicom) with the baud rate set to 115200 on that console.
8. Run the `petalinux-boot` command as follows on your workstation

   ```
   $ petalinux-boot --jtag --prebuilt 2 --hw_server-url <hostname:3121>
   ```

   The `--jtag` option tells `petalinux-boot` to boot on hardware via JTAG, and the `--prebuilt 2` option downloads the prebuilt bitstream (FSBL for Zynq UltraScale+ MPSoCs and Zynq-7000 devices, and PLM and PSM for Versal ACAP) to target board, and then boot prebuilt U-Boot on target board.
9. When autoboot starts, hit any key to stop it. The example of a workstation console output for successful U-Boot download is:
10. Check whether the `TFTP` server IP address is set to the IP Address of the host where the image resides. This can be done using the following command:

    ```
    ZynqMP> print serverip
    ```
11. Set the server IP address to the host IP address using the following command:

    ```
    ZynqMP> setenv serverip <HOST IP ADDRESS>
    ```
12. Set the board IP address using the following command:

    ```
    ZynqMP> setenv ipaddr <BOARD IP ADDRESS>
    ```
13. Get the pxe boot file using the following command:

    ```
    ZynqMP> pxe get
    ```
14. Boot the kernel using the following command:

    ```
    ZynqMP> pxe boot
    ```

Troubleshooting

Table 11: **PetaLinux Image on Hardware with TFTP**

Problem / Error Message	Description and Solution
`Error: "serverip" not defined.`	**Problem Description:** This error message indicates that `U-Boot` environment variable `serverip` is not set. You have to set it to IP Address of the host where the image resides. **Solution:** Use the following command to set the `serverip`: `ZynqMP> setenv serverip <HOST IP ADDRESS>;saveenv`

BSP Packaging

BSPs are useful for distribution between teams and customers. Customized PetaLinux project can be shipped to next level teams or external customers through BSPs. This section explains, with an example, how to package a BSP with PetaLinux project.

Prerequisites

This section assumes that you have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. For more information, see Importing Hardware Configuration.

Steps for BSP Packaging

Steps on how to package a project are as follows:

1. You can go outside the PetaLinux project directory to run `petalinux-package` command.
2. Use the following commands to package the BSP.

   ```
   $ petalinux-package --bsp -p <plnx-proj-root> --output MY.BSP
   ```

   This generates `MY.BSP`, including the following elements from the specified project:

   - `<plnx-proj-root>/project-spec/`
   - `<plnx-proj-root>/config.project`
   - `<plnx-proj-root>/.petalinux/`
   - `<plnx-proj-root>/pre-built/`
   - `<plnx-proj-root>/.gitignore`

- `<plnx-proj-root>/components`

Additional BSP Packaging Options

1. BSP packaging with hardware source.

   ```
   $ petalinux-package --bsp -p <plnx-proj-root> --hwsource <hw-project-root> --output MY.BSP
   ```

   It does not modify the specified PetaLinux project `<plnx-proj-root>`. It puts the specified hardware project source to `<plnx-proj-root>/hardware/` inside `MY.BSP` archive.

2. Exclude workspace changes

   The default `petalinux-package --bsp` command checks for sources in `components/plnx-workspace/sources` directory and applies those changes to the meta-user layer. To skip this, use `--exclude-workspace` as shown in the following code snippet:

   ```
   $ petalinux-packge --bsp -p <plnx-proj-root> --exclude-workspace
   ```

   Alternatively, you can clean the project before executing the `petalinux-package --bsp` command as shown below.

   ```
   $ petalinux-build -x mrproper -f
   ```

   This removes the sources and appends directories from `components/yocto/workspace/`.

3. BSP packaging with external sources.

The support for search path is obsolete. It is your responsibility to copy the external sources under `<plnx-proj-root>/components/ext_sources`. For more information, see Using External Kernel and U-Boot with PetaLinux.

Upgrading the Workspace

To upgrade the workspace, use the `petalinux-upgrade` command. You can upgrade the tool in the following three cases. For `petalinux-upgrade` options, see petalinux-upgrade Options.

Upgrading Between Minor Releases (2021.1 Tool with 2021.X Tool)

PetaLinux tool has system software components (embedded software, ATF, Linux, U-Boot, OpenAMP, and Yocto framework) and host tool components (Vivado® Design Suite and Vitis™ software development platform). To upgrade to the latest system software components only, you need to install the corresponding host tools.

The `petalinux-upgrade` command resolves this issue by upgrading the system software components without changing the host tool components. The system software components are upgraded in two steps: first, by upgrading the installed PetaLinux tool, and then by upgrading existing PetaLinux projects. This allows you to upgrade without having to install the latest version of the Vivado hardware project or Vitis software platform.

Upgrade PetaLinux Tool

Upgrade from Local File

Download the target system software components content from the server URL http://petalinux.xilinx.com/sswreleases/rel-v2021/sdkupdate.

`petalinux-upgrade` command would expect the downloaded path as input.

1. Install the tool if you do not have it installed.

   Note: Ensure the install area is writable.

2. Change into the directory of your installed PetaLinux tool using `cd <plnx-tool>`.
3. Type: `source settings.sh`.
4. Enter command: `petalinux-upgrade -f <downloaded sdkupdate path>`.

Example:

```
petalinux-upgrade -f “/scratch/ws/upgrade-workspace/sdkupdate”
```

Upgrade from Remote Server

Follow these steps to upgrade the installed tool target system software components from the remote server.

1. Install the tool if you do not have it installed.

   ***Note*:** The tool should have R/W permissions.

2. Go to installed tool.
3. Type: `source settings.sh`.
4. Enter command: `petalinux-upgrade -u <url>`.

Example:

```
petalinux-upgrade -u "http://petalinux.xilinx.com/sswreleases/rel-v2021/
sdkupdate/"
```

IMPORTANT! *Only minor version upgrades are supported.*

Upgrading only Preferred Platforms in Tool

- **To upgrade all platforms:**

  ```
  $ petalinux-upgrade -u/-f <path/url>
  ```

  To upgrade the eSDKs for all (Zynq devices, Zynq UltraScale+ MPSoC, Versal, microblaze_lite, microblaze_full).

- **To upgrade only Zynq-7000 platform:**

  ```
  $ petalinux-upgrade -u/-f <path/url> --platform "arm"
  ```

- **To upgrade eSDKs for Zynq, Zynq UltraScale+ MPSoC, and Versal platforms:**

  ```
  $ petalinux-upgrade -u/-f <path/url> --platform "arm aarch64"
  ```

- **To upgrade eSDKs for microblaze_lite:**

  ```
  $ petalinux-upgrade -u/-f <path/url> --platform "microblaze_lite
  microblaze_full"
  ```

Upgrade PetaLinux Project

Upgrade an Existing Project with the Upgraded Tool

Use the following steps to upgrade existing project with upgraded tool.

1. Run `petalinux-build -x mrproper` in the existing project before upgrading the tool.
2. Upgrade the tool. To upgrade from local file, see Upgrade from Local File. To upgrade from remote server, see Upgrade from Remote Server.
3. Go to the PetaLinux project you want to upgrade.
4. Enter either `petalinux-build` or `petalinux-config` to upgrade the project with all new system components.
5. When asked to upgrade the eSDK, please select **y** to extract the new eSDK as shown below.

   ```
   WARNING: Your Yocto SDK was changed in tool.
   Please input "y" to proceed the installing SDK into project, "n" to
   exit:y
   ```

   Now your project is built with the upgraded tool.
6. If you had used only the `petalinux-config` command in step 4, run the `petalinux-build` command to build the project.

Upgrading the Installed Tool with More Platforms

Initially, you installed PetaLinux tool with only the arm platform as specified in Installing the PetaLinux Tool. To install the aarch64 platform, follow these steps.

For the following use case, if you want to upgrade with 2020.1 platforms, use http://petalinux.xilinx.com/sswreleases/rel-v2021/sdkupdate.

1. Go to the installed tool.
2. Source `settings.sh` file.
3. Run: `petalinux-upgrade -u http://petalinux.xilinx.com/sswreleases/rel-v2021/sdkupdate/ -p aarch64`

The new platform is part of your `<plnx-tool>/components/yocto/source/aarch64`.

Use Cases

- To get the Zynq platform only:

```
$ petalinux-upgrade -u/-f <path/url> --platform "arm"
```

- To get Zynq, Zynq UltraScale+ MPSoC, and Versal platforms:

```
$ petalinux-upgrade -u/-f <path/url> --platform "arm aarch64"
```

- To get the MicroBlaze platforms:

```
$ petalinux-upgrade -u/-f <path/url> --platform "microblaze_lite
microblaze_full"
```

Upgrading the Installed Tool with your Customized Platform

From 2020.1 release onwards, `platform/esdk` is part of your project `<plnx-proj-root>/components/yocto`. You can make changes in the `esdk/platform` and you can build those changes using the `petalinux-build -esdk` option. The newly built eSDK is in `<plnx-proj-root>/images/linux/esdk.sh`. Rename the newly built `esdk.sh` as `aarch64/arm/mb-lite/mb-full` based on your project.

1. Go to the installed tool.
2. Source `settings.sh`.
3. Run `petalinux-upgrade -f <plnx-proj-root>/images/linux/ -p <platform>`.

   The tool will be upgraded with your new platform changes.

Note: These procedures work only between minor releases.

Customizing the Project

Firmware Version Configuration

This section explains how to do firmware version configuration using `petalinux-config` command.

Prerequisites

This section assumes that you have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. For more information, see the Importing Hardware Configuration.

Steps for Firmware Version Configuration

1. Change into the root directory of your PetaLinux project.

   ```
   $ cd <plnx-proj-root>
   ```

2. Launch the top level system configuration menu.

   ```
   $ petalinux-config
   ```

3. Select **Firmware Version Configuration**.
4. Select Host Name, Product Name, Firmware Version as per the requirement to edit them.
5. Exit the menu and select `<Yes>` when asked: Do you wish to save your new configuration?
6. Once the target is booted, verify the host name in `cat /etc/hostname`, product name in `cat /etc/petalinux/product`, and the firmware version in `cat /etc/petalinux/version`.

Root File System Type Configuration

This section details configuration of `RootFS` type using `petalinux-config` command.

Prerequisites

This section assumes that you have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. For more information, see the Importing Hardware Configuration.

Steps for Root File System Type Configuration

1. Change into the root directory of your PetaLinux project.

```
$ cd <plnx-proj-root>
```

2. Launch the top level system configuration menu.

```
$ petalinux-config
```

3. Select **Image Packaging Configuration → Root File System Type**.
4. Select **INITRAMFS** or **INITRD** or **JFFS2** or **UBI/UBIFS** or **NFS** or **EXT4 (SD/eMMC/SATA/USB)** as per the requirement.

   ***Note*:** EXT4 boot functionality expects the root file system to be mounted on ext4 partition and all other boot images in FAT32 partition.

   ***Note*:** INITRAMFS or INITRD configuration with `petalinux-initramfs-image` as INITRAMFS/INITRD Image name expects the root file system to be mounted on ext4 partition to use the `switch_root`.
5. Save Configuration settings.

Boot Images Storage Configuration

This section provides details about configuration of the Boot Device, for example, Flash and SD/MMC using `petalinux-config` command.

Prerequisites

This section assumes that you have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. For more information, see the Importing Hardware Configuration.

Modifying Boot Script Configuration

1. Change into the root directory of your PetaLinux project.

```
$ cd <plnx-proj-root>
```

2. Launch the top level system configuration menu.

```
$ petalinux-config
```

3. Select **u-boot Configuration → u-boot script configuration**.
4. In the u-boot script configuration submenu, you have the following options:

```
[*] Append base address to image offsets
()  Pre bootenv
    JTAG/DDR image offsets  --->
    QSPI/OSPI image offsets  --->
    NAND image offsets  --->
(Image) Kernel image name
(image.ub) Fit image name
```

- To append provided offsets to the memory/DDR base address, select **[*] Append base address to image offsets**. PetaLinux reads the DDR base address from the hardware file and appends it to the offsets provided. This is enabled by default.
- To add any U-Boot environment variables which have to be executed before the boot command, select **() Pre bootenv**.
- To check/modify the JTAG/DDR offsets, use **JTAG/DDR image offsets -->**. These are used in the JTAG boot mode and also to load the images from flash/storage to the DDR memory.

  Note: By default, the following load addresses are used by PetaLinux for the `boot.scr` on the JTAG boot:

  Table 12: **Default Load Address on JTAG Boot**

Device	Load Address of boot.scr
Zynq UltraScale+ MPSoC and Versal ACAP	DDR base address + 0x20000000
Zynq-7000 devices	DDR base address + 0x3000000
MicroBlaze processors	(DDR base address + DDR size) - 0xe00000

```
(0x1000) Devicetree offset
(0x200000) Kernel offset
(0x4000000) Ramdisk image offset
(0x10000000) Fit image offset
```

- To check/modify the QSPI/OSPI offsets/sizes, use **QSPI/OSPI image offsets --->** configuration menu. Make sure the specified offsets/sizes do not overlap with `boot.scr` or image to image.

  Note: By default, the following load addresses are used by PetaLinux for the `boot.scr` on the QSPI/OSPI boot:

Table 13: **Default Load Address on QSPI/OSPI Boot**

Device	Load Address of boot.scr
Zynq UltraScale+ MPSoC	0x3E80000
Versal ACAP	0x7F80000
Zynq-7000 devices	0xFC0000
MicroBlaze processors	0x1F00000

```
(0xF00000) QSPI/OSPI Kernel offset
(0x1D00000) QSPI/OSPI Kernel size
(0x2E00000) QSPI/OSPI Ramdisk offset
(0x4000000) QSPI/OSPI Ramdisk size
(0xF40000) QSPI/OSPI fit image offset
(0x6400000) QSPI/OSPI fit image size
```

- To check/modify the NAND offsets/sizes, use the **NAND image offsets -->** configuration menu. Make sure the specified offsets/sizes do not overlap with the `boot.scr` or image to image.

  Note: By default, the following load addresses are used by PetaLinux for the `boot.scr` on the NAND boot:

Table 14: **Default Load Address on NAND Boot**

Device	Load Address of boot.scr
Zynq UltraScale+ MPSoC	0x3E80000
Versal ACAP	0x7F80000
Zynq-7000 devices	0xFC0000
MicroBlaze processors	0x1F00000

```
(0x4100000) NAND Kernel offset
(0x3200000) NAND Kernel size
(0x8200000) NAND Ramdisk offset
(0x3200000) NAND Ramdisk size
(0x4180000) NAND fit image offset
(0x6400000) NAND fit image size
```

Troubleshooting

This section describes some common issues you may experience while working with boot device configuration.

Table 15: **Boot Images Storage Troubleshooting**

Problem / Error Message	Description and Solution
`ERROR: Failed to config linux/kernel!`	**Problem Description:** This error message indicates that it is unable to configure the linux-kernel component with menuconfig. **Solution:** Check whether all required libraries/packages are installed properly. For more information, see the Installation Requirements.

Primary Flash Partition Configuration

This sections provides details on how to configure flash partition with PetaLinux menuconfig.

1. Change into the root directory of your PetaLinux project.

   ```
   $ cd <plnx-proj-root>
   ```

2. Launch the top level system configuration menu.

   ```
   $ petalinux-config
   ```

3. Select **Subsystem AUTO Hardware Settings → Flash Settings**.
4. Select a flash device as the Primary Flash.
5. Set the name and the size of each partition.

Managing Image Size

In an embedded environment, it is important to reduce the size of the kernel image stored in flash and the static size of kernel image in RAM. This section describes impact of `config` item on kernel size and RAM usage.

By default, the FIT image is composed of kernel image, DTB, and Tiny RootFS image.

Prerequisites

This section assumes that you have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. For more information, see the Importing Hardware Configuration.

Steps for Managing Image Size

The FIT Image size can be reduced using the following methods:

1. Launch the root file system configuration menu using the following command:

```
$ cd <plnx-proj-root>
$ petalinux-config -c rootfs
```

2. Select **File System Packages**.

   Under this submenu, you can find the list of options corresponding to the root file system packages. If your requirement does not need some of these packages, you can shrink the size of the root file system image by disabling them.

3. Launch the kernel configuration menu using the following command:

```
$ cd <plnx-proj-root>
$ petalinux-config -c kernel
```

4. Select **General Setup**.

   Under this sub-menu, you can find options to set the `config` items. Any item that is not mandatory to have in the system can be disabled to reduce the kernel image size. For example, `CONFIG_SHMEM`, `CONFIG_AIO`, `CONFIG_SWAP`, `CONFIG_SYSVIPC`. For more details, see the Linux kernel documentation.

   Note: Note that disabling of some `config` items may lead to unsuccessful boot. It is expected that you have the knowledge of `config` items before disabling them.

   Including extra configuration items and file system packages lead to increase in the kernel image size and the root file system size respectively.

   If the kernel or the root file system size increases and is greater than 128 MB, make the following changes in `bsp.cfg`:

   Any U-Boot configuration and environment variables that are added to `bsp.cfg` are included in the U-Boot build.

5. Mention the Bootm length in `<plnx-proj-root>/project-spec/meta-user/recipes-bsp/u-boot/files/bsp.cfg`.

```
#define CONFIG_SYS_BOOTM_LEN <value greater than image size>
```

6. Undef `CONFIG_SYS_BOOTMAPSZ` in `<plnx-proj-root>/project-spec/meta-user/recipes-bsp/u-boot/files/bsp.cfg`.

Configuring INITRD BOOT

Initial RAM disk (INITRD) provides the capability to load a RAM disk by the boot loader during the PetaLinux startup process. The Linux kernel mounts it as RootFS and starts the initialization process. This section describes the procedure to configure the INITRD boot.

Prerequisites

This section assumes that you have created a new PetaLinux project (see Creating an Empty Project from a Template) and imported the hardware platform (see Importing Hardware Configuration).

Steps to Configure INITRD Boot

1. Set the RootFS type to INITRD. For more information, see Root File System Type Configuration.
2. Set RAMDISK loadaddr. Ensure loadaddr does not overlap with kernel or DTB address and that it is a valid DDR address.
3. Set INITRAMFS/INITRD Image name. The default image name is set to `petalinux-intramfs-image` to enable the `switch_root`. The build system generates two types of root file systems: ramdisk images and rootfs images. The ramdisk image is as specified in INITRAMFS/INITRD Image name. It is packed into Fit image (`image.ub`).

   Note: Setting petalinux-initramfs-image enables the `switch_root` and searches for the rootfs in ext2/3/4 of SD partitions.
4. Build the system image. For more information, see Build System Image.
5. Use one of the following methods to boot the system image:
   a. Boot a PetaLinux Image on Hardware with SD Card, see Boot a PetaLinux Image on Hardware with an SD Card.
   b. Boot a PetaLinux Image on Hardware with JTAG, see Boot a PetaLinux Image on Hardware with JTAG.
      - Make sure you have configured TFTP server in host.
      - Set the server IP address to the host IP address using the following command at U-Boot prompt:

        ```
        ZynqMP> setenv serverip <HOST IP ADDRESS>
        ```

      - Read the images using following command:

        ```
        ZynqMP> tftpb <dtb load address> system.dtb;tftpb <kernel load address> Image; tftpb <rootfs load address> rootfs.cpio.gz.u-boot.
        ```

- Boot images using following command:

```
ZynqMP> booti <kernel load address> <rootfs loadaddress> <device
tree load address>
```

Configuring INITRAMFS Boot

Initial RAM file system (INITRAMFS) is the successor of INITRD. It is a `cpio` archive of the initial file system that gets loaded into memory during the PetaLinux startup process. The Linux kernel mounts it as RootFS and starts the initialization process.

This section describes the procedure to configure INITRAMFS boot.

Prerequisites

This section assumes that you have created a new PetaLinux project (see Creating an Empty Project from a Template) and imported the hardware platform (see Importing Hardware Configuration).

Steps to Configure INITRAMFS Boot

1. Set the RootFS type to `INITRAMFS`. For more information, see Root File System Type Configuration.
2. Set INITRAMFS/INITRD Image name. The default image name is set to `petalinux-intramfs-image` to enable the `switch_root`. The build system generates two types of root file systems: ramdisk images and rootfs images. The ramdisk image is generated as specified in the INITRAMFS/INITRD Image name. It is packed into Fit image (`image.ub`) and Kernel image.

   ***Note*:** Setting petalinux-initramfs-image enables the `switch_root` and searches for the rootfs in ext2/3/4 of SD partitions.
3. Build the system image. For more information, see Build System Image.
4. Use one of the following methods to boot the system image.
   a. Boot a PetaLinux Image on QEMU, see Booting a PetaLinux Image on QEMU.
   b. Boot a PetaLinux Image on Hardware with SD Card, see Boot a PetaLinux Image on Hardware with an SD Card.
   c. Boot a PetaLinux Image on Hardware with JTAG, see Boot a PetaLinux Image on Hardware with JTAG.

IMPORTANT! *The default mode in the PetaLinux BSP is the INITRD mode.*

In INITRAMFS mode, RootFS is included in the kernel image.

- Image → Image (kernel) + `ramdisk.cpio/rootfs.cpio` (for Zynq® UltraScale+™ MPSoC and Versal™ ACAP)
- zImage → zImage (kernel) + `ramdisk.cpio/rootfs.cpio` (for Zynq-7000 devices)
- `linux.bin.ub` → `simpleImage.mb` (kernel) + `ramdisk.cpio/rootfs.cpio` (for MicroBlaze™ processors)

As you select the RootFS components, its size increases proportionally.

Configure TFTP Boot

This section describes how to configure the host and the PetaLinux image for the TFTP boot.

TFTP boot saves a lot of time because it is much faster than booting through JTAG and you do not have to flash the image for every change in kernel source.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have created a new PetaLinux project (see Creating an Empty Project from a Template) and imported the hardware platform (see Importing Hardware Configuration).
- You have TFTP server running on your host.

PetaLinux Configuration and Build System Image

Steps to configure PetaLinux for TFTP boot and build the system image are:

1. Change to root directory of your PetaLinux project.

   ```
   $ cd <plnx-proj-root>
   ```

2. Launch the top level system configuration menu.

   ```
   $ petalinux-config
   ```

3. Select **Image Packaging Configuration**.
4. Select **Copy final images to tftpboot** and set tftpboot directory. By default, the TFTP directory ID is /tftpboot. Ensure this matches your host's TFTP server setup.
5. Save configuration settings and build system image as explained in Build System Image.

Configuring NFS Boot

One of the most important components of a Linux system is the root file system. A well-developed root file system can provide you with useful tools to work on PetaLinux projects. Because a root file system can become big in size, it is hard to store it in flash memory.

The most convenient thing is to mount the entire root file system from the network allowing the host system and the target to share the same files. The root file system can be modified quickly and also on the fly (meaning that the file system can be modified while the system is running). The most common way to setup a system like the one described is through NFS.

In case of NFS, no manual refresh is needed for new files.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have created a new PetaLinux project (see Creating an Empty Project from a Template) and imported the hardware platform (see Importing Hardware Configuration).
- You have Linux file and directory permissions.
- You have an NFS server setup on your host. Assuming it is set up as `/home/NFSshare` in this example.

PetaLinux Configuration and Build System Image

Steps to configure the PetaLinux for NFS boot and build the system image are as follows:

1. Change to root directory of your PetaLinux project.

   ```
   $ cd <plnx-proj-root>
   ```

2. Launch the top level system configuration menu.

   ```
   $ petalinux-config
   ```

3. Select **Image Packaging Configuration → Root File System Type**.
4. Select **NFS** as the RootFS type.
5. Select **Location of NFS root directory** and set it to `/home/NFSshare`.
6. Exit menuconfig and save configuration settings. The boot arguments in the auto generated DTSI is automatically updated after the build. You can check `<plnx-proj-root>/components/plnx_workspace/device-tree/device-tree/system-conf.dts`.
7. Launch Kernel configuration menu.

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

8. Select **Networking support → IP: kernel level configuration**.
   - IP:DHCP support
   - IP:BOOTP support
   - IP:RARP support
9. Select **File systems → Network file systems → Root file systems** on NFS.
10. Build the system image.

    Note: For more information, see Build System Image.

11. You can see the updated boot arguments only after building.

Booting with NFS

In case of NFS Boot, RootFS is mounted through the NFS but bootloader (FSBL, bitstream, U-Boot), and kernel can be downloaded using various methods as mentioned below.

1. JTAG: In this case, bootloader and kernel is downloaded on to the target through JTAG. For more information, see Boot a PetaLinux Image on Hardware with JTAG.

> **TIP:** *If you want to make use of prebuilt capability to boot with JTAG, package images into prebuilt directory. For more information, see Packaging Prebuilt Images.*

1. tftpboot: In this case, bootloader is downloaded through JTAG and kernel is downloaded on to the target through tftpboot. For more information, see Boot a PetaLinux Image on Hardware with TFTP.
2. SD card: In this case, bootloader (`BOOT.BIN`), bootscript (boot.scr) and kernel image (`image.ub`) is copied to the SD card downloaded from the SD card. For more information, see Boot a PetaLinux Image on Hardware with an SD Card.

Configuring JFFS2 Boot

Journaling flash file system version 2 or JFFS2 is a log-structured file system for use with flash memory devices. This section describes the procedure to configure JFFS2 boot.

Prerequisites

This section assumes that you have created a new PetaLinux project (see Creating an Empty Project from a Template) and imported the hardware platform (see Importing Hardware Configuration).

Steps to Configure JFFS2 Boot

1. Set the root file system type to JFFS2. For more information, see Root File System Type Configuration.
2. Select petalinux-config -->Image Packaging Configuration ---> jffs2 erase block size(KByte) (jffs2 erase block size: 128KiB)

   Note: 8KiB erase block size does not work with 5.10 kernel because they disabled the dependent kernel configs to support UBIFS boot support.
3. Set Primary Flash as boot device and boot images storage. For more information, see Boot Images Storage Configuration and Primary Flash Partition Configuration.
4. Build the system image. For more information, see Build System Image.
5. Boot a PetaLinux Image on hardware with JTAG, see Boot a PetaLinux Image on Hardware with an SD Card.
6. Make sure you have configured TFTP server in host.
7. Set the server IP address to the host IP address using the following command at U-Boot prompt.

   ```
   ZynqMP> setenv serverip <HOST IP ADDRESS>;
   ```

   a. Detect Flash Memory.

   ```
   ZynqMP> sf probe 0 0 0
   ```

   b. Erase Flash Memory.

   ```
   ZynqMP> sf erase 0 0x8000000
   ```

   c. Read images onto Memory and write into Flash.

   - Read `BOOT.BIN`.

     ```
     ZynqMP> tftpboot 0x80000 BOOT.BIN
     ```

   - Write `BOOT.BIN`.

     ```
     ZynqMP> sf write 0x80000 0x0 <size of boot.bin>
     ```

     Example: `sf write 0x80000 0x0 0x10EF48`

   - Read `image.ub`.

     ```
     ZynqMP> tftpboot 0x80000 image.ub
     ```

   - Write `image.ub`.

     ```
     ZynqMP>sf write 0x80000 <loading address of kernel> <size of
     image.ub>
     ```

     Example: `sf write 0x80000 0xF40000 0x6cb0e4`

- Read `rootfs.jffs2`.

```
ZynqMP> tftpboot 0x80000 rootfs.jffs2
```

- Write `rootfs.jffs2`.

```
ZynqMP> sf write 0x80000 <loading address of rootfs.jffs2> <size of
rootfs.jffs2>
```

Example: `sf write 0x80000 0x04000000 0x7d4000`

- Read `boot.scr`

```
ZynqMP> tftpboot 0x80000 boot.scr
```

- Write `boot.scr`

```
ZynqMP> sf write 0x80000 <loading address of boot.scr> < size of
boot.scr>
```

Example: sf write 0x80000 0x03e80000 0x80000

Note: Check the offsets for kernel and root file system at **petalinux-config → u-boot Configuration → u-boot script configuration**. If they do not match, the process may fail at the U-Boot prompt.

8. Enable QSPI flash boot mode on board.
9. Reset the board (booting starts from flash).

Table 16: **Error Message while loading Image and ramdisk.cpio.gz.u-boot**

Error Message	Description

`Wrong Image Format for bootm command`
`ERROR: can't get kernel image!`
`Booting using Fit image failed`
	This error message appears when you load Image and ramdisk.cpio.gz.u-boot for QSPI/OSPI/NAND boot mode. You can ignore this error message.

Configuring UBIFS Boot

Unsorted Block Images File System (UBIFS) is a flash file system that is different from traditional Linux file systems (for example, Ext2, XFS, and JFS). It works with MTD devices and not with block devices. The other Linux file system of this class is JFFS2.

Prerequisites

This section assumes that you have created a new PetaLinux project (see Creating an Empty Project from a Template) and imported the hardware platform (see Importing Hardware Configuration).

Steps to Configure UBIFS Boot

1. Set the root file system to `UBIFS`. For more information, see Root File System Type Configuration.

```
        Root filesystem type (UBI/UBIFS)  --->
    ()  mkfs.ubifs args (NEW)
    ()  ubinize args (NEW)
    (ubifs) ubi part name (NEW)
    (image.ub) name for bootable kernel image
    (cpio cpio.gz cpio.gz.u-boot ext4 tar.gz jffs2 ubi) Root filesystem formats
    (0x1000) DTB padding size
    [*] Copy final images to tftpboot
    (/tftpboot) tftpboot directory
```

   .

   a. Select **petalinux-config → Image Packaging Configuration → Root filesystem type (UBI/UBIFS)**.

   b. Specify the arguments used in `mkfs.ubifs` to create the UBIFS partition `-m 2 -e 130944 -c 400` For QSPI 128 erase block size.

   c. Specify the arguments used in `ubisize` to create the UBIFS partition. `-m 2048 -p 128KiB -s 512` For QSPI 128 erase block size.

   d. Specify the ubifs part name to add in bootargs, for example, ubifs.

      Note: Ensure that you create the QSPI partition with the specified name.

   e. Specify the root file system format to build `rootfs.ubi` during the build.

      For example,

```
tar.gz cpio cpio.gz.u-boot cpio.gz ext4 jffs2 ubi
```

      .

2. Set Primary Flash as boot device and boot images storage. For more information, see Boot Images Storage Configuration and Primary Flash Partition Configuration.
3. Build the system image. For more information, see Build System Image.
4. Boot a PetaLinux Image on Hardware with SD Card. For more information, see Boot a PetaLinux Image on Hardware with an SD Card.
5. Ensure that you have configured the TFTP server in the host.
6. Set the server IP address to the host IP address using the following command at U-Boot prompt.

```
ZynqMP> setenv serverip <HOST IP ADDRESS>;
```

a. Detect Flash Memory.

```
ZynqMP> sf probe 0 0 0
```

b. Erase Flash Memory.

```
ZynqMP> sf erase 0 0x8000000
```

c. Read images onto Memory and write into Flash.

- Read `BOOT.BIN`.

  ```
  ZynqMP> tftpboot 0x80000 BOOT.BIN
  ```

- Write `BOOT.BIN`.

  ```
  ZynqMP> sf write 0x80000 0x0 <size of boot.bin>
  ```

  Example: `sf write 0x80000 0x0 0x10EF48`

- Read `image.ub`.

  ```
  ZynqMP> tftpboot 0x80000 image.ub
  ```

- Write `image.ub`.

  ```
  ZynqMP>sf write 0x80000 <loading address of kernel> <size of
  image.ub>
  ```

  Example: `sf write 0x80000 0xF40000 0x6cb0e4`

- Read `rootfs.ubi`.

  ```
  ZynqMP> tftpboot 0x80000 rootfs.ubi
  ```

- Write `rootfs.ubi`.

  ```
  ZynqMP> sf write 0x80000 <loading address of rootfs.ubi> <size of
  rootfs.ubi>
  ```

  Example: `sf write 0x80000 0x04000000 0x7d4000`

- Read `boot.scr`

  ```
  ZynqMP> tftpboot 0x80000 boot.scr
  ```

- Write `boot.scr`

  ```
  ZynqMP> sf write 0x80000 <loading address of boot.scr> < size of
  boot.scr>
  ```

  Example: sf write 0x80000 0x03e80000 0x80000

  Note: Check the offsets for kernel and root file system at **petalinux-config → u-boot Configuration → u-boot script configuration**. If they do not match, the process may fail at the U-Boot prompt.

7. Enable QSPI flash boot mode on board.

8. Reset the board (booting starts from flash).

Calculating mkfs.ubifs and ubinize Arguments

After calculating the `mkfs.ubifs` and `ubinize`, which are used in the creation of the UBI file system, the final values need to be added to the `petalinux-config` options.

Calculating mkfs.ubifs Arguments

Table 17: **mkfs.ubifs Args**

Argument	Description
`-m / --min-io-size`	The minimum I/O unit. Values are static based on the Flash type. • QSPI single the value is 1 • QSPI dual the value is 2 • OSPI the value is 1
`-e / --leb-size`	The logical erase block size of the UBI volume. leb-size/-e = PEB size - EC - VID = 128K - 64bytes - 64 bytes = 130944 bytes
`-c / --max-leb-cnt`	Specifies maximum file-system size in logical erase blocks • PEB(physical erase block) size = 128 Kbytes . Spi datasheet or `sf probe 0 0 0` command from U-boot terminal provide this. • EC header for spi flash = 64 bytes. The link shows the default values for QSPI. max-leb-cnt/-c = QSPI ubifs partition size÷(division) leb-size = 50MB ÷ 130944 bytes = 400 bytes

The final expanded values for

```
mkfs.ubifs  arguments are -m 2 -e 130944 -c 400
```

Calculating ubinize Arguments

mkfs.ubifs Arguments

Table 18: **mkfs.ubifs Args**

Argument	Description
`-m / --min-io-size`	The minimum I/O unit. The following values are static based on the Flash type. • QSPI single the value is 1 • QSPI dual the value is 2 • OSPI the value is 1
`-p / --peb-size`	Tells ubinize that the physical erase block size of the flash chip for which the UBI image is created is 128KiB. -peb-size / -p = 128 Kbytes . The SPI datasheet or `sf probe 0 0 0` command from u-boot terminal provide this value.
`-s / --sub-page-size`	Tells ubinize that the flash supports sub-pages. This should not be more that the min-io-size. --sub-page-size / -s = 1 based on the min-io-size.

```
The final expanded values for ubinize arguments are -m 2 -p 128KiB -s
1
```

***Note*:** The specified `mkfs.ubifs` arguments and `ubinize` arguments are based on QSPI 128KB erase size

Configuring SD Card ext File System Boot

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have created a new PetaLinux project (see Creating an Empty Project from a Template) and imported the hardware platform (see Importing Hardware Configuration).
- An SD memory card with at least 8 GB of storage space. It is recommended to use a card with speed-grade 6 or higher to achieve optimal file transfer performance.

Preparing the SD Card

Steps to prepare the SD card for the PetaLinux SD card ext file system boot as are follows:

For more information on how to format and partition the SD card, see Appendix H: Partitioning and Formatting an SD Card.

The SD card is formatted with two partitions using a partition editor such as gparted. The first partition should be at least 500 MB in size and formatted as a FAT32 file system. Ensure that there is 4 MB of free space preceding the partition. The first partition contains the boot loader, device tree, and kernel images. The second partition should be formatted as an `ext4` files system and can take up the remaining space on the SD card. This partition stores the system root file system.

1. Label the first partition as BOOT.
2. Label the second partition as RootFS.
3. Copy the files as follows:
   - FAT partition:`BOOT.BIN`, `boot.scr`, Image, and `ramdisk.cpio.gz.u-boot` (if `switch_root` is enabled)
   - EXT partition: Extract `rootfs.tar.gz`/`rootfs.cpio.gz`

For optimal performance ensure that the SD card partitions are 4 MB aligned.

PetaLinux Configuration and Build System Image

Steps to configure PetaLinux for SD card ext file system boot and build the system image are as follows:

1. Change to root directory of your PetaLinux project.

   ```
   $ cd <plnx-proj-root>
   ```

2. Launch top level system configuration menu.

   ```
   $ petalinux-config
   ```

3. Select **Image Packaging Configuration → Root file system type**.
4. Select **EXT4 (SD/eMMC/SATA/USB)** as the root file system type.

   ***Note*:** Choose this setting to configure your PetaLinux build for EXT root. By default, it adds the SD/eMMC device name in bootargs. For other devices (SATA/USB), you must change the ext4 device name, as shown in the following examples:

   - eMMC or SD root = `/dev/mmcblkYpX`
   - SATA or USB root= `/dev/sdX`

5. Exit menuconfig and save configuration settings.

   ***Note*:** The boot arguments is automatically updated in the `<plnx-proj-root>/components/plnx_workspace/device-tree/device-tree/system-conf.dtsi`. These changes are reflected only after the build.

6. Build PetaLinux images. For more information, see Build System Image.
7. Generate boot image. For more information, see Generate Boot Image for Zynq UltraScale+ MPSoC.
8. The generated `rootfs.tar.gz` file is present in `images/linux` directory. To extract, use `tar xvf rootfs.tar.gz`.

Copying Image Files

This section explains how to copy image files to SD card partitions. Assuming the two partitions get mounted at `/media/BOOT` and `/media/rootfs`.

1. Change to root directory of your PetaLinux project.

```
$ cd <plnx-proj-root>
```

2. Copy `BOOT.BIN` and `image.ub` to BOOT partition of SD card. The `image.ub` file has device tree and kernel image files.

```
$ cp images/linux/BOOT.BIN /media/BOOT/
$ cp images/linux/image.ub /media/BOOT/
$ cp images/linux/boot.scr /media/BOOT/
```

3. Extract `rootfs.tar.gz` file to the root file system partition of the SD card and extract the file system.

```
$ sudo tar xvf rootfs.tar.gz -C /media/rootfs
```

In order to boot this SD card ext image, see Boot a PetaLinux Image on Hardware with an SD Card. To Create an SD image using PetaLinux use the `petalinux-package --wic` command.

Troubleshooting

Table 19: **Configuring SD Card ext Filesystem Boot**

Problem / Error Message	Description and Solution
`EXT4-fs (mmcblk0p2): mounted filesystem with ordered data mode. Opts: (null) Kernel panic - not syncing: No working init found.`	**Problem Description:** This message indicates that the Linux kernel is unable to mount EXT4 File System and unable to find working init. **Solution:** Extract RootFS in RootFS partition of SD card. For more information, see the Copying Image Files.

Customizing the Root File System

Including Prebuilt Libraries

This section explains how to include pre-compiled libraries to PetaLinux root file system.

If a library is developed outside PetaLinux, you may just want to add the library in the PetaLinux root file system. In this case, an application template is created to allow copying of the existing content to target file system.

If the application, library, or module name has '_', uppercase letters, or starts with an uppercase letter, see Recipe Name has ' _ ' or Uppercase Letters or Starts with an Uppercase Letter.

Prerequisites

This section assumes that you have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. For more information, see Importing Hardware Configuration.

Steps to Include Prebuilt Libraries

If your prebuilt library name is `mylib.so`, including this into PetaLinux root file system is explained in following steps.

1. Ensure that the pre-compiled code has been compiled for your PetaLinux target architecture, for example, MicroBlaze™ processors, Arm® cores, etc.
2. Create an application with the following command.

   ```
   $ petalinux-create -t apps --template install --name mylib --enable
   ```

   Note: `--enable` should be executed after the `petalinux-config` command.
3. Change to the newly created application directory.

   ```
   $ cd <plnx-proj-root>/project-spec/meta-user/recipes-apps/mylib/files/
   ```

4. Remove existing `mylib` file, and copy the prebuilt `mylib.so` into `mylib/files` directory.

```
$ rm mylib
$ cp <path-to-prebuilt-mylib.so> ./
```

5. Create an application and include a prebuilt library into the root file system with a single command instead of following steps 2, 3, and 4. The following command creates `mylib app` and copies `mylib.so` from `<path-to-dir> to mylib/files` directory.

```
$ petalinux-create -t apps --template install --name mylib --srcuri
<path-to-dir>/mylib.so --enable
```

Note: This is applicable for applications and modules.

6. Create an application with multiple source files.

```
$ petalinux-create -t apps --template install --name mylibs --srcuri
"<path-to-dir>/mylib1.so <path-to-dir>/mylib2.so"
```

Note: This is applicable for applications and modules.

7. Create an app with remote sources. The following examples will create applications with specified git/http/https pointing to the srcuri.

```
$ petalinux-create -t apps -n myapp --enable --srcuri http://
example.tar.gz
```

```
$ petalinux-create -t apps -n myapp --enable --srcuri git://example.git
\;protocol=https
```

```
$ petalinux-create -t apps -n myapp --enable --srcuri https://
example.tar.gz
```

Note: This is applicable for applications and modules.

8. Edit `<plnx-proj-root>/project-spec/meta-user/recipes-apps/mylib/mylib.bb`.

   The file should look like the following.

```
# This file is the libs recipe.
#

SUMMARY = "Simple libs application"
SECTION = "PETALINUX/apps"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/
MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

SRC_URI = "file://mylib.so \
 "

S = "${WORKDIR}"

TARGET_CC_ARCH += "${LDFLAGS}"

do_install() {
      install -d ${D}${libdir}
```

```
        install -m 0655 ${S}/mylib.so ${D}${libdir}
}

FILES_${PN} += "${libdir}"
FILES_SOLIBSDEV = ""
```

9. Run `petalinux-build -c rootfs`.

IMPORTANT! *You need to ensure that the binary data being installed into the target file system by an install template application is compatible with the underlying hardware implementation of your system.*

Including Prebuilt Applications

If an application is developed outside PetaLinux (for example, through the Vitis™ software development platform), you may just want to add the application binary in the PetaLinux root file system. In this case, an application template is created to allow copying of the existing content to target file system.

If the application, library, or module name has '_', uppercase letters, or starts with an uppercase letter, see Recipe Name has ' _ ' or Uppercase Letters or Starts with an Uppercase Letter.

This section explains how to include pre-compiled applications to PetaLinux root file system.

Prerequisites

This section assumes that you have PetaLinux tools software platform ready for building a Linux system customized for your hardware platform. For more information, see Importing Hardware Configuration.

Steps to Include Prebuilt Applications

If your prebuilt application name is myapp, including this into PetaLinux root file system is explained in following steps.

1. Ensure that the pre-compiled code has been compiled for your PetaLinux target architecture, for example, MicroBlaze™ processors, Arm® cores etc.
2. Create an application with the following command.

```
$ petalinux-create -t apps --template install --name myapp --enable
```

3. Change to the newly created application directory.

```
$ cd <plnx-proj-root>/project-spec/meta-user/recipes-apps/myapp/files/
```

4. Remove existing `myapp` app and copy the prebuilt myapp into `myapp/files` directory.

```
$ rm myapp
$ cp <path-to-prebuilt-app> ./
```

> **IMPORTANT!** *You need to ensure that the binary data being installed into the target file system by an install template application is compatible with the underlying hardware implementation of your system.*

Creating and Adding Custom Libraries

This section explains how to add custom Libraries to PetaLinux root file system.

Prerequisites

This section assumes that you have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. For more information, see Importing Hardware Configuration.

Steps to Add Custom Libraries

The basic steps are as follows:

1. Create a user application by running `petalinux-create -t apps` from inside a PetaLinux project on your workstation:

```
$ cd <plnx-proj-root>
$ petalinux-create -t apps --template c --name <user-library-name> --
enable
```

For example:

```
$ petalinux-create -t apps --template c --name libsample --enable
```

Note: If the application name has '_' or uppercase letters or starts with an uppercase letter, see Recipe Name has ' _ ' or Uppercase Letters or Starts with an Uppercase Letter.

The new application sources can be found in the `<plnx-proj-root>/project-spec/meta-user/recipes-apps/libsample` directory.

2. Change to the newly created application directory.

```
$ cd <plnx-proj-root>/project-spec/meta-user/recipes-apps/libsample
```

3. Edit the file `project-spec/meta-user/recipes-apps/libsample/libsample.bb`.

The file should look like the following.

```
#
# This file is the libsample recipe.
#
SUMMARY = "Simple libsample application"
SECTION = "libs"
LICENSE = "MIT"
LIC_FILES_CHKSUM ="file://${COMMON_LICENSE_DIR}/
MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

SRC_URI = "file://libsample.c \
    file://libsample.h \
    file://Makefile \
    "

S = "${WORKDIR}"

PACKAGE_ARCH = "${MACHINE_ARCH}"
PROVIDES = "sample"TARGET_CC_ARCH += "${LDFLAGS}"
do_install() {
    install -d ${D}${libdir}
    install -d ${D}${includedir}
    oe_libinstall -so libsample ${D}${libdir}    install -d -m 0655 ${D}$
{includedir}/SAMPLE
    install -m 0644 ${S}/*.h ${D}${includedir}/SAMPLE/
    }

    FILES_${PN} = "${libdir}/*.so.* ${includedir}/*"
    FILES_${PN}-dev = "${libdir}/*.so"
```

4. Edit the file `project-spec/meta-user/recipes-apps/libsample/files/libsample.c`. The file should look like the following:

```
#include <stdio.h>
#include "libsample.h"

int function()
{
    printf("Hello World!\n");
    return 0;
}

void samplelib()
{
    printf("Hello, Welcome to PetaLinux -- samplelib !\n");
}
```

5. Create a new file `project-spec/meta-user/recipes-apps/libsample/files/libsample.h` and add below line.

```
void samplelib();
```

6. Edit the file `project-spec/meta-user/recipes-apps/libsample/files/Makefile`. Refer to the makefile content at https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842475/PetaLinux+Yocto+Tips#PetaLinuxYoctoTips-HowtoAddPre-builtLibrariesinPetaLinuxorYoctoRecipes for more details.

7. Build recipe.

```
petalinux-build -c libsample
```

Testing User Libraries

Prerequisites

This section assumes that you have built and installed pre-compiled/custom user applications.

Steps to Test User Libraries

1. Create an application using following command.

```
petalinux-create -t apps --template c -n sampleapp --enable
```

2. Modify the file `<plnx-proj-root>/project-spec/meta-user/recipes-apps/sampleapp/sampleapp.bb` as below:

```
#
# This file is the sampleapp recipe.
#

SUMMARY = "Simple sampleapp application"
SECTION = "PETALINUX/apps"
LICENSE = "MIT"
LIC_FILES_CHKSUM =
"file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

SRC_URI = "file://sampleapp.c \
                   "
S = "${WORKDIR}"

DEPENDS = " sample"

do_compile() {
        ${CC} ${CFLAGS} ${LDFLAGS} -o testsamplelib testsamplelib.c -
lsample
}

do_install() {
             install -d ${D}${bindir}
             install -m 0755 sampleapp ${D}${bindir}
}
FILES_${PN} += "sampleapp"
```

3. Edit the file `project-spec/meta-user/recipes-apps/sampleapp/files/sampleapp.c`.

```
#include <stdio.h>
#include <SAMPLE/libsample.h>

int main(int argc, char **argv)
{
    printf("Hello World!\n");
    samplelib();
    return 0;
}
```

4. Build the application using the following command:

```
petalinux-build -c sampleapp
```

5. Boot the newly created system image.
6. Run your user application on the target system console. For example, to run user application sampleapp:

```
# sampleapp
```

7. Confirm that the result of the application is as expected.

Creating and Adding Custom Applications

This section explains how to add custom applications to PetaLinux root file system.

If the application, library, or module name has '_', uppercase letters, or starts with an uppercase letter, see Recipe Name has ' _ ' or Uppercase Letters or Starts with an Uppercase Letter.

Prerequisites

This section assumes that you have PetaLinux tools software platform ready for building a Linux system customized to your hardware platform. For more information, see Importing Hardware Configuration.

Steps to Add Custom Applications

The basic steps are as follows:

1. Create a user application by running `petalinux-create -t apps` from inside a PetaLinux project on your workstation:

```
$ cd <plnx-proj-root>
$ petalinux-create -t apps --template <TYPE> --name <user-application-name> --enable
```

For example, to create a user application called myapp in C (the default):

```
$ petalinux-create -t apps --name myapp --enable
```

or:

```
$ petalinux-create -t apps --template c --name myapp --enable
```

To create a C++ application template, pass the `--template c++` option, as follows:

```
$ petalinux-create -t apps --template c++ --name myapp --enable
```

To create an autoconf application template, pass the `--template autoconf` option, as follows:

```
$ petalinux-create -t apps --template autoconf --name myapp --enable
```

The new application sources can be found in the `<plnx-proj-root>/project-spec/meta-user/recipes-apps/myapp` directory.

2. Change to the newly created application directory.

```
$ cd <plnx-proj-root>/project-spec/meta-user/recipes-apps/myapp
```

You should see the following PetaLinux template-generated files:

Table 20: **Adding Custom Applications Files**

Template	Description
`<plnx-proj-root>/project- spec/meta-user/conf/user-rootfsconfig`	Configuration file template - This file controls the integration of your application into the PetaLinux RootFS menu configuration. It also allows you select or de-select the app and its dev, dbg packages into the target root file system
Makefile	Compilation file template - This is a basic Makefile containing targets to build and install your application into the root file system. This file needs to be modified when you add additional source code files to your project.
README	A file to introduce how to build the user application.
`myapp.c` for C; `myapp.cpp` for C++	Simple application program in either C or C++, depending upon your choice.

Note: If you want to use the build artifacts for debugging with the third party utilities, add the following line in `<plnx-proj-root>/project-spec/meta-user/conf/petalinuxbsp.conf`:

```
RM_WORK_EXCLUDE += "myapp"
```

Note: You can find all build artifacts under `${TMPDIR}/work/aarch64-xilinx-linux/myapp/1.0-r0/`.

Note: Applications created using the `petalinux-create -t apps` command have debug symbols by default in the following path if you comment out `rm_work`: `<plnx-proj-root>/build/conf/local.conf.<plnx-proj-root>/build/tmp/work/aarch64-xilinx-linux/<app-name>/1.0-r0/packages-split/<app-name>-dbg/usr/bin/.debug/<app-name>`.

TIP: *Mapping of Make file clean with `do_clean` in recipe is not recommended. This is because Yocto maintains its own `do_clean`.*

3. `myapp.c/myapp.cpp` file can be edited or replaced with the real source code for your application. If you want to modify your custom user application later, this file should be edited.

CAUTION! *You can delete the app directory if it is no longer required. You must also remove the line: `CONFIG_myapp` from `<plnx-proj-root>/project-spec/meta-user/conf/user-rootfsconfig`. Deleting the directory by keeping the mentioned line throws an error.*

Creating and Adding Custom Kernel Modules

This section explains how to add custom kernel modules to PetaLinux root file system.

If the application, library, or module name has '_', uppercase letters, or starts with an uppercase letter, see Recipe Name has ' _ ' or Uppercase Letters or Starts with an Uppercase Letter.

Prerequisites

This section assumes that you have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. For more information, see Importing Hardware Configuration for more information.

Steps to Add Custom Modules

1. Create a user module by running `petalinux-create -t modules` from inside a PetaLinux project on your workstation:

```
$ cd <plnx-proj-root>
$ petalinux-create -t modules --name <user-module-name> --enable
```

For example, to create a user module called mymodule in C (the default):

```
$ petalinux-create -t modules --name mymodule --enable
```

You can use `-h` or `--help` to see the usage of the `petalinux-create -t modules`. The new module recipe you created can be found in the `<plnx-proj-root>/project-spec/meta-user/recipes-modules/mymodule` directory.

Note: If the module name has '_' or uppercase letters or starts with an uppercase letter, see Recipe Name has ' _ ' or Uppercase Letters or Starts with an Uppercase Letter.

2. Change to the newly created module directory.

```
$ cd <plnx-proj-root>/project-spec/meta-user/recipes-modules/
mymodule
```

You should see the following PetaLinux template-generated files:

Table 21: **Adding Custom Module Files**

Template	Description
Makefile	Compilation file template - This is a basic Makefile containing targets to build and install your module into the root file system. This file needs to be modified when you add additional source code files to your project. Click here to customize the make file.
README	A file to introduce how to build the user module.
mymodule.c	Simple kernel module in C.
`<plnx-proj-root>/project- spec/meta-user/conf/user-rootfsconfig`	Configuration file template - This file controls the integration of your `application/modules/libs` into the PetaLinux RooFS menu configuration system. It also allows you to select or de-select the app and its dev, dbg packages into the target root file system.

3. `mymodule.c` file can be edited or replaced with the real source code for your module. Later if you want to modify your custom user module, you are required to edit this file.

   Note: If you want to use the build artifacts for debugging with the third party utilities, add the following line in `project-spec/meta-user/conf/petalinuxbsp.conf`:

   ```
   RM_WORK_EXCLUDE += "mymodule"
   ```

Note: You can find all build artifacts under `${TMPDIR}/work/aarch64-xilinx-linux/mymodule/1.0-r0/`.

Note: The modules created with `petalinux-create -t` modules have debug symbols by default.

> **CAUTION!** *You can delete the module directory if it is no longer required. Apart from deleting the module directory, you have to remove the line: `CONFIG_mymodule` from `<plnx-proj-root>/project-spec/meta-user/conf/user-rootfsconfig`. Deleting the directory by keeping the mentioned line in `user-rootfsconfig` throws an error.*

Building User Applications

This section explains how to build and install pre-compiled/custom user applications to PetaLinux root file system.

Prerequisites

This section assumes that you have included or added custom applications to PetaLinux root file system (see Creating and Adding Custom Applications).

Steps to Build User Applications

Running `petalinux-build` in the project directory `<plnx-proj-root>` rebuilds the system image including the selected user application myapp. (The output directory for this build process is `<TMPDIR>/work/aarch64-xilinx-linux/myapp/1.0-r0/`.)

```
$ petalinux-build
```

To build myapp into an existing system image:

```
$ cd <plnx-proj-root>
$ petalinux-build -c rootfs
$ petalinux-build -x package
```

Other `petalinux-build` options are explained with `--help`. Some of the build options are:

- To clean the selected user application:

  ```
  $ petalinux-build -c myapp -x do_clean
  ```

- To rebuild the selected user application:

  ```
  $ petalinux-build -c myapp
  ```

  This compiles the application. The compiled executable files are in the `${TMPDIR}/work/aarch64-xilinx-linux/myapp/1.0-r0/` directory.

  If you want to use the build artifacts for debugging with the third party utilities, add the line: `RM_WORK_EXCLUDE += "myapp"` in `<plnx-proj-root>/project-spec/meta-user/conf/petalinuxbsp.conf`. Without this line, the BitBake removes all the build artifacts after building successfully.

- To see all list of tasks for myapp:

  ```
  petalinux-build -c myapp -x listtasks
  ```

- To install the selected user application:

  ```
  $ petalinux-build -c myapp -x do_install
  ```

  This installs the application into the target the root file system host copy: `<TMPDIR>/work/<MACHINE_NAME>-xilinx-linux/petalinux-image-minimal/1.0-r0/rootfs/`.

  TMPDIR can be found in **petalinux-config → Yocto-settings → TMPDIR**. If the project is on local storage, TMPDIR is `<plnx-proj-root>/build/tmp/`.

  If you want to use the build artifacts for debugging with third party utilities, add the following line in `project-spec/meta-user/conf/petalinuxbsp.conf`:

  ```
  RM_WORK_EXCLUDE += "myapp"
  ```

Testing User Applications

Prerequisites

This section assumes that you have built and installed pre-compiled/custom user applications. For more information, see Building User Applications.

Steps to Test User Application

1. Boot the newly created system image on target or QEMU.
2. Confirm that your user application is present on the PetaLinux system by running the following command on the target system login console:

   ```
   # ls /usr/bin
   ```

   Unless you have changed the location of the user application through its Makefile, the user application is placed into the `/usr/bin` directory.
3. Run your user application on the target system console. For example, to run the user application myapp:

   ```
   # myapp
   ```
4. Confirm that the result of the application is as expected.

If the new application is missing from the target file system, ensure that you have completed the `petalinux-build -x package` step as described in the previous section. This ensures that your application binary is copied into the root file system staging area, and that the target system image is updated with this new file system.

Building User Modules

This section explains how to build and install custom user kernel modules to PetaLinux root file system.

Prerequisites

This section assumes that you have included or added custom modules to PetaLinux root file system (see Creating and Adding Custom Kernel Modules).

Steps to Build User Modules

Running `petalinux-build` in the project directory "`<plnx-proj-root>`" rebuilds the system image including the selected user module mymodule. (The output directory for this build process is `<TMPDIR>/work/<MANCHINE_NAME>-xilinx-linux/mymodule/1.0-r0/`)

```
$ petalinux-build
```

To build mymodule into an existing system image:

```
$ cd <plnx-proj-root>
$ petalinux-build -c rootfs
$ petalinux-build -x package
```

Other `petalinux-build` options are explained with `--help`. Some of the build options are:

- To clean the selected user module:

  ```
  $ petalinux-build -c mymodule -x do_cleansstate
  ```

- To rebuild the selected user module:

  ```
  $ petalinux-build -c mymodule
  ```

  This compiles the module. The compiled executable files are placed in `<TMPDIR>/work/<MANCHINE_NAME>-xilinx-linux/mymodule/1.0-r0/` directory.

- To see all list of tasks for this module:

  ```
  $ petalinux-build -c mymodule -x listtasks
  ```

- To install the selected user module:

  ```
  $ petalinux-build -c mymodule -x do_install
  ```

  This installs the module into the target the root file system host copy: `<TMPDIR>/work/<MACHINE_NAME>-xilinx-linux/petalinux-image-minimal/1.0-r0/rootfs/`.

  TMPDIR can be found in **petalinux-config → Yocto-settings → TMPDIR**. If the project is on local storage, TMPDIR is `<${PROOT}>/build/tmp/`.

  If you want to use the build artifacts for debugging with third party utilities, add the following line in `project-spec/meta-user/conf/petalinuxbsp.conf`:

  ```
  RM_WORK_EXCLUDE += "mymodule"
  ```

PetaLinux Auto Login

This section explains how to login directly from boot without having to enter login credentials.

Prerequisites

This section assumes that you have PetaLinux tools software platform ready for building a Linux system customized to your hardware platform. For more information, see Importing Hardware Configuration.

Steps for PetaLinux Auto Login

Follow these steps to find the PetaLinux Auto Login feature:

1. Change to the root directory of your PetaLinux project.

   ```
   cd <plnx-proj-root>
   ```

2. Run `petalinux-config -c rootfs`.
3. Select **Image Features → auto-login**.

   ***Note*:** Auto-login is enabled by default. Deselect this option to disable this and add user name and password.

4. Save the configuration and exit.
5. Run `petalinux-build`.

Application Auto Run at Startup

This section explains how to add applications that run automatically at system startup.

Prerequisites

This section assumes that you have already added and built the PetaLinux application. For more information, see Creating and Adding Custom Applications and Building User Applications.

Steps for Application Auto Run at Startup

If you have a prebuilt or newly created custom user application myapp located in your PetaLinux project at `<plnx-proj-root>/project-spec/meta-user/recipes-apps/`, you may want to execute it at system startup. The steps to enable that are:

If you have prebuilt application and you have not included in PetaLinux Root file system, see Including Prebuilt Applications. If you want to create custom application and install it in PetaLinux Root file system, see Creating and Adding Custom Applications. If your auto run application is a blocking application which never exits, launch this application as a daemon.

1. Create and install a new application named myapp-init

```
cd <plnx-proj-proot>
petalinux-create -t apps --template install -n myapp-init  --enable
```

2. Edit the file `project-spec/meta-user/recipes-apps/myapp-init/myapp-init.bb`. The file should look like the following:

```
#
# This file is the myapp-init recipe.
#
SUMMARY = "Simple myapp-init application"
SECTION = "PETALINUX/apps"
LICENSE = "MIT"
LIC_FILES_CHKSUM ="file://${COMMON_LICENSE_DIR}/
MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

SRC_URI = "file://myapp-init \
        "
S = "${WORKDIR}"

FILESEXTRAPATHS_prepend := "${THISDIR}/files:"

inherit update-rc.d

INITSCRIPT_NAME = "myapp-init"
INITSCRIPT_PARAMS = "start 99 S ."

do_install() {
        install -d ${D}${sysconfdir}/init.d
        install -m 0755 ${S}/myapp-init ${D}${sysconfdir}/init.d/myapp-
init
}
FILES_${PN} += "${sysconfdir}/*"
```

3. To run myapp as daemon, edit the file `project-spec/meta-user/recipes-apps/myapp-init/files/myapp-init`.

   The file should look like below:

```
#!/bin/sh
DAEMON=/usr/bin/myapp-init
start ()
{
        echo " Starting myapp-init"
        start-stop-daemon -S -o --background -x $DAEMON
}
stop ()
{
        echo " Stoping myapp-init"
        start-stop-daemon -K -x $DAEMON
}
restart()
{
        stop
        start
}
[ -e $DAEMON ] || exit 1

        case "$1" in
                start)
                        start; ;;
```

```
                stop)
                        stop; ;;
                restart)
                        restart; ;;
                *)
                        echo "Usage: $0 {start|stop|restart}"
                        exit 1
        esac
exit $?
```

4. Run `petalinux-build`.

Adding Layers

You can add layers to the PetaLinux project. The upstream layers for the Gatesgarth version of Yocto can be found at http://layers.openembedded.org/layerindex/branch/gatesgarth/layers/.

The following steps demonstrate how to add the meta-my layer into the PetaLinux project.

1. Copy or create a layer in `<proj_root>/project-spec/meta-mylayer`.
2. Run **petalinux-config → Yocto Settings → User Layers**.
3. Enter the following:

```
${PROOT}/project-spec/meta-mylayer
```

4. Save and exit.
5. Verify by viewing the file in `<proj_root>/build/conf/bblayers.conf`.

***Note*:** The 2021 release of the PetaLinux tool is on the Gatesgarth base line. Choose the layers/recipes from the Gatesgarth branch only. Some of the layers/recipes might not be compatible with the current architecture. You are responsible for all additional layers/recipes.

***Note*:** You can also add a layer that is outside your project; such layers can be shared across projects. Ensure that the added layer has `<layer>/conf/layer.conf`; otherwise, it causes build errors.

IMPORTANT! *If you want to change the layer priority, you can update `${PROOT}/project-spec/meta-mylayer/conf/layer.conf` to set BBFILE_PRIORITY_meta-mylayer = 6 (0 to 10, higher values have higher priority).*

Adding an Existing Recipe into the Root File System

Most of the root file system menu config is static. These are the utilities that are supported by Xilinx. You can add your own layers in a project or add existing additional recipes from the existing layers in PetaLinux. Layers in PetaLinux can be found in `<plnx-proj-root>/components/yocto/layers`.

By default, `iperf3` is not in the root file system menuconfig. The following example demonstrates adding the `iperf3` into the root file system menuconfig.

1. The location of the recipe is `<plnx-proj-root>/components/yocto/layers/meta-openembedded/meta-oe/recipes-benchmark/iperf3/iperf3_3.2.bb`.
2. Add the following line in `<plnx-proj-root>/project-spec/meta-user/conf/user-rootfsconfig`.

   ```
   CONFIG_iperf3
   ```

3. Run `petalinux-config -c rootfs`.
4. Select **user packages → iperf3**. Enable it, save and exit.
5. Run `petalinux-build`.

   Note: It is your responsibility to add the recipes in the layers available in PetaLinux tools, apart from PetaLinux default RootFS menuconfig.

   Note: The above procedure is applicable only to the recipes from the existing layers.

IMPORTANT! *All recipes which are in `petalinux-image-full` have sstate locked. To unlock you have to add `SIGGEN_UNLOCKED_RECIPES += "my-recipe"` in `project-spec/meta-user/conf/petalinuxbsp.conf`.*

For example, you have changes to be made in mtd-utils package, so you have created a .bbappend for the same without SIGGEN_UNLOCKED_RECIPES += "mtd-utils" in `project-spec/meta-user/conf/petalinuxbsp.conf`. During project build, you should see the following warning. Your changes for the package are not included in the build.

```
"The mtd-utils:do_fetch sig is computed to be
92c59aa3a7c524ea790282e817080d0a, but the sig is locked to
9a10549c7af85144d164d9728e8fe23f in SIGGEN_LOCKEDSIGS_t"
```

Adding a Package Group

One of the best approaches for customizing images is to create a custom package group to be used for building the images. Some of the package group recipes are shipped with the PetaLinux tools.

For example:

```
<plnx-proj-root>/components/yocto/layers/meta-petalinux/recipes-core/
packagegroups/packagegroup-petalinux-self-hosted.bb
```

The name of the package group should be unique and should not conflict with the existing recipe names.

You can create custom package group, for example, an ALSA package group would look like:

```
DESCRIPTION = "PetaLinux ALSA supported Packages"

inherit packagegroup

ALSA_PACKAGES = " \
        alsa-lib \
        alsa-plugins \
        alsa-tools \
        alsa-utils \
        alsa-utils-scripts \
        pulseaudio \
        "
RDEPENDS_${PN}_append = " \
        ${ALSA_PACKAGES} \
        "
```

This can be added to `<plnx-proj-root>/meta-user/recipes-core/packagegroups/packagegroup-petalinux-alsa.bb`.

To add this package group in RootFS menuconfig, add `CONFIG_packagegroup-petalinux-alsa` in `<plnx-proj-root>/project-spec/meta-user/conf/user-rootfsconfig` to reflect in menuconfig.

Then launch `petalinux-config -c rootfs`, select **user packages → packagegroup-petalinux-alsa**, save and exit. Then run `petalinux-build`.

Appending Root File System Packages

In earlier releases, to add new packages to the root file system, you had to edit the `<plnx-proj-root>/project-spec/meta-user/recipes-core/images/petalinux-image-full.bbappend` file. For example:

```
IMAGE_INSTALL_append = "opencv"
```

From 2020.1 release onwards, you have to use the `<plnx-proj-root>/project-spec/meta-user/conf/user_rootfsconfig` file to append new root file system packages to PetaLinux images. For example:

```
CONFIG_opencv
```

Debugging

Debugging the Linux Kernel in QEMU

This section describes how to debug the Linux Kernel inside QEMU using the GNU debugger (GDB). Note that this function is only tested with Zynq®-7000 devices. For more information, see *Vitis Unified Software Platform Documentation: Embedded Software Development* (UG1400).

Prerequisites

This section assumes that you have built PetaLinux system image. For more information, see Build System Image.

Steps to Debug the Linux Kernel in QEMU

1. Launch QEMU with the currently built Linux by running the following command:

   ```
   $ petalinux-boot --qemu --kernel
   ```

2. Watch the QEMU console. You should see the details of the QEMU command. Get the GDB TCP port from `-gdb tcp:<TCP_PORT>`.

3. Open another command console (ensuring the PetaLinux settings script has been sourced), and change to the Linux directory:

   ```
   $ cd "<plnx-proj-root>/images/linux"
   ```

4. Start GDB on the vmlinux kernel image in command mode:

   ```
   $ petalinux-util --gdb vmlinux
   ```

   You should see the GDB prompt. For example:

   ```
   GNU gdb (GDB) 8.3.1
   Copyright (C) 2019 Free Software Foundation, Inc.
   License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/
   gpl.html>
   This is free software: you are free to change and redistribute it.
   There is NO WARRANTY, to the extent permitted by law.
   Type "show copying" and "show warranty" for details.
   This GDB was configured as "--host=x86_64-oesdk-linux --target=aarch64-
   xilinx-elf".
   ```

```
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
vmlinux: No such file or directory.
(gdb)
```

5. Attach to the QEMU target in GDB by running the following GDB command:

```
(gdb) target remote :9000
```

6. To let QEMU continue execution:

```
(gdb) continue
```

7. You can use `Ctrl+C` to interrupt the kernel and get back the GDB prompt.
8. You can set break points and run other GDB commands to debug the kernel.

CAUTION! *If another process is using port 9000, `petalinux-boot` attempts to use a different port. See the output of `petalinux-boot` to determine what port was used. In the following example, port 9001 is used: `INFO: qemu-system-arm ... -gdb tcp::9001 ...`*

TIP: *It may be helpful to enable kernel debugging in the kernel configuration menu (**petalinux-config -- kernel → Kernel hacking → Kernel debugging**), so that kernel debug symbols are present in the image.*

Troubleshooting

This section describes some common issues you may experience while debugging the Linux kernel in QEMU.

Table 22: **Debugging the Linux Kernel in QEMU Troubleshooting**

Problem / Error Message	Description and Solution
`(gdb) target remote W.X.Y.Z:9000:9000: Connection refused.`	**Problem Description:** GDB failed to attach the QEMU target. This is most likely because the port 9000 is not the one QEMU is using **Solution:** Check your QEMU console to ensure QEMU is running. Watch the Linux host command line console. It should show the full QEMU commands and you should be able to see which port is being used by QEMU.

Debugging Applications with TCF Agent

This section describes debugging user applications with the Eclipse Target Communication Framework (TCF) Agent. The procedure for debugging applications with TCF agent remains the same for Versal™ platform, Zynq® UltraScale+™ MPSoC, and Zynq-7000 devices. This section describes the basic debugging procedure for Zynq platform user application myapp.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- Working knowledge of the Vitis™ software platform. For more information, see *Vitis Unified Software Platform Documentation: Embedded Software Development* (UG1400).
- The PetaLinux Working Environment is properly set. For more information, see PetaLinux Working Environment Setup.
- You have created a user application and built the system image including the selected user application. For more information, see Building User Applications.

Preparing the Build System for Debugging

1. Change to the project directory:

```
$ cd <plnx-proj-root>
```

2. Run `petalinux-config -c rootfs` on the command console:

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

3. Scroll down the Linux/RootFS configuration menu to file system packages.

```
admin   --->
audio   --->
base   --->
baseutils   --->
benchmark   --->
bootloader   --->
console   --->
devel   --->
fonts   --->
kernel   --->
libs   --->
misc   --->
multimedia   --->
net   --->
network   --->
optional   --->
power management --->
utils   --->
x11   --->
```

4. Select **misc** submenu:

```
admin   --->
audio   --->
base   --->
baseutils   --->
benchmark   --->
bootloader   --->
console   --->
devel   --->
fonts   --->
kernel   --->
libs   --->
misc   --->
multimedia   --->
net   --->
network   --->
optional   --->
power management --->
utils   --->
x11   --->
```

5. Packages are in alphabetical order. Navigate to the letter 't', as shown below:

```
serf   --->
sysfsutils   --->
sysvinit-inittab   --->
tbb   --->
tcf-agent --->
texi2html   --->
tiff   --->
trace-cmd   --->
util-macros   --->
v4l-utils   --->
```

6. Ensure that tcf-agent is enabled.

```
[*] tcf-agent
[ ] tcf-agent-dev
[ ] tcf-agent-dbg
```

7. Select **console/network** submenu, and then click into **dropbear** submenu. Ensure "dropbear-openssh-sftp-server" is enabled.

```
[*] dropbear
```

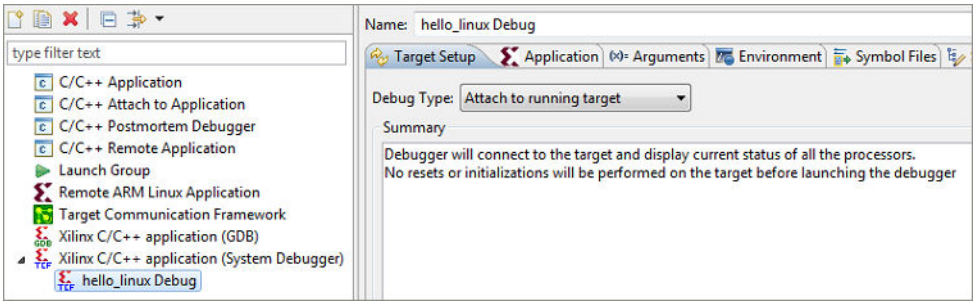
8. Select **console/network → openssh**. Ensure that "openssh-sftp-server" is enabled.
9. Exit the menu.
10. Rebuild the target system image including myapp. For more information, see Build System Image.

Performing a Debug Session

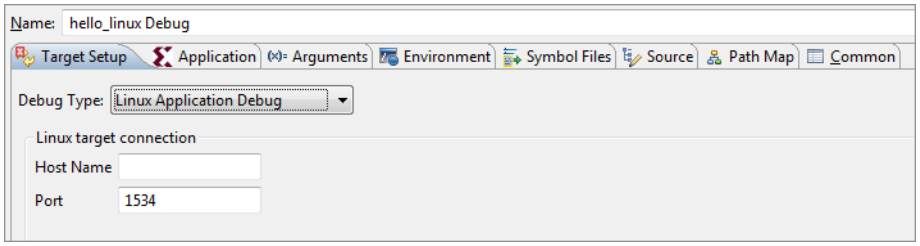
To perform a debug session, follow these steps:

1. Launch the Vitis software platform.

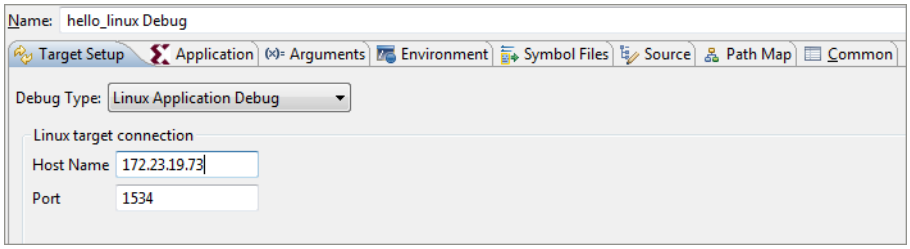
2. Create a Linux application.
3. Select the application you want to debug.
4. Select **Run → Debug Configurations**.
5. Click **Launch on Hardware (Single Application Debug)** to create a new configuration.



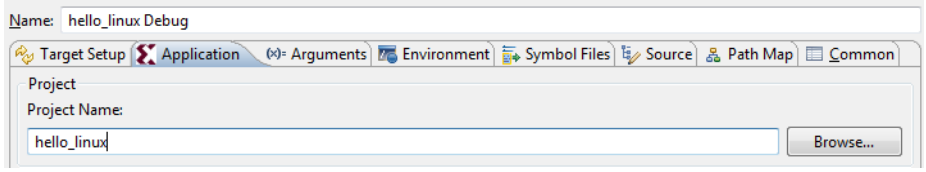
6. In the Debug Configuration window:
   a. Click the **Target Setup** tab.
   b. From the Debug Type drop-down list, select **Linux Application Debug**.



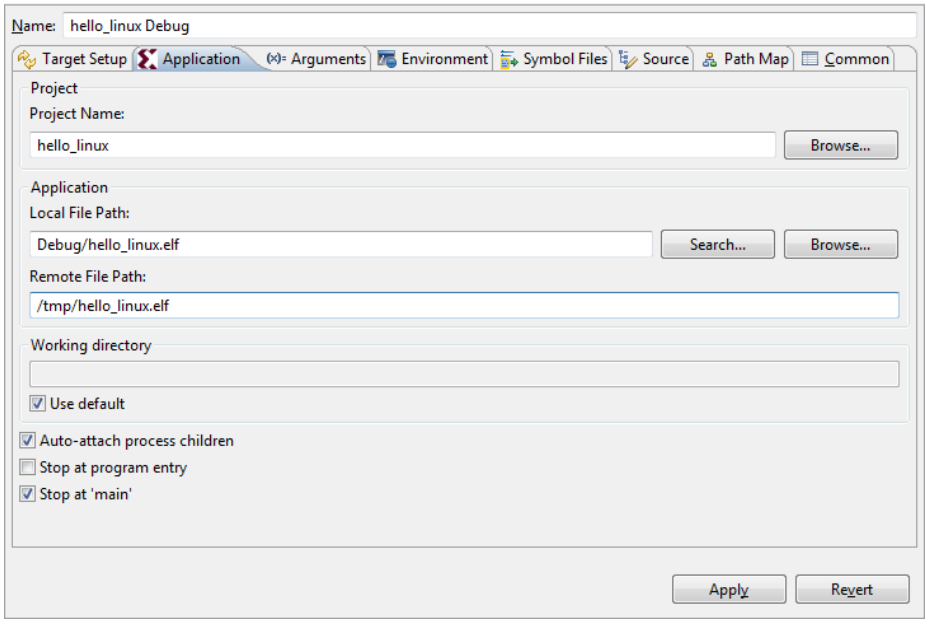
   c. Provide the Linux host name or IP address in the Host Name field.



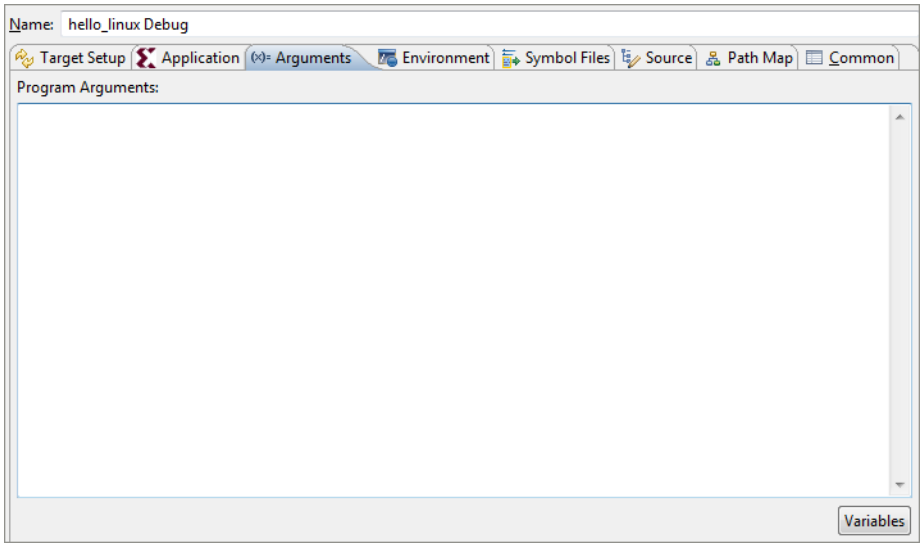
   d. By default, tcf-agent runs on the 1534 port on the Linux. If you are running tcf-agent on a different port, update the **Port** field with the correct port number.
   e. In the Application Tab, click **Browse** and select the project name. The Vitis software platform automatically fills the information in the application.



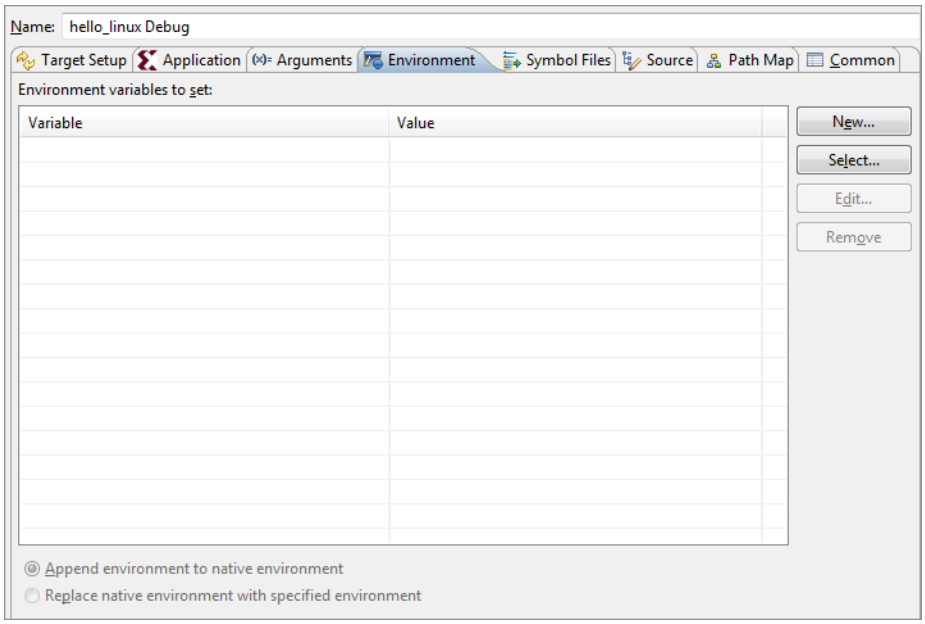
   f. In the Remote File Path field, specify the path where you want to download the application in Linux.



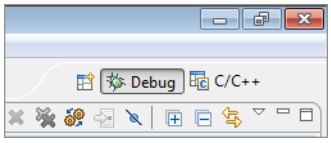
g. If your application is expecting some arguments, specify them in the Arguments tab.

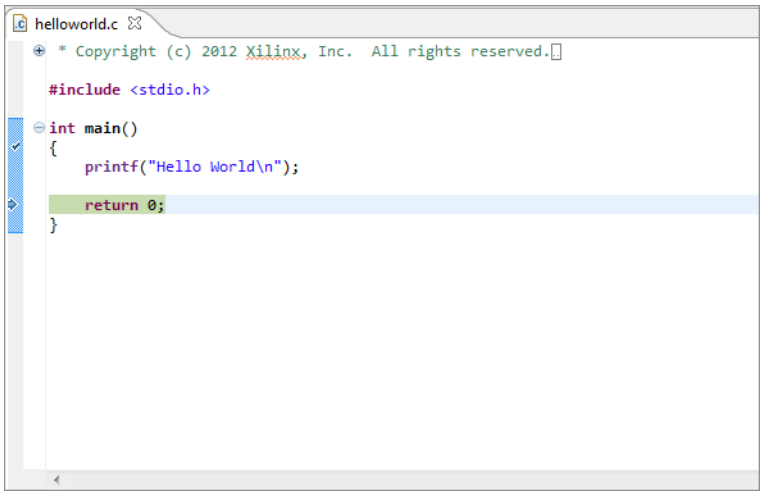


h. If your application is expecting to set some environment variables, specify them in the Environment tab.



i. Click the **Debug** button. A separate console automatically opens for process standard I/O operations.





j. Click the **Terminate** button to terminate the application.

7. Enter the Local File Path to your compiled application in the project directory. For example, `<TMPDIR>/work/aarch64-xilinx-linux/hello-linux/1.0-r0/image/usr/bin/`.

   Note: While creating the application, you need to add `RM_WORK_EXCLUDE += "hello-linux"` in `project-spec/meta-user/conf/petalinuxbsp.conf`, otherwise the images will not be available for debugging.

8. The remote file path on the target file system should be the location where the application can be found. For example, `/usr/bin/hello-linux`.
9. Select **Debug** to Apply the configuration and begin the Debug session. (If asked to switch to Debug Perspective, accept).

Debugging Zynq UltraScale+ MPSoC and Versal ACAP Applications with GDB

PetaLinux supports debugging Zynq® UltraScale+™ MPSoC and Versal™ user applications with GDB. This section describes the basic debugging procedure. For more information, refer to *Vitis Unified Software Platform Documentation: Embedded Software Development* (UG1400).

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- The PetaLinux Working Environment is properly set. For more information, see PetaLinux Working Environment Setup.
- You have created a user application and built the system image including the selected user application. For more information, see Building User Applications.

Preparing the Build System for Debugging

1. Change to the project directory:

```
$ cd <plnx-proj-root>
```

2. Add the following lines in `<plnx-proj-root>/project-spec/meta-user/conf/user-rootfsconfig`:

```
CONFIG_myapp-dev
CONFIG_myapp-dbg
```

3. Add the following lines in `<plnx-proj-root>/project-spec/meta-user/recipe-apps/myapp/myapp.bb`.

```
DEBUG_FLAGS = "-g3 -O0"

# Specifies to build packages with debugging information
DEBUG_BUILD = "1"
```

```
# Do not remove debug symbols
INHIBIT_PACKAGE_STRIP = "1"

# OPTIONAL: Do not split debug symbols in a separate file
INHIBIT_PACKAGE_DEBUG_SPLIT = "1"
```

4. Run `petalinux-config -c rootfs` on the command console:

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

5. Scroll down the user packages Configuration menu to Debugging:

```
Filesystem Packages    --->
PetaLinux Package Groups --->
apps    --->
user packages    --->
PetaLinux RootFS Settings   --->
```

6. Select **user packages**.

```
[X] myapp-dbg
[ ] myapp-dev
```

7. Select **myapp-dbg**. Exit the myapp sub-menu.
8. Exit the user packages sub-menu, and select **Filesystem Packages → misc → gdb**.
9. Select **gdb**, and ensure that the GDB server is enabled:

```
[ ] gdb
[ ] gdb-dev
[X] gdbserver
[ ] gdb-dbg
```

10. Exit the menu and select **<Yes>** to save the configuration.
11. Select **Filesystem Packages → misc → tcf-agent**.
12. Deselect **tcf-agent** as it uses the same 1534 port. Exit the menu and save the configuration.
13. Rebuild the target system image. Add the following line in `<plnx-proj-root>/project-spec/meta-user/conf/petalinuxbsp.conf`.

```
RM_WORK_EXCLUDE += "myapp"
```

For more information, see Build System Image.

Performing a Debug Session

1. Boot your board (or QEMU) with the new image created above.
2. For the QEMU boot to redirect the host port, issue the following qemu-args.

```
petalinux-boot --qemu --kernel --qemu-args "-net nic -net nic -net nic -
net nic,netdev=gem3 -netdev user,id=gem3,hostfwd=tcp::1534-:1534"
```

Note: Based on the gem number, provide the `-net nic` options. You can find the gem number in the `<plnx-proj-root>/components/plnx_workspace/device-tree/device-tree/pcw.dtsi` file.

3. Run `gdbserver` with the user application on the target system console (set to listening on port 1534):

```
root@plnx_aarch64:~# gdbserver :1534 /usr/bin/myapp
Process /bin/myapp created; pid = 73
Listening on port 1534
```

1534 is the `gdbserver` port - it can be any unused port number

4. On the workstation, navigate to the compiled user application's directory:

```
$ cd <<TMPDIR>/work/cortexa72-cortexa53-xilinx-linux/myapp/1.0-r0/image/usr/bin/
```

5. Run GDB client.

```
$ petalinux-util --gdb myapp
```

The GDB console starts:

```
...
GNU gdb (crosstool-NG 1.18.0) 7.6.0.20130721-cvs
...
(gdb)
```

6. In the GDB console, connect to the target machine using the command:
   - Use the IP address of the target system, for example: 192.168.0.10. If you are not sure about the IP address, run `ifconfig` on the target console to check.
   - Use the port 1534. If you select a different GDB server port number in the earlier step, use that value instead.

> **IMPORTANT!** *If debugging on QEMU, refer to the QEMU Virtual Networking Modes for information regarding IP and port redirection when testing in non-root (default) or root mode. For example, if testing in non-root mode, you should use local host as the target IP in the subsequent steps.*

```
(gdb) target remote 192.168.0.10:1534
```

The GDB console attaches to the remote target. The GDB server on the target console displays the following confirmation, where the host IP is displayed:

```
Remote Debugging from host 192.168.0.9
```

7. Before starting the execution of the program, create some breakpoints. Using the GDB console you can create breakpoints throughout your code using function names and line numbers. For example, create a breakpoint for the `main` function:

```
(gdb) break main
Breakpoint 1 at 0x10000444: file myapp.c, line 10.
```

8. Run the program by executing the continue command in the GDB console. GDB begins the execution of the program.

```
(gdb) continue
Continuing.
Breakpoint 1, main (argc=1, argv=0xbffffe64) at myapp.c:10
10  printf("Hello, PetaLinux World!\n");
```

9. To print a list of the code at current program location, use the `list` command.

```
(gdb) list
5 */
6 #include <stdio.h>
7
8 int main(int argc, char *argv[])
9 {
10 printf("Hello, PetaLinux World!\n");
11 printf("cmdline args:\n");
12 while(argc--)
13 printf("%s\n",*argv++);
14
```

10. Try the `step`, `next` and `continue` commands. Breakpoints can be set and removed using the `break` command. More information on the commands can be obtained using the GDB console `help` command.
11. The GDB server application on the target system exits when the program has finished running. Here is an example of messages shown on the console:

```
Hello, PetaLinux World!
cmdline args:
/usr/bin/myapp
Child exited with status 0
GDBserver exiting
root@plnx_aarch64:~#
```

TIP: *A `.gdbinit` file is automatically created to setup paths to libraries. You may add your own GDB initialization commands at the end of this file.*

Going Further with GDB

Visit www.gnu.org for more information. For information on general usage of GDB, refer to the GDB project documentation.

Troubleshooting

This section describes some common issues you may experience while debugging applications with GDB.

Table 23: **Debugging Zynq UltraScale+ MPSoC Applications with GDB Troubleshooting**

Problem / Error Message	Description and Solution
`GDB error message: <IP Address>:<port>: Connection refused. GDB cannot connect to the target board using <IP>: <port>`	**Problem Description:** This error message indicates that the GDB client failed to connect to the GDB server. **Solution:** Check whether the `gdbserver` is running on the target system. Check whether there is another GDB client already connected to the GDB server. This can be done by looking at the target console. If you can see Remote Debugging from host <IP>, it means there is another GDB client connecting to the server. Check whether the IP address and the port are correctly set.

Debugging Individual PetaLinux Components

PMU Firmware

https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841724/PMU+Firmware#PMUFirmware-DebuggingPMUFWusingSDK

FSBL

https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842019/FSBL#FSBL-WhatarevariouslevelsofdebugprintsinFSBL

U-Boot

https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842557/Debug+U-boot

Linux

https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/123011167/Linux+Debug+infrastructure+KProbe+UProbe+LTTng

https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/123011146/Linux+Debug+infrastructure+Kernel+debugging+using+KGDB

Advanced Configurations

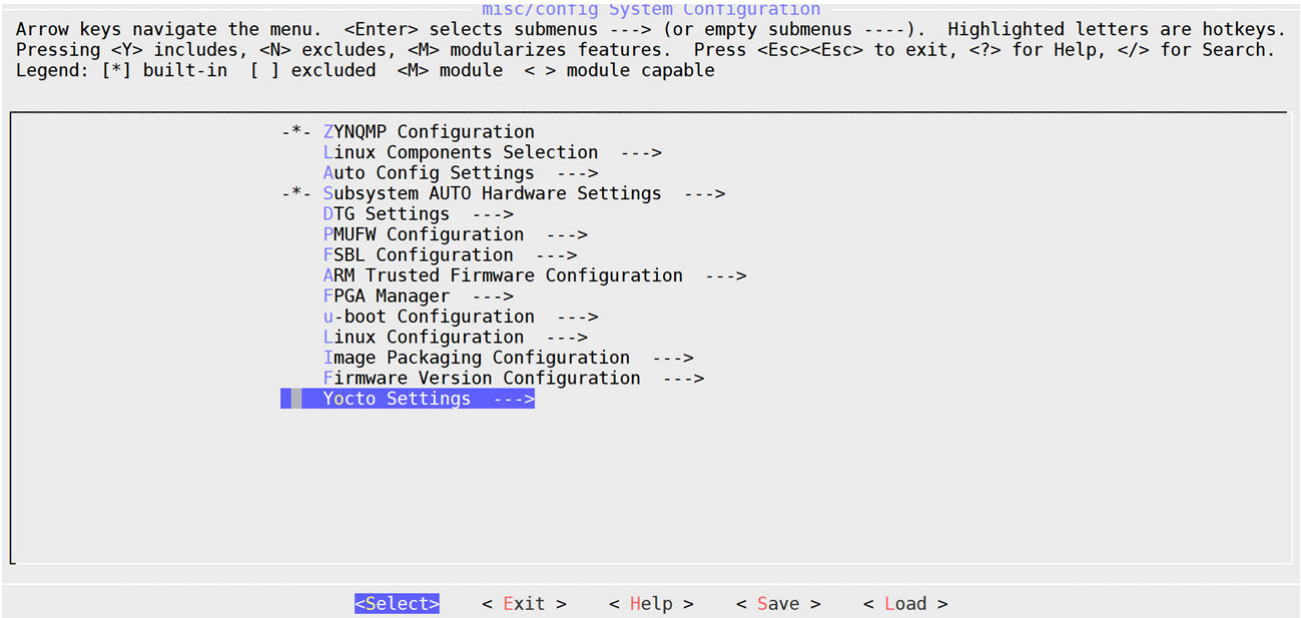
Menuconfig Usage

To select a menu/submenu which was deselected before, press the down arrow key to scroll down the menu or the up arrow key to scroll up the menu. Once the cursor is on the menu, then press **y**. To deselect a menu/submenu, follow the same process and press **n** at the end.

PetaLinux Menuconfig System

The PetaLinux menuconfig system configuration for Zynq® UltraScale+™ MPSoC is shown in the following figure:

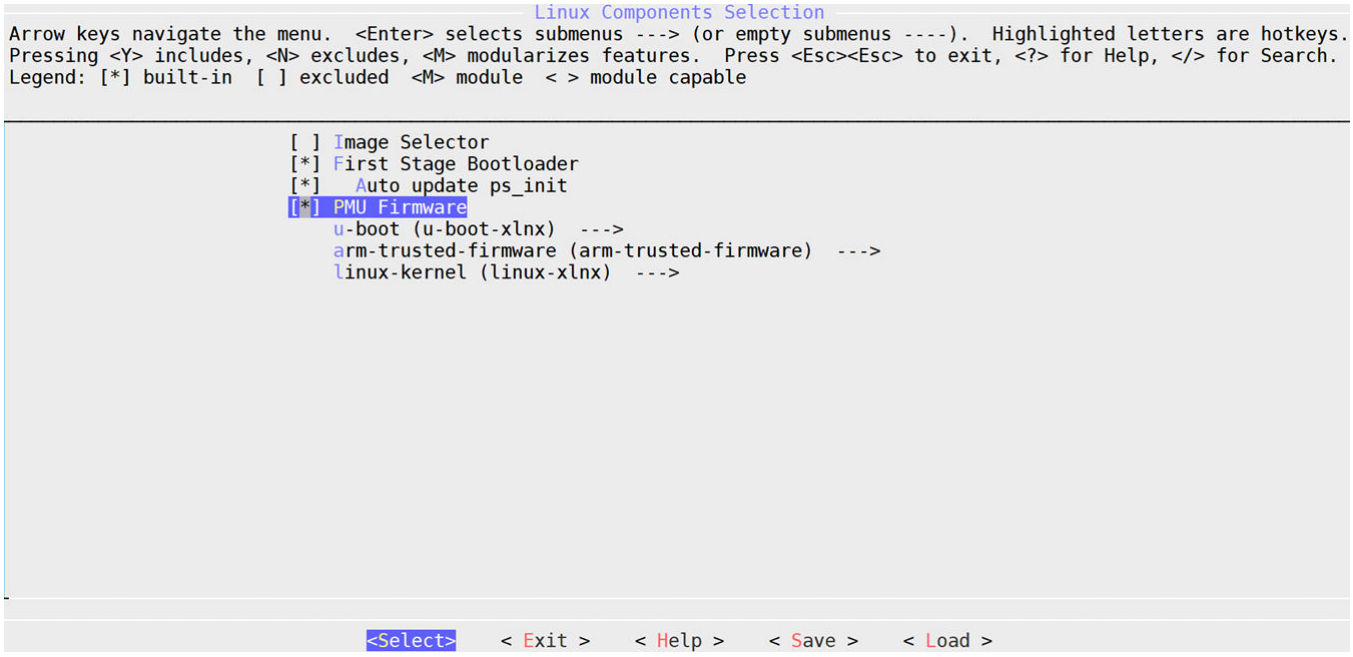
Figure 2: **PetaLinux Menuconfig System**



Linux Components Selection

For Zynq UltraScale+ MPSoC, the Linux system components available in the sub-menu are as follows:

Figure 3: **Linux Components Selection**



- Image Selector (Zynq UltraScale+ MPSoC only)
- First stage boot loader
- PMU firmware, for Zynq® UltraScale+™ MPSoC only
- PLM and PSM firmware for Versal™ platform only
- U-Boot
- Kernel
- ATF, for Zynq UltraScale+ MPSoC and Versal platforms only

For ATF, U-Boot, and kernel there are three available options:

- **Default:** The default component is shipped with the PetaLinux tool.
- **External source:** When you have a component downloaded at any specified location, you can feed your component instead of the default one through this configuration option.

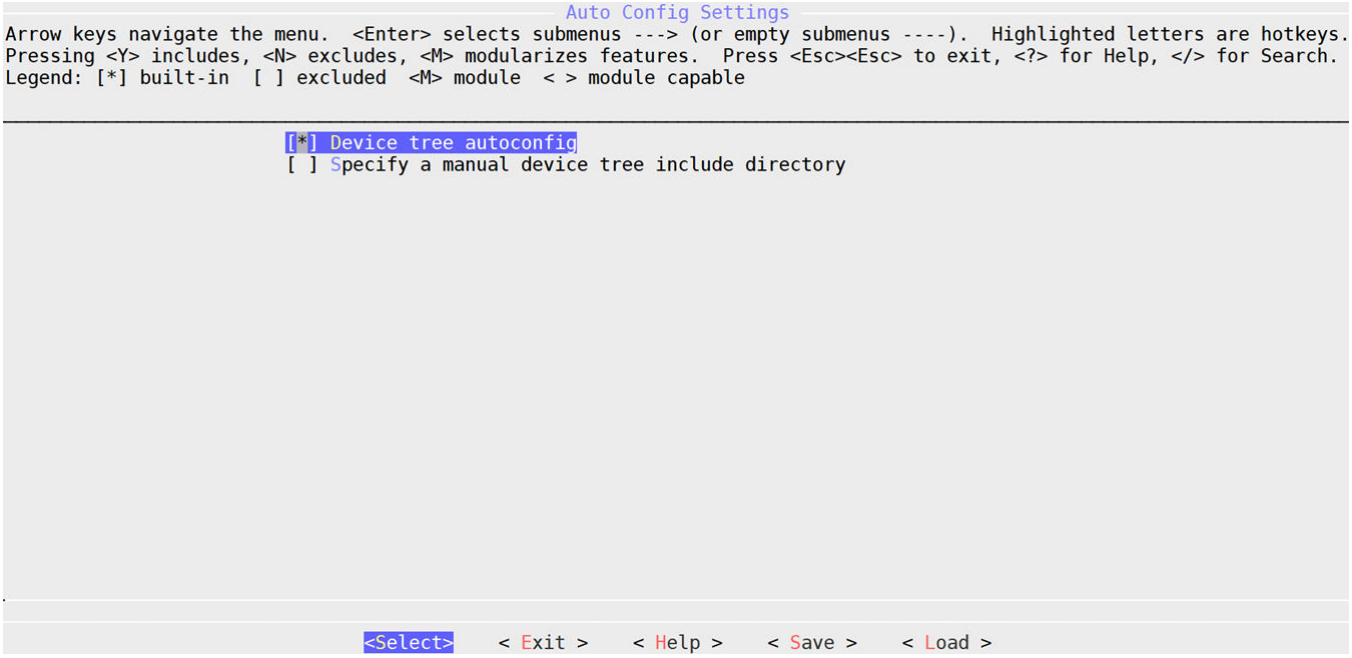
  ***Note*:** The external source folder is required to be unique to a project and its user, but the content can be modified. If the external source is a git repository, its checked out state should be appropriate for building this project.

- **Remote:** If you want to build a component which was on a custom git repository, this configuration option has to be used.

Auto Config Settings

When a component is selected to enable automatic configuration (autoconfig) in the system-level menuconfig, its configuration files are automatically updated when the `petalinux-config` is run.

Figure 4: **Auto Config Settings**



***Note*:** The kernel autoconfig and u-boot auto config options are only available for MicroBlaze processors. They are unavailable for other architectures.

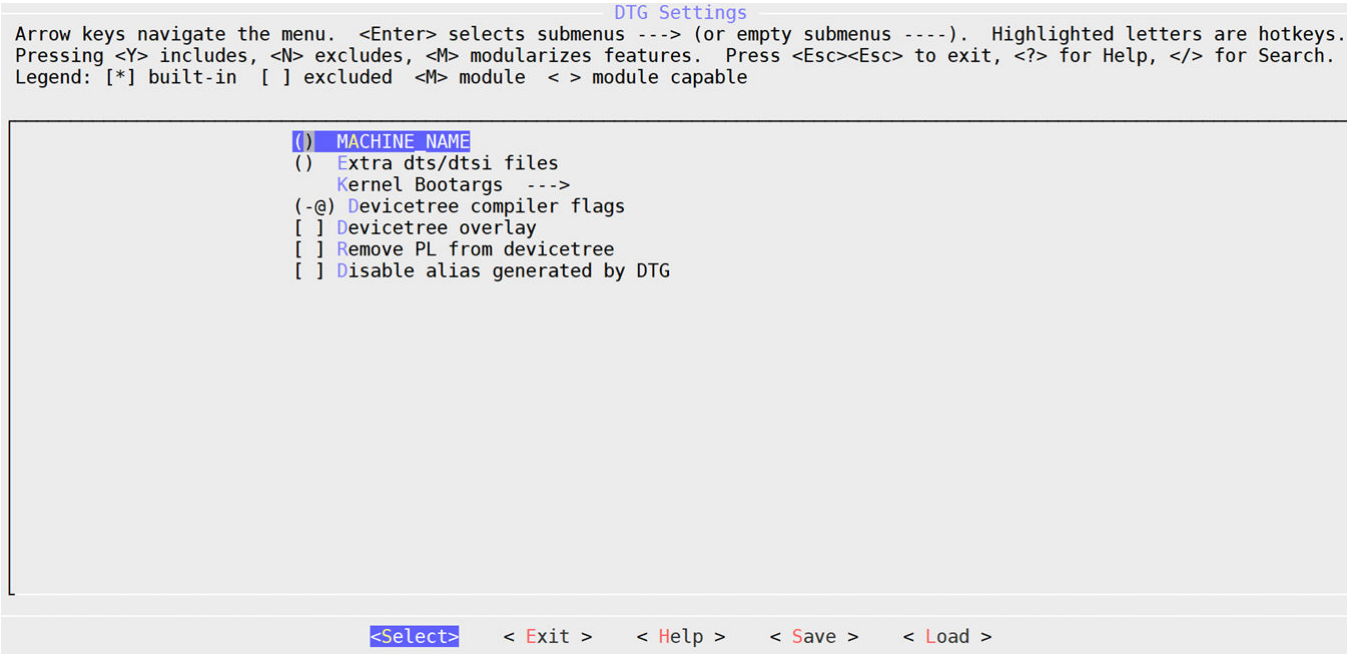
Table 24: **Components and their Configuration Files**

Component in the Menu	Files Impacted when the Autoconfig is enabled
Device tree	The following files are in `<plnx-proj-root>/components/plnx_workspace/device-tree/device-tree/` • `skeleton.dtsi` (Zynq-7000 devices only) • `zynq-7000.dtsi` (Zynq-7000 devices only) • `zynqmp-clk-ccf.dtsi` (Zynq UltraScale+ MPSoC only) • `pcw.dtsi` (Zynq-7000 devices, Zynq UltraScale+ MPSoC, and Versal devices) • `pl.dtsi` • `system-conf.dtsi` • `system-top.dts` • `<board>.dtsi` • `versal.dtsi` (for Versal devices only) • `versal-clk.dtsi` (for Versal devices which have clock framework.)
kernel (only for MicroBlaze processors)	The following files are in `<plnx-proj-root>/project-spec/configs/linux-xlnx/plnx_kernel.cfg` ***Note***: `plnx_kernel.cfg` is generated only when **petalinux-config → Auto Config Settings → kernel autoconfig** is enabled.
U-Boot (only for MicroBlaze processors)	The following files are in `<plnx-proj-root>/project-spec/configs/u-boot-xlnx/config.cfg` • `config.mk` (MicroBlaze only) • `platform-auto.h` ***Note***: `config.cfg` and `platform-auto.h` are generated only when **petalinux-config → Auto Config Settings → u-boot autoconfig** is enabled.

Building Device Tree Overlays for PS

You can build the device tree overlays for PS using the below shown menu config. You can provide dts/dtsi files with relative/absolute path with space separation. It builds the dtbos and packs them as part of the root filesystem in `/boot/devicetree`.

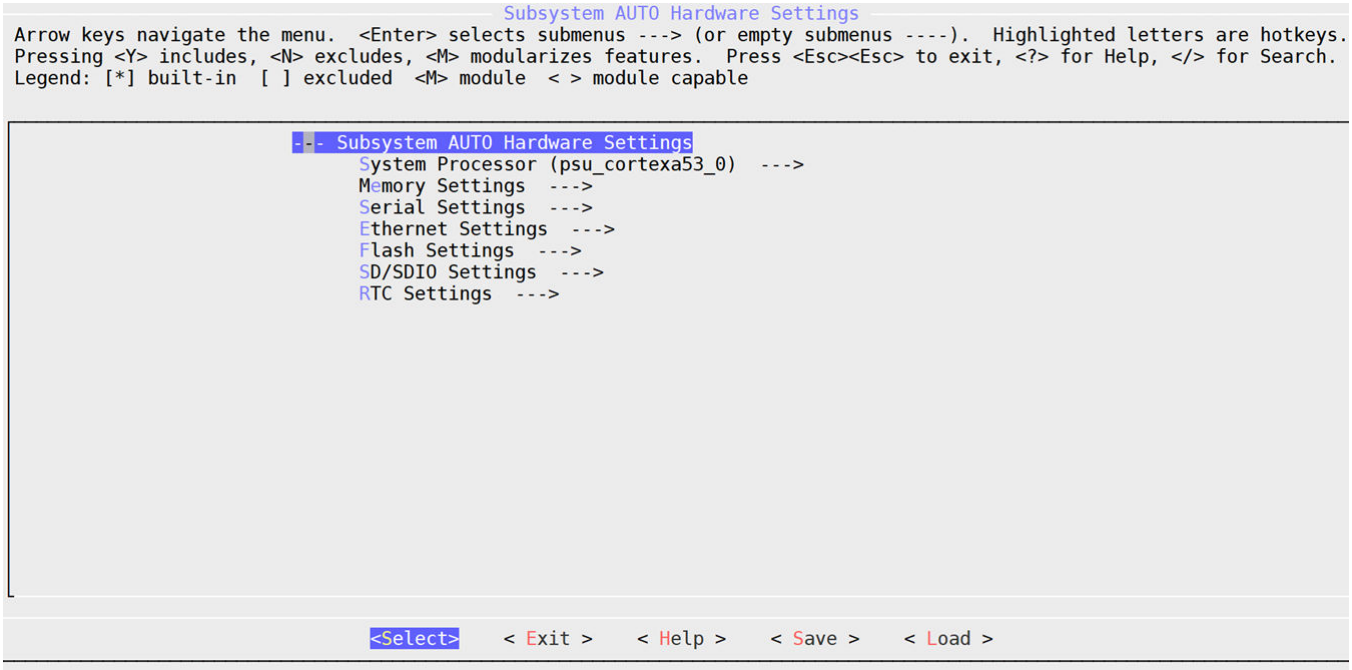
Figure 5: **Carrie Card dtbos**



Subsystem AUTO Hardware Settings

The Subsystem AUTO Hardware settings menu allows you to customize how the Linux system interacts with the underlying hardware platform.

Figure 6: **Subsystem AUTO Hardware Settings**



System Processor

The System processor menu specifies the CPU processor on which the system runs.

Memory Settings

The memory settings menu allows you to:

- Select which memory IP is the primary system memory
- Set the system memory base address
- Set the size of the system memory
- Set the U-Boot text base address offset to a memory high address

The configuration in this menu impacts the memory settings in the device tree and U-Boot automatic configuration (autoconfig) files.

If manual is selected as the primary memory, you are responsible for ensuring proper memory settings for the system.

Serial Settings

The Serial Settings sub-menu allows you to select which serial device is the system's primary STDIN/STDOUT interface. If `manual` is selected as the primary serial, you are responsible for ensuring proper serial interface settings for the system.

Ethernet Settings

The Ethernet Settings sub-menu allows you to:

- Select which Ethernet is the systems' primary Ethernet
- Select to randomize MAC address
- Set the MAC address of the selected primary Ethernet. It is set to ff:ff:ff:ff:ff:ff invalid mac, by default.

  If a MAC address is programmed into the EEPROM, then due to the above invalid MAC address, it will read from the EEPROM. If no MAC address is configured in the EEPROM, then the U-Boot generates a random MAC address. The order in which the U-Boot picks the MAC address is as follows. Refer to the U-Boot documentation for commands to program EEPROM and to configure for the same.

  - The MAC address from the device tree has the highest priority. Anything configured in the **petalinux-config → Subsystem Auto Hardware Settings → Ethernet Settings → MAC address** goes to the device tree.
  - If there is no MAC address in the device tree, then it will check in EEPROM.
  - If there is no MAC address in the EEPROM, then it will generate a random MAC address.

- Set whether to use DHCP or static IP on the primary Ethernet

If manual is selected as the primary Ethernet, you are responsible for ensuring proper Ethernet settings for the system.

Flash Settings

The Flash Settings sub-menu allows you to:

- Select a primary flash for the system
- Set the flash partition table

If manual is selected as the primary flash, you are responsible for the flash settings for the system.

SD/SDIO Settings

The SD/SDIO Settings sub-menu is for Zynq-7000 devices and Zynq UltraScale+ MPSoC only. It allows you to select which SD controller is the system's primary SD card interface.

If manual is selected as the primary flash, you are responsible for the flash settings for the system.

Timer Settings

The Timer Settings sub-menu is for MicroBlaze processors and Zynq UltraScale+ MPSoC. It allows you to select which timer is the primary timer.

IMPORTANT! *A primary timer is required for a MicroBlaze system.*

Reset GPIO Settings

The Reset GPIO Settings sub-menu is for MicroBlaze processors only. It allows you to select which GPIO is the system reset GPIO.

TIP: *MicroBlaze systems use GPIO as a reset input. If a reset GPIO is selected, you can reboot the system from Linux.*

RTC Settings

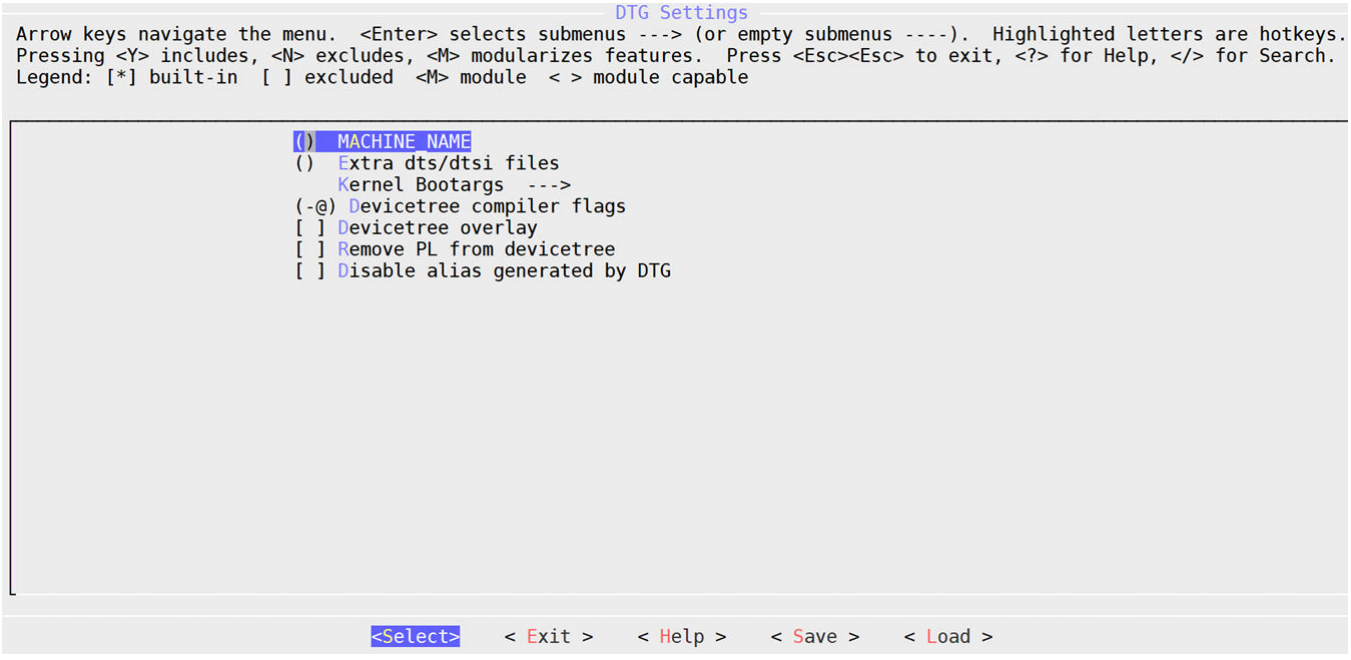
Select an RTC instance that is used as a primary timer for the Linux kernel. If your preferred RTC is not on the list, select **manual** to enable the proper kernel driver for your RTC.

Table 25: **Flash Partition Table**

Bootable Image/U-Boot Environment Partition	Default Partition Name	Description
Boot Image	boot	`BOOT.BIN` for Zynq-7000 devices, Zynq UltraScale+ MPSoC, and Versal ACAP. Relocatable U-Boot BIN file (`u-boot-s.bin`) for MicroBlaze processors.
U-Boot Environment Partition	bootenv	U-Boot environment variable partition. When **primary sd** is selected, U-Boot environment is stored in the first partition. When **primary flash** is selected, U-Boot environment is stored in the partition mentioned in flash partition name option.
Kernel Image	kernel	Kernel image `image.ub` (FIT format)
DTB Image	dtb	If "Advanced bootable images storage Settings" is disabled and a DTB partition is found in the flash partition table settings, PetaLinux configures U-Boot to load the DTB from the partition table. Else, it assumes a DTB is contained in the kernel image.

DTG Settings

Figure 7: **DTG Settings**



Machine Name

For custom boards, do not change the configuration. For Xilinx® evaluation boards, see Table 7: BSP and Machine Names.

Extra dts/dtsi file

Provide the extra device tree files path separated with space. This config should be set with dts/dtsi absolute/full path. Provided dts/dtsi will be build as part of device-tree and deploy into /boot/devicetree.

Note: All dtsi files which are included in the dts file should be added to this config. For example, the `zynqmp-test.dts` file includes the `custom.dtsi` and `main.dtsi` files andyou must add all three of them separated by spaces to this config.

```
${PROOT}/project-spec/dts_dir/zynqmp-test.dts ${PROOT}/project-spec/
dts_dir/custom.dtsi ${PROOT}/project-spec/dts_dir/main.dtsi
```

Kernel Bootargs

The Kernel Bootargs sub-menu allows you to let PetaLinux automatically generate the kernel boot command-line settings in DTS, or pass PetaLinux user defined kernel boot command-line settings. The following are the default bootargs.

```
Microblaze-full -- console=ttyS0,115200 earlyprintk
Microblaze-lite -- console=ttyUL0,115200 earlyprintk
zynq            -- console=ttyPS0,115200 earlyprintk
zynqmp          -- earlycon clk_ignore_unused root=/dev/ram0 rw
versal          -- console=ttyAMA0earlycon=pl011,mmio32,0xFF000000,115200n8
clk_ignore_unused
```

Note: If you want to see kernel panic prints on the console for Zynq UltraScale+ MPSoCs and Versal devices, add `earlycon console=<device>,<baud rate> clk_ignore_unused root=/dev/ram rw`. For example, `earlycon console=/dev/ttyPS0,115200 clk_ignore_unused root=/dev/ram rw` is added in the `system_user.dtsi`.

For more information, see kernel documentation.

Device Tree Overlay Configuration for Versal ACAPs, Zynq-7000 Devices, and Zynq UltraScale+ MPSoCs

Select this option to separate pl from base DTB and build the `pl.dtsi` to generate `pl.dtbo`. After creating a PetaLinux project follow the below steps to add overlay support:

1. Go to `cd <proj root directory>`.
2. In the `petalinux-config` command, select **DTG Settings → Device tree overlay**.
3. Run `petalinux-build` to generate the `pl.dtbo` in `images/linux` directory.

FPGA manager overrides all the options. This come into play only when FPGA manager is not selected.

Converting Bitstream from .bit to .bin

1. Create a bif file with the following content:

```
all:
{
        [destination_device = pl] <bitstream in .bit> ( Ex:
systemdesign_1_wrapper.bit )
}
```

2. Run following command:

```
bootgen -image bitstream.bif -arch zynqmp -process_bitstream bin
```

***Note*:** The bit/bin file name should be same as the firmware name specified in `pl.dtsi` (`design_1_wrapper.bit.bin`).

Removing PL from the Device Tree

Select this configuration option to skip PL nodes if the user does not depend on the PL IPs. Also, if any PL IP in DTG generates an error then you can simply enable this flag and the DTG will not generate any PL nodes.

1. Go to `cd <proj root directory>`.
2. In the `petalinux-config` command, select **DTG Settings → Remove PL** from device tree.
3. Run `petalinux-build`.

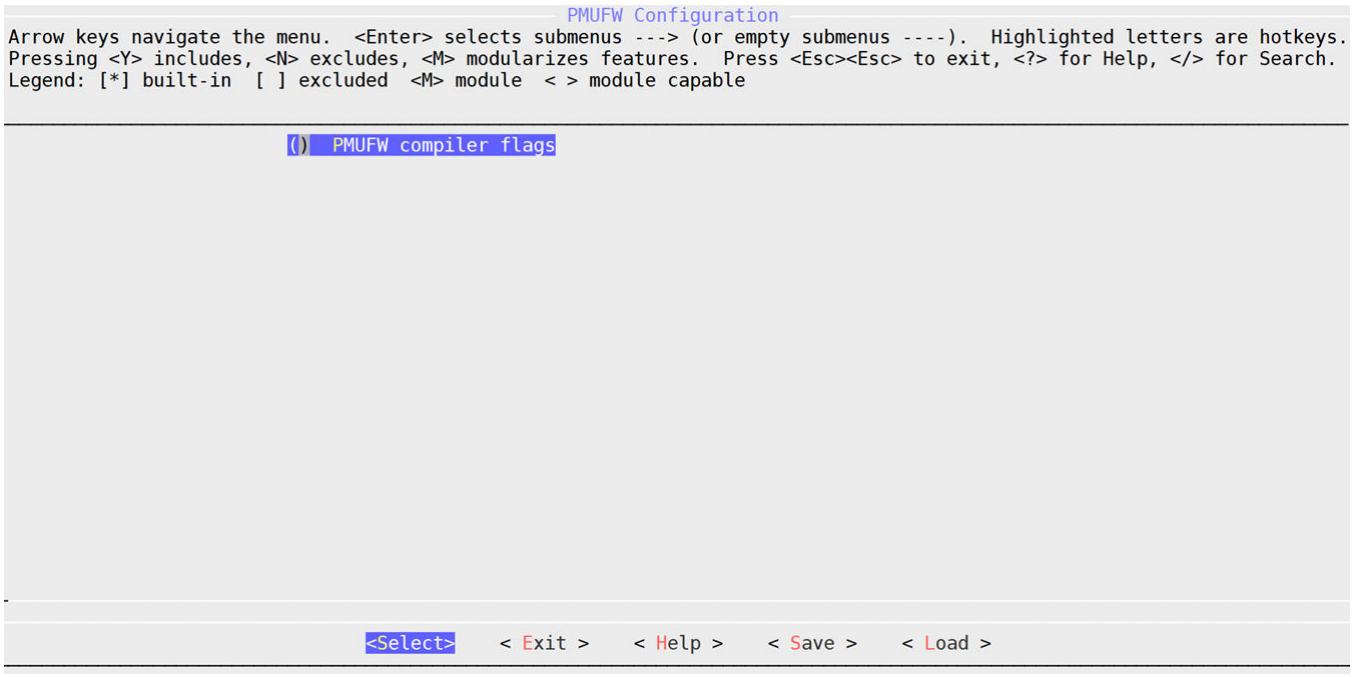
***Note*:** FPGA manager overrides all these options. This come into play only when FPGA manager is not selected.

***Note*:** If you select both the device tree overlay and remove the PL from the device tree, then the base DTB will have an entry for overlay support but there no `pl.dtbo` will be generated.

PMU Firmware Configuration for Zynq UltraScale+ MPSoC

You can provide compiler flags for the PMU firmware application. For example: `-DENABLE_IPI_CRC` to enable CRC check on IPI messages.

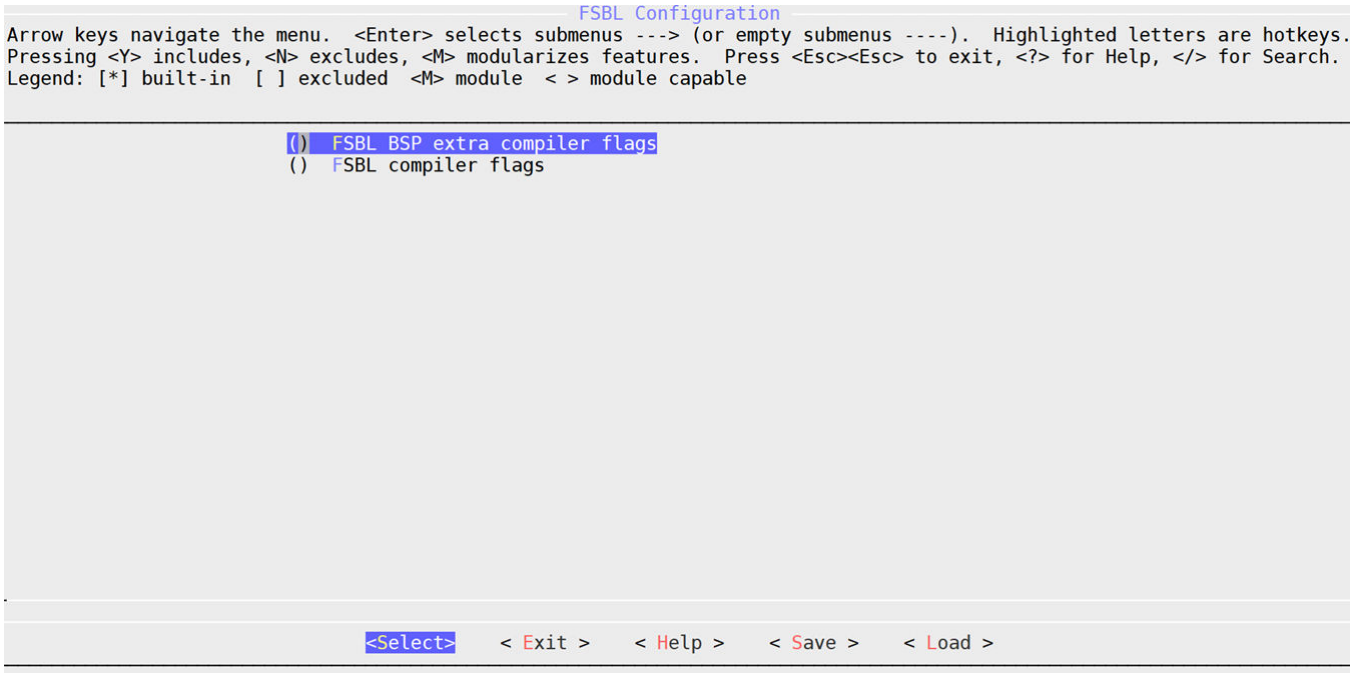
Figure 8: **PMU Firmware Configuration**



FSBL Configuration for Zynq UltraScale+ MPSoC

- **FSBL BSP extra compiler flags:** You can put multiple settings there, separated with space. For example: `-DENABLE_IPI_CRC` for enabling CRC check on IPI messages.
- **FSBL compilation Settings:** You can put multiple settings there, separated with space. For example: `-DFSBL_PROT_BYPASS`.

Figure 9: **FSBL Configuration**

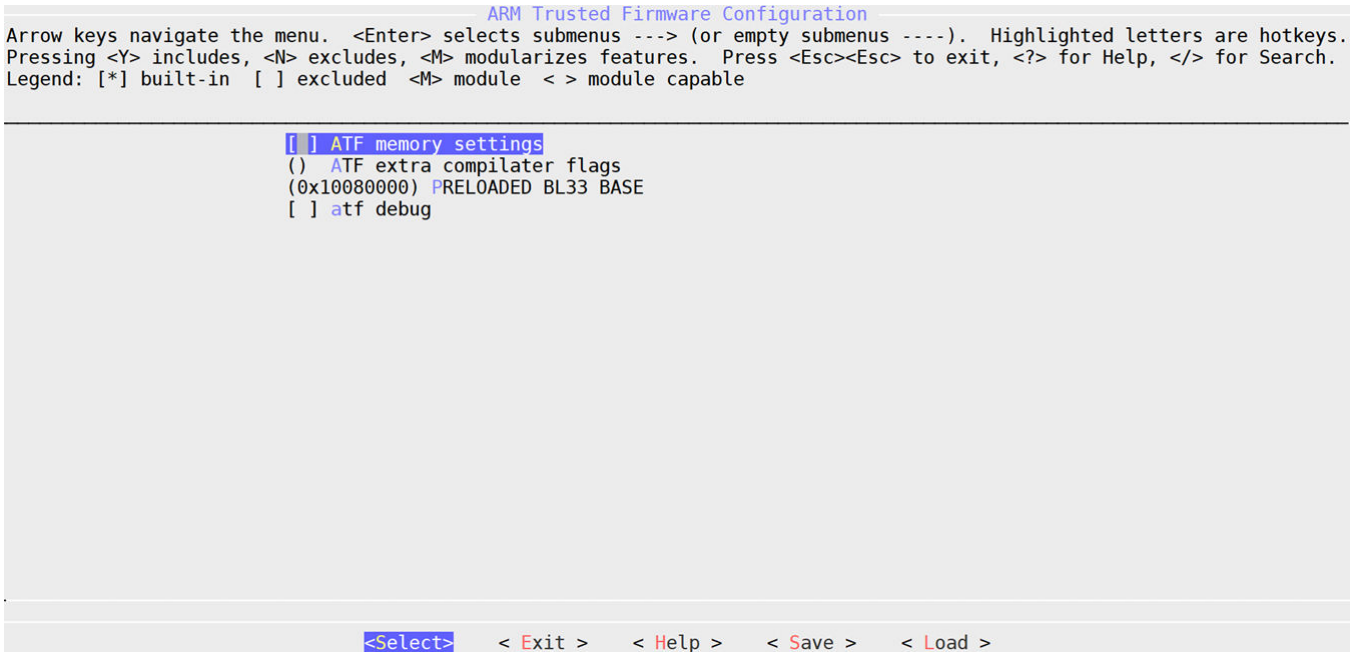


Arm Trusted Firmware Configuration for Zynq UltraScale+ MPSoC and Versal ACAP

The ATF Compilation configuration sub-menu allows you to set:

- Extra ATF compilation settings
- Change the base address of bl31 binary
- Change the size of bl31 binary
- Enable debug in ATF.

Figure 10: **Arm Trusted Firmware Configuration**



FPGA Manager Configuration and Usage for Zynq-7000 Devices and Zynq UltraScale+ MPSoC

The FPGA manager provides an interface to Linux for configuring the programmable logic (PL). It packs the dtbos and bitstreams in the `/lib/firmware/xilinx` directory in the root file system.

After creating a PetaLinux project for Zynq UltraScale+ MPSoC, follow these steps to build the FPGA manager support:

1. Go to `cd <plnx-proj-root>`.
2. In the `petalinux-config` command, select **FPGA Manager → [*] Fpga Manager**.

   ***Note*:** The PetaLinux FPGA manager configuration performs the following:

   1. Generates the `pl.dtsi` nodes as a dt overlay (dtbo).
   2. Packs the dtbos and bitstreams in the `.bin` format into the `/lib/firmware/xilinx/base` directory in the root file system.
   3. The `BOOT.BIN` generated using `petalinux-package` command does not have bitstream.
3. Specify extra hardware files directory path in **FPGA Manager → Specify hardware directory path**.

Note: This step is optional. It is required only if multiple bitstreams for same PS and the corresponding dtbos have to be packed into the root file system. It generates and packs the bitstream in .bin form and its dtbo in the RootFS in the `/lib/firmware/xilinx/<XSA name>` folder. Ensure that the PS design is the same for XSA in the hardware directory path and `<plnx-proj-root>/project-spec/hw-description/system<.xsa>`.

a. You can customize the DT nodes defined in the hardware file provided in the FPGA manager hardware path. For example, if you have `can.xsa` in the hardware directory path, customize the nodes by creating a `can.dtsi` file and add DT nodes/properties in the same hardware directory path.

4. You can also use the `petalinux-create` command to add the PL xsa files into the PetaLinux project. The following command will create the `fpgamanager_dtg` app with the xsa file to generate the dtsi and bit files.

```
petalinux-create -t apps --template fpgamanager_dtg -n can-interface --
srcuri <path-to-xsa>/system.xsa --enable
INFO: Create apps: can-interface
INFO: Copying source urls into the project directory
INFO: New apps successfully created in <plnx-proj-root>/project-spec/
meta-user/recipes-apps/can-interface
INFO: Enabling created component...
INFO: Sourcing build environment
INFO: Silentconfig rootfs
INFO: can-interface has been enabled
```

Note: For each XSA, create a separate app using the previous command. FPGA manager should be enabled in `petalinux-config` for both `fpgamanager` and `fpgamanager_dtg` template apps.

5. Run `petalinux-build`.

Example loading full bitstream on target:

```
root@xilinx-zcu102-2020_1:~# fpgautil -o /lib/firmware/xilinx/base/pl.dtbo -
b
/lib/firmware/xilinx/base/design_1_wrapper.bit.bin

Time taken to load DTBO is 239.000000 milli seconds. DTBO loaded through
ZynqMP FPGA manager successfully.
```

See the petalinux-package command for generating BOOT.BIN.

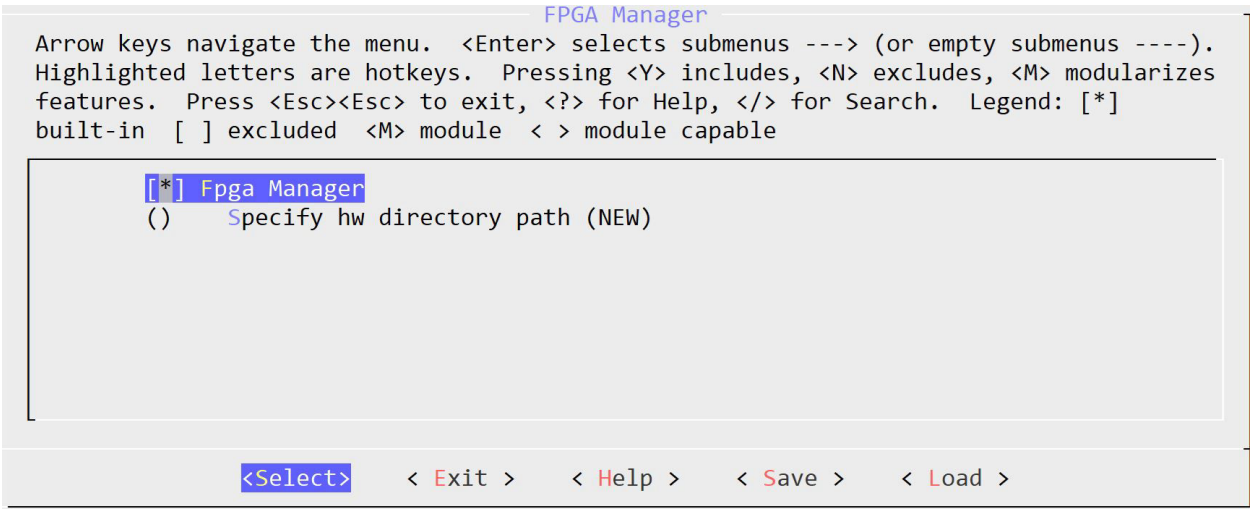
Loading a full bitstream through sysfs – loading bitstream only:

```
root@xilinx-zcu102-2020_1:~# fpgautil  -b /mnt/design_1_wrapper.bit.bin

Time taken to load BIN is 213.000000 milli seconds. BIN FILE loaded through
zynqMP FPGA manager successfully.
```

See help section for more option: `root@xilinx-zcu102-2020_1:~# fpgautil -h`. For more information, see http://www.wiki.xilinx.com/Solution+ZynqMP+PL+Programming.

Figure 11: **FPGA Manager**



Adding Custom dtsi and bit Files to the FPGA Manager for Zynq-7000 Devices and Zynq UltraScale+ MPSoCs

This section provides information on the mechanism and infrastructure required to work with readily (hand-stitched) available dtsi files instead of relying on the XSA to generate them when the FPGA manager is enabled. This generates the dtbo and bin files and copies them into the `rootfs /lib/firmware/xilinx` directory.

1. Create the FPGA manager template:

```
$ petalinux-create -t apps --template fpgamanager -n can-interface --
enable
            INFO: Create apps: can-interface
            INFO: New apps successfully created in <plnx-proj-root>/
project-spec/meta-user/recipes-apps/can-interface
            INFO: Enabling created component...
            INFO: sourcing build environment
            INFO: silentconfig rootfs
            INFO: can-interface has been enabled
```

2. Replace the default files with your own dtsi files. You can generate dtsi files using the device tree generator.

```
$ cp can.dtsi can.bit project-spec/meta-user/recipes-apps/can-interface/
files/
```

3. You can provide the dtsi and bit files when creating the fpgamanager template application. Use te following command to create the application and copy the provided .dtsi and .bit files to the application files directory.

```
petalinux-create -t apps --template fpgamanager -n can-interface --
srcuri "<path-to-dtsi>/pl.dtsi <path-to-bitfile>/system.bit" --enable
INFO: Create apps: can-interface
INFO: Copying source urls into the project directory
INFO: New apps successfully created in <plnx-proj-root>/project-spec/
```

```
meta-user/recipes-apps/can-interface1
INFO: Enabling created component...
INFO: Sourcing build environment
INFO: Silentconfig rootfs
INFO: can-interface has been enabled
```

4. Build the application:

```
$ petalinux-build
```

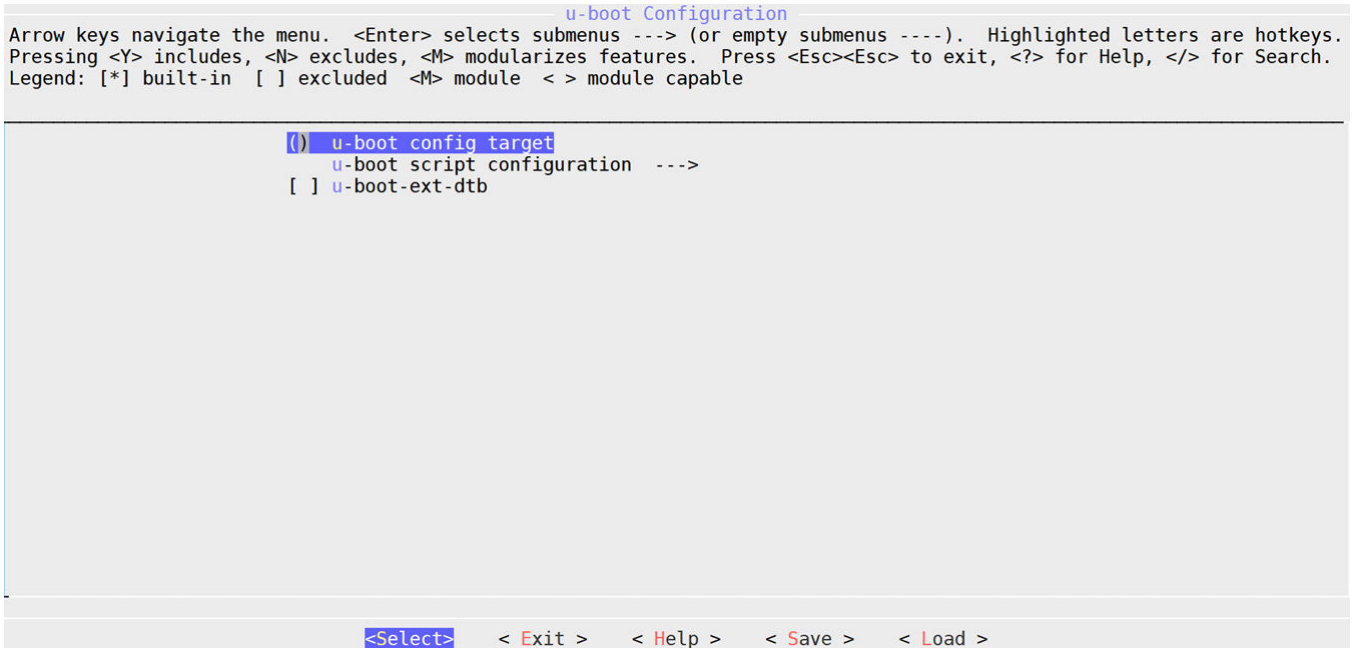
5. Check the target for dtbo and .bin files:

```
$ ls /lib/firmware/xilinx/can-interface/
                pl.dtbo   system.bit.bin
```

Note: You can modify the FPGA manager template recipe at `<project-root-dir>/project-spec/meta-user/recipes-apps/can-interface/can-interface.bb`.

U-Boot Configuration

Figure 12: **U-Boot Configuration**



By default, PetaLinux uses the board configuration "other". If you want configurations from design, select **PetaLinux u-boot config**.

Building a Separate U-Boot DTB

As shown in the following figure, if you enable the `u-boot dtb` configuration, you can build separate U-Boot dtb with the provided design and you can find the binary in `<plnx-proj-root>/images/linux` with the name `u-boot.dtb`. You can change the name with the `uboot dtb package name` option. If you have a dts file to build a separate U-Boot dtb, then you can use the "uboot dts file path" to provide the dts/dtsi files with space separation.

Figure 13: **Separate U-Boot DTB**

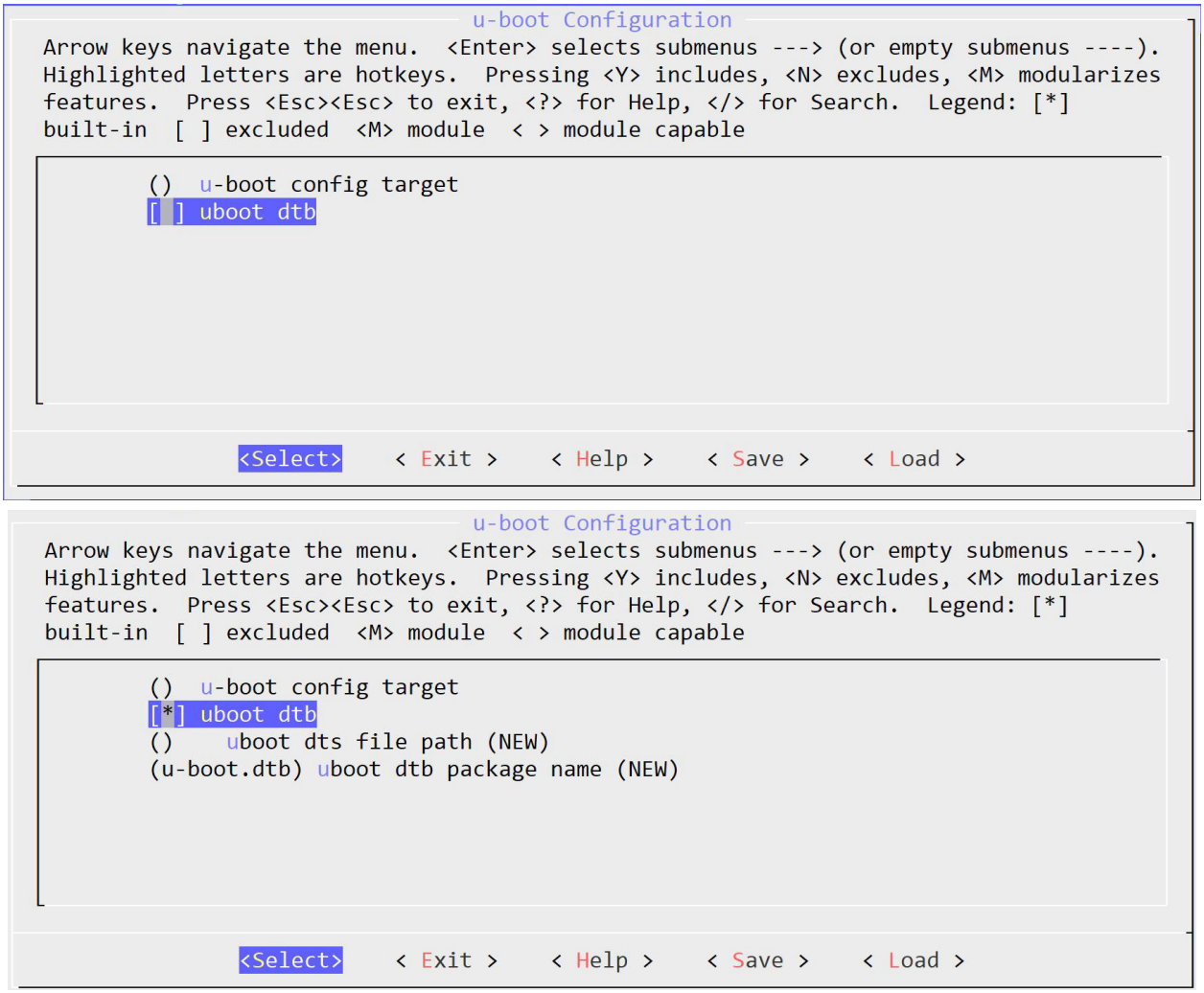


Image Packaging Configuration

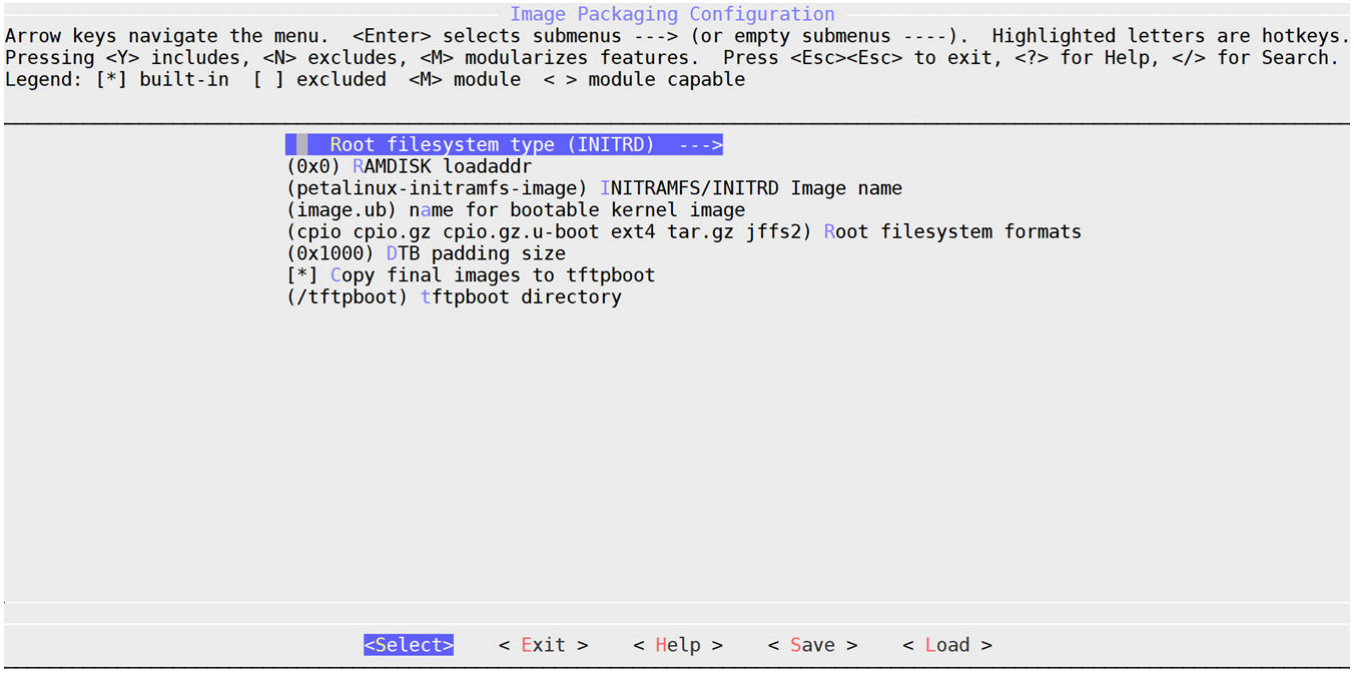
The Image Packaging Configuration sub-menu allows you to set the following image packaging configurations:

- Adding required root file system types.

- File name of the generated bootable kernel image.
- Linux kernel image hash function.
- DTB padding size.
- Whether to copy the bootable images to host TFTP server directory.

TIP: *The `petalinux-build` tool always generates a FIT image as the kernel image.*

Figure 14: **Image Packaging Configuration**



***Note*:** You can add extra spaces in the root file system by adding value to this variable`<project>/project-spec/meta-user/conf/petalinuxbsp.conf` IMAGE_ROOTFS_EXTRA_SPACE.

Meta-user Layer Changes

In earlier releases, `u-boot-zynq-scr.bbappend` and bootcmd files were available in meta-user layers. In the 2021.1 release, they are removed from the meta-user layer and moved to Yocto eSDKs at `<plnx-proj-root>/components/yocto/layers/meta-xilinx/meta-xilinx-bsp/recipes-bsp/u-boot/u-boot-zynq-scr`. You can find these files on running the `petalinux-config/petalinux-build` command.

Added `buildtools-extended` as an option in `petalinux-config`. Enable this option to use gcc and other binaries as part of the PetaLinux tool. If you are using the newer Yocto version, Gatesgarth, to build the project, the gcc version should be higher than version 6 which is not available by default in most of the supported operating systems. Enable this option to unblock this issue.

The auto login option in rootfs config is enabled by default. You do not need to enter your user name as root and password as root after booting Linux.

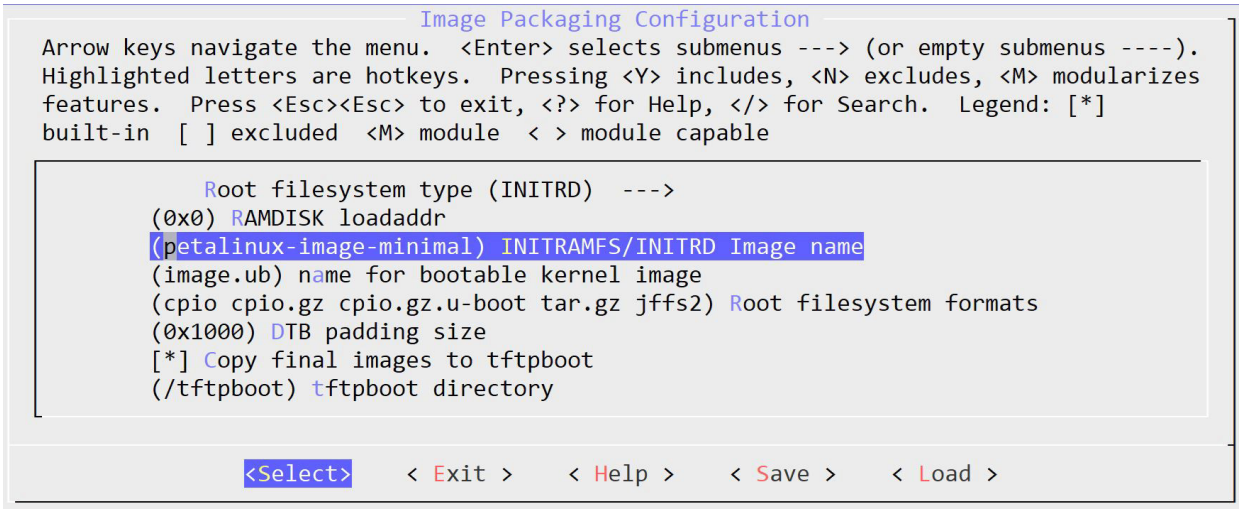
Linux Components Selection

Added Image selector menuconfig option. It is disabled by default.

Option to Change RAM-based Filesystem

There is a menu config option to build the initial RAM-based root filesystem as highlighted in the following figure.

Figure 15: **RAM-based Root File System**



You can provide the name of the Yocto image recipe that is used to build an initial RAM filesystem in the menu config option, INITRAMFS/INITRD Image name. By default, it is set to `petalinux-image-minimal` for Zynq-7000 devices and MicroBlaze processors and to `petalinux-initramfs-image` for Zynq UltraScale+ MPSoCs and Versal ACAPs. The `petalinux-initramfs-image` is a small file system which is deployed into `images/linux` as `ramdisk.cpio.gz.u-boot`. It is also packed into Fit image as `image.ub`. This allows the specified image to be bundled inside the kernel image. An INITRAMFS/INITRD image provides a temporary root file system used for early system initialization, for example, when loading modules that are needed to locate and mount the "real" root filesystem.

After configuration, you can find the following bootargs changes in `<plnx-proj-root>/configs/config`:

```
init_fatal_sh=1:
Launch the initramfs prompt if the specified device not available.
```

You can see the two root file systems in `<plnx-proj-root>/images/linux`:

```
ramdisk.cpio, ramdisk.cpio.gz.u-boot etc..
rootfs.cpio, rootfs.cpio.gz.u-boot etc..
```

Using the `petalinux-initramfs-image` as the INITRD/INITRAMFS image will boot the system with tiny rootfs, search for the ext2, ext3, and ext4 SD partitions, and check for the /dev,/etc and `/sbin/init file/` folder in it to validate whether the rootfs is valid and can be used as a real file system using the command `switch_root`. If you want to specify the SD device which has the ext rootfs, add `ext4=/dev/mmcblk0p2:/rootfs` in system bootargs as shown below:

```
petalinux-config ---> DTG Settings  ---> Kernel Bootargs  ---> [ ] generate
boot args automatically
```

```
petalinux-config ---> DTG Settings  ---> Kernel Bootargs  ---> (earlycon
console=ttyPS0,115200 clk_ignore_unused init_fatal_sh=1 ext4=/dev/
mmcblk0p2:/rootfs) user set kernel bootargs
```

Disabling switch_root

If you do not want to use the `switch_root` command which is the default command to mount and use the ext rootfs in Zynq UltraScale+ MPSoC and Versal devices, you can disable it in build time using the following command:

```
petalinux-config ---> Image Packaging Configuration  --->
```

Update the name `petalinux-initramfs-image` to `petalinux-image-minimal` in INITRAMFS/INITRD image name and then run the `petalinux-build` command.

Note: If you have previously built using the `petalinux-initramfs-image` command, you will continue to see the `ramdisk.*` images in the `images/linux` folder. To avoid confusion, check the time stamp.

Note: If you enable Xen when `switch_root` is enabled, you will see build failures because Xen only supports the ramfs boot. Enabling `switch_root` will enable the ext4-based boot. To resolve the issue, change the above configuration to `petalinux-image-minimal` from `petalinux-initramfs-image`.

Firmware Version Configuration

The Firmware Version Configuration sub-menu allows you to set the firmware version information:

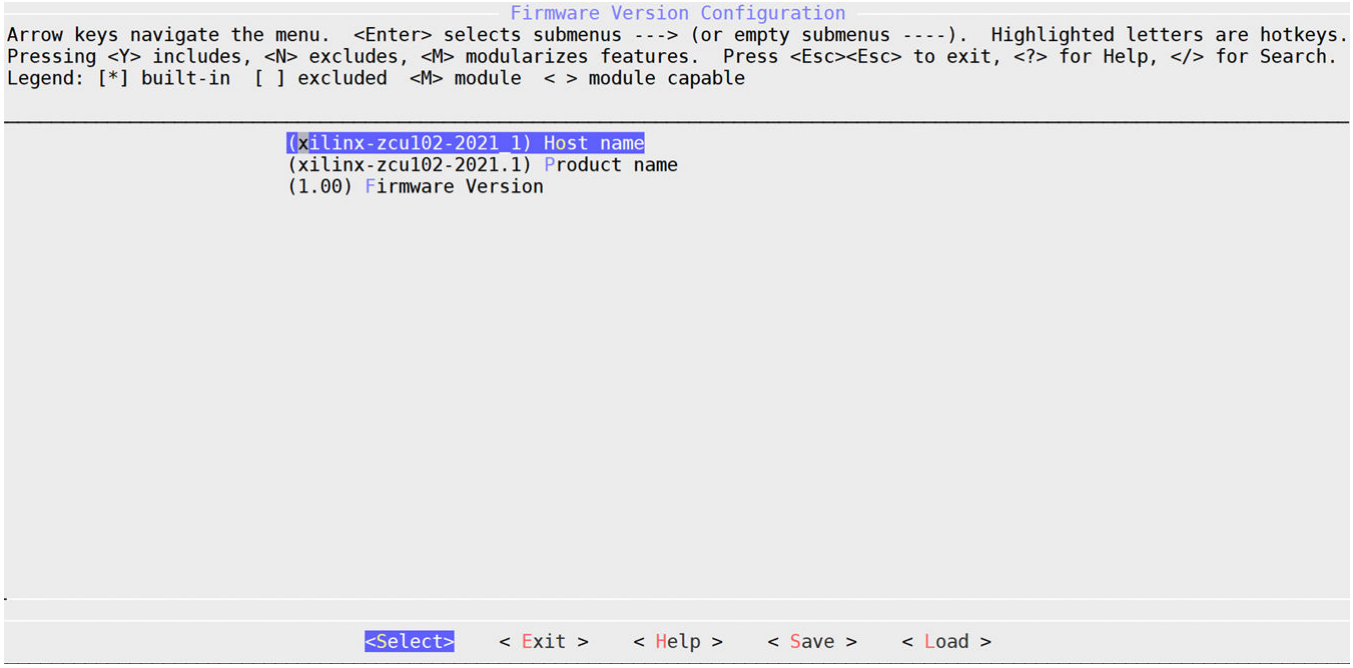
Table 26: **Firmware Version Options**

Firmware Version Option	File in the Target RootFS
Host name	`/etc/hostname`
Product name	`/etc/petalinux/product`

Table 26: **Firmware Version Options** *(cont'd)*

Firmware Version Option	File in the Target RootFS
Firmware Version	`/etc/petalinux/version`

Figure 16: **Firmware Version Configuration**



TIP: *The host name does not get updated. Please see Xilinx Answer 69122 for more details.*

Yocto Settings

Yocto settings allows you to configure various Yocto features available in a project.

Figure 17: **Yocto settings**

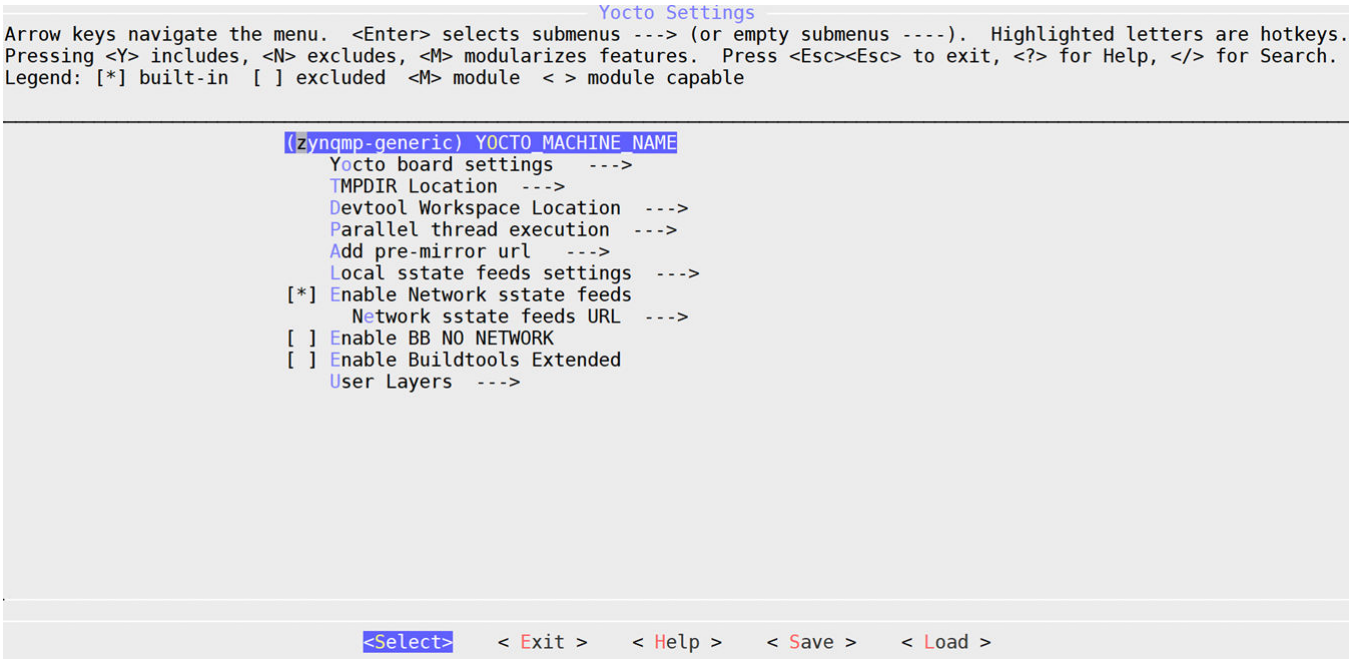
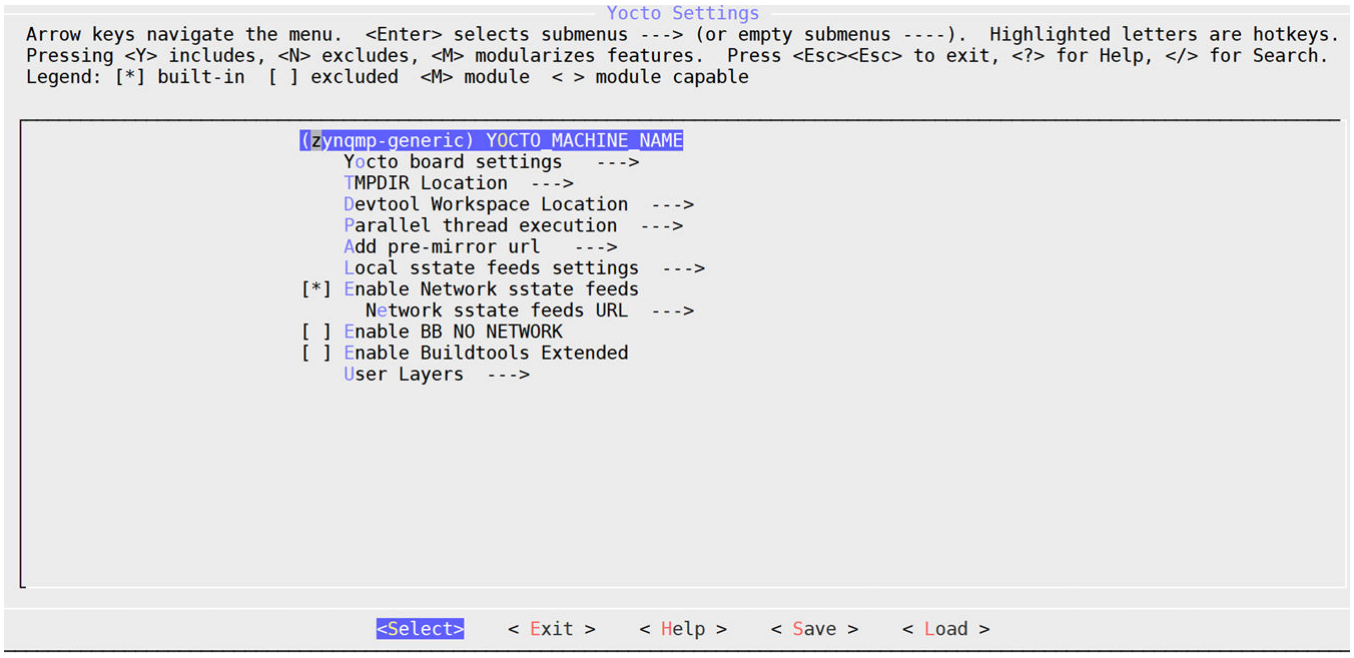


Table 27: **Yocto Settings**

Parameter	Description
TMPDIR Location	This directory is used by BitBake to store logs and build artifacts
YOCTO_MACHINE_NAME	Specifies the Yocto machine name for the project
YOCTO_BOARD_NAME	Specifies the Yocto board name for the project
Parallel thread execution	To limit the number of threads of BitBake instances
Add pre-mirror url	Adds mirror sites for downloading source code of components
Local sstate feeds settings	To use local sstate cache at a specific location
Enable Network sstate feeds	Enabled NW sstate feeds
Enable Buildtools extended	Use this option on a machine that does not meet the minimal gcc, Git, tar, and Python requirements
User layers	Adds user layers into projects
BB_NO_NETWORK	When enabled, internet access is disabled on the build machine

Figure 18: **Yocto Settings**



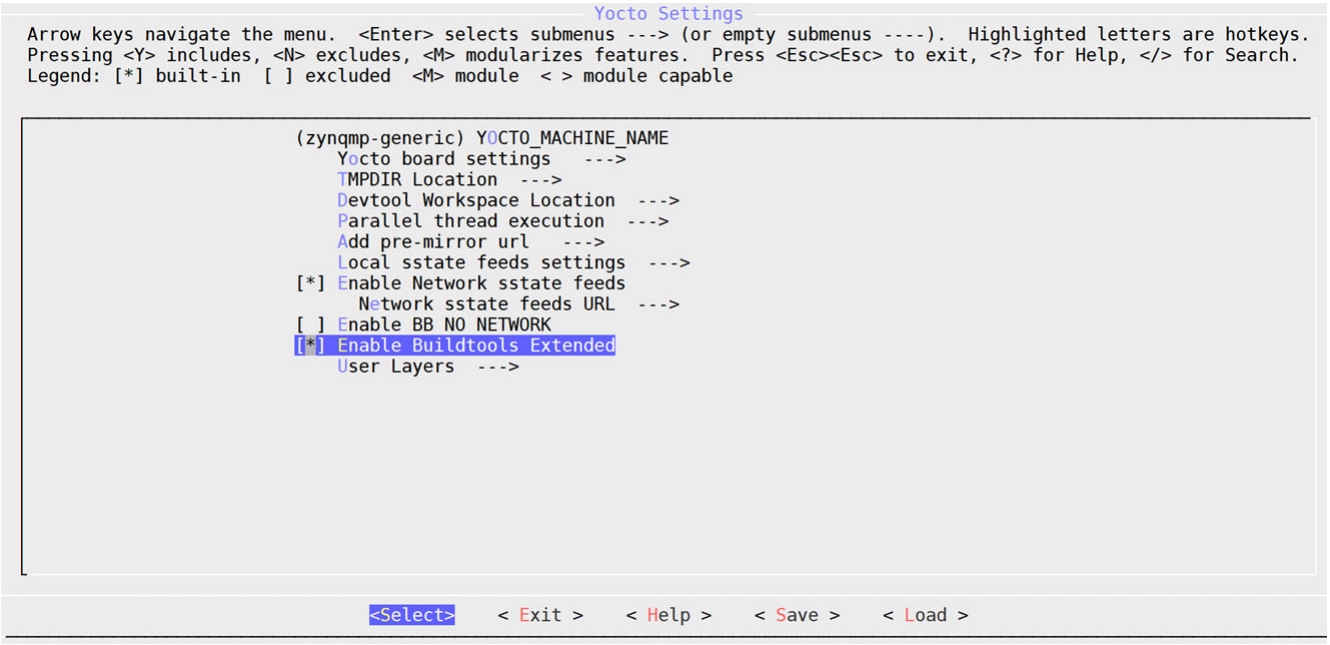
Integrated buildtools-extended into PetaLinux

The latest Yocto SDK version (gatesgarth) requires host development machines to run gcc version 6 or higher. This is currently not available by default in RHEL7.x /CENTOS 7.x/UBUNTU 16.x . If this requirement is not met, the following error is displayed when `petalinux-config` or `petalinux-build` is run:

```
Error: Incompatible SDK installer! Your host gcc version is 4.8 and this
SDK was built by gcc higher version.
```

To fix this issue, a config option is provided to allow you to use the gcc and other binaries/ libraries as a part of the PetaLinux tool (`$PETALINUX/components/yocto/buildtools_extended`). Enable the `buildtools-extended` option in **petalinux-config → Yocto Settings → Enable Buildtools Extended**, as shown in the following figure, to use this option:

Figure 19: **Integrated buildtools-extended**



Open Source Bootgen for On-target Use for Zynq Devices, Versal ACAP, and Zynq UltraScale+ MPSoC

If you want to build an open source bootgen as part of the root file system, follow these steps.

1. Go to the PetaLinux project: `cd <plnx-proj-root>`
2. Run `petalinux-config -c rootfs` and select **Filesystem Packages → Bootgen**
3. Run `petalinux-build`.

   Once the target is up, you can find the bootgen binary in `/usr/bin`.

Configuring Out-of-tree Build

PetaLinux has the ability to automatically download up-to-date kernel/U-Boot source code from a git repository. This section describes how this features works and how it can be used in system-level menu config. It describes two ways of doing the out-of-tree builds.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. For more information, see Importing Hardware Configuration.
- Internet connection with `git` access is available.

Steps to Configure Out-of-tree Build

Use the following steps to configure `UBOOT/Kernel` out-of-tree build.

1. Change into the root directory of your PetaLinux project.

   ```
   $ cd <plnx-proj-root>
   ```

2. Launch the top level system configuration menu.

   ```
   $ petalinux-config
   ```

3. Select **Linux Components Selection** sub-menu.
   - For kernel, select **linux-kernel () → remote**.

     ( ) linux-xlnx

     (X) remote

     ( ) ext-local-src
   - For U-Boot, select **u-boot () → remote**.

     ( ) u-boot-xlnx

     (X) remote

     ( ) ext-local-src

4. For kernel, select **Remote linux-kernel settings → Remote linux-kernel git URL**, and enter git URL for Linux kernel.

   For example: To use https://github.com/Xilinx/linux-xlnx, enter:

   ```
   git://github.com/Xilinx/linux-xlnx.git;protocol=https
   ```

   For U-Boot, select **Remote U-Boot settings → Remote u-boot git URL** and enter git URL for U-Boot. For example:

   ```
   git://github.com/Xilinx/u-boot-xlnx.git;protocol=https
   ```

   Once a remote git link is provided, you must provide any of the following values for "git TAG/ Commit ID" selection, otherwise an error message is expected.

You have to set any of the following values to this setting, otherwise an error message appears.

- To point to HEAD of repository of the currently checked out branch:

```
${AUTOREV}
```

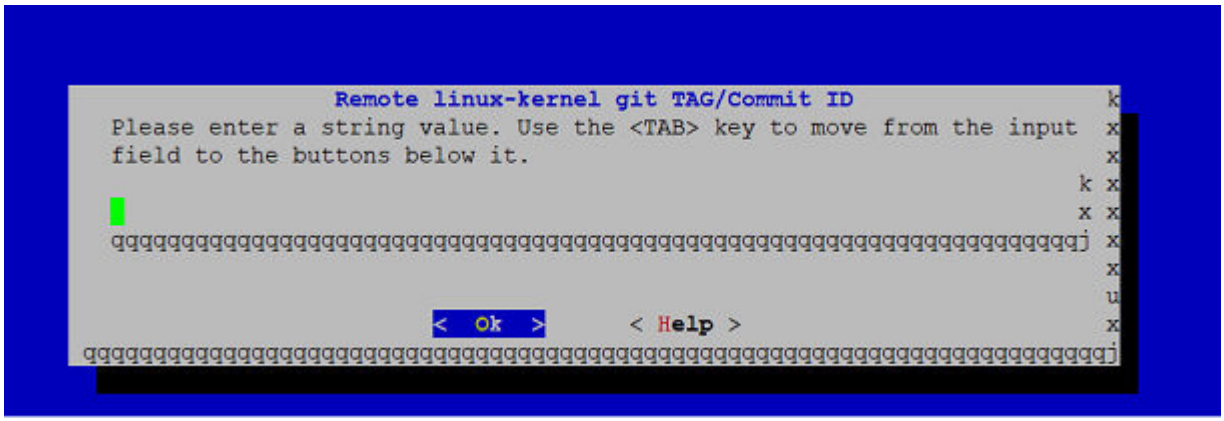
- To point to any tag:

```
tag/mytag
```

- To point to any commit id:

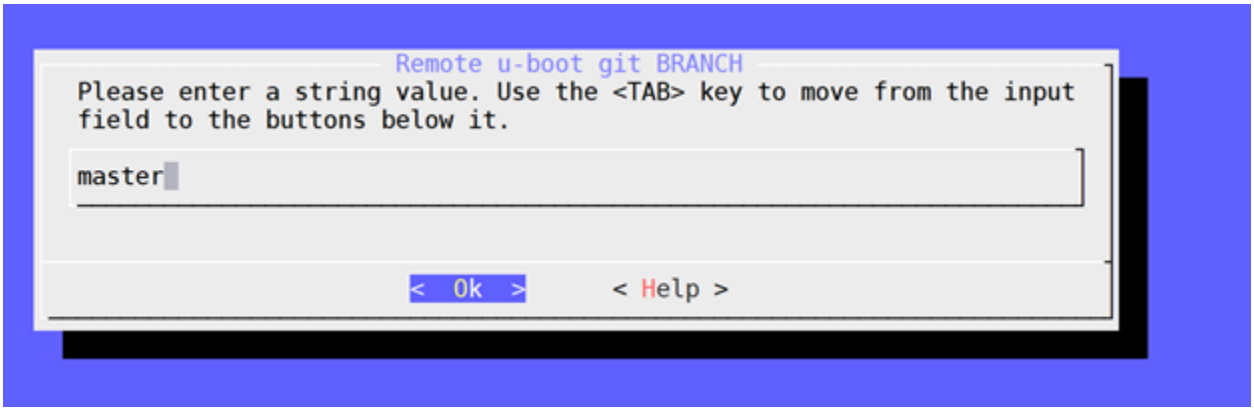
```
commit id sha key
```

Once you select git Tag/Commit ID, you can see a prompt to enter a string value as shown in the following figure. Enter any of the above set values.



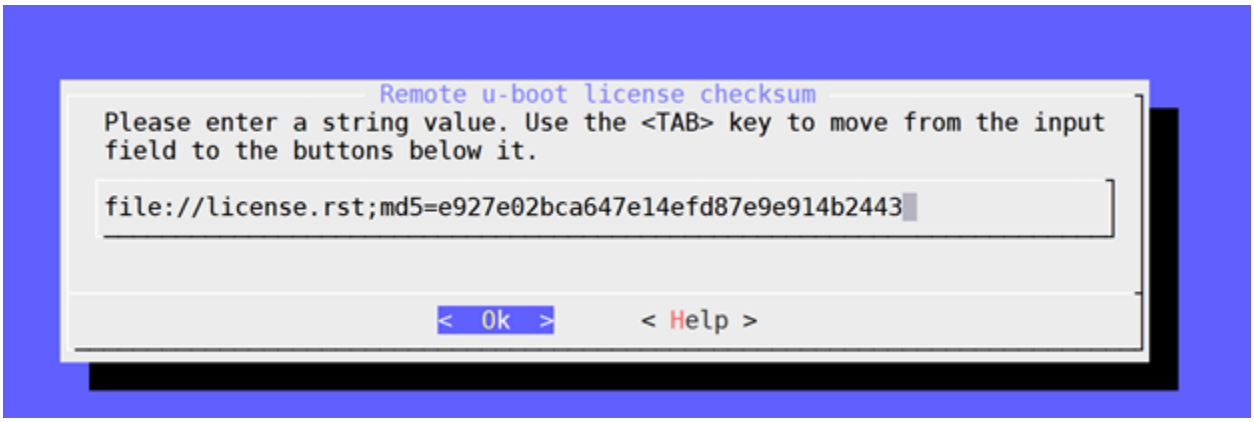
5. To specify BRANCH to `kernel/u-boot/arm-trusted-firmware`, select **Remote settings (Optional)**.

   For example: To specify the master branch, type `master`as shown in the following figure:



6. To specify LICENSE checksum to `kernel/u-boot/arm-trusted-firmware`, select **Remote settings (Optional)**.

For example: To specify `file://license.rst;md5=e927e02bca647e14efd87e9e914b2443`, enter the string value as shown in the following figure:



7. Exit the menu, and save your settings.

Using External Kernel and U-Boot with PetaLinux

PetaLinux includes kernel source and U-Boot source. However, you can build your own kernel and U-Boot with PetaLinux.

PetaLinux supports local sources for kernel, U-Boot, and ATF.

For external sources create a directory `<plnx-proj-root>/components/ext_sources/`.

1. Copy the kernel source directory:

```
<plnx-proj-root>/components/ext_sources/<MY-KERNEL>
```

2. Copy the U-Boot source directory:

```
<plnx-proj-root>/components/ext_sources/<MY-U-BOOT>
```

3. Run `petalinux-config`, and go into Linux Components Selection sub-menu.
   - For kernel, select **linux-kernel () --->** and then select **ext-local-src**.

     ( ) linux-xlnx

     ( ) remote

     (X) ext-local-src
   - For U-Boot, select **u-boot () --->** and then select **ext-local-src**.

     ( ) u-boot-xlnx

     ( ) remote

     (X) ext-local-src

4. Add external source path.
   - For kernel, select **External linux-kernel local source settings --->**. Enter the path:

     `${PROOT}/components/ext_sources/<MY-KERNEL>`
   - For U-Boot, select **External u-boot local source settings --->**. Enter the path:

     `${PROOT}/components/ext_sources/<MY-U-BOOT>`

     ${PROOT} is a PetaLinux variable pointing to `<plnx-proj-root>/` directory. You can also specify an absolute path of the source. The sources can be placed outside the project as well.

     Note: If after setting `ext-local-src`, you try to change it to `linux-xlnx/u-boot-xlnx` in `petalinux-config`, it will give the following warning.

     ```
     WARNING: Workspace already setup to use from <ext-local-src path>,
     Use 'petalinux-devtool reset linux-xlnx' To remove this (or) Use this
     for your development.
     ```

   Note: When creating a BSP with external sources in project, it is your responsibility to copy the sources into the project and do the packing. For more information, see BSP Packaging.

> **IMPORTANT!** *It is not mandatory to have external sources under `components/`. You can specify any location outside the project as well. However, while packaging the BSP, you are responsible for copying the external sources into `components/` and setting relative path.*

Note: If the external source is a git repository, its checked out state must be appropriate for the project that is being built.

Troubleshooting

This section describes some common issues you may experience while configuring out-of-tree build.

Table 28: **Configuring Out-of-Tree Build Troubleshooting**

Problem / Error Message	Description

`fatal: The remote end hung up unexpectedly`
`ERROR: Failed to get linux-kernel`
	Problem Description:
	This error message indicates that system is unable to download the source code (`Kernel/UBOOT`) using remote git URL and hence can not proceed with `petalinux-build`.
	Solution:
	Check whether entered remote git URL is proper or not.
	If above solution does not solve the problem, cleanup the build with the following command:
	`$ petalinux-build -x mrproper`
	Above command will remove following directories.
	`<plnx-proj-root>/images/`
	`<plnx-proj-root>/build/`
	Re-build the system image. For more information, see the Build System Image.

Configuring Project Components

If you want to perform advanced PetaLinux project configuration such as enabling Linux kernel options or modifying flash partitions, use the `petalinux-config` tool with the appropriate `-c COMPONENT` option.

IMPORTANT! *Only Xilinx® drivers or optimizations in the Linux kernel configuration are supported by Xilinx technical support. For more information on Xilinx drivers for Linux, see https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841873/Linux+Drivers.*

The examples below demonstrate how to use `petalinux-config` to review or modify your PetaLinux project configuration.

1. Change into the root directory of your PetaLinux project.

```
$ cd <plnx-proj-root>
```

2. Launch the top level system configuration menu and configure it to meet your requirements:

```
$ petalinux-config
```

3. Launch the Linux kernel configuration menu and configure it to meet your requirements:

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

4. Launch the root file system configuration menu and configure it to meet your requirements:

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

5. Use `--silentconfig` for the components when you do not have Kconfig/Menuconfig support or to skip the launching of configuration menu

```
$ petalinux-config -c <COMPONENT> --silentconfig
```

Warning Message for petalinux-config or petalinux-build Commands

The following warning message appears when you run the `petalinux-config` or `petalinux-build` for components (Ex: `petalinux-build -c u-boot`) and this can be ignored.

```
WARNING: SRC_URI is conditionally overridden in this recipe, thus several
devtool-override-* branches have been created, one for each override that
makes changes to SRC_URI. It is recommended that you make changes to the
devtool branch first, then checkout and rebase each devtool-override-*
branch and update any unique patches there (duplicates on those branches
will be ignored by devtool finish/update-recipe).
```

TIP: *Set U-Boot target in `petalinux-config` menuconfig as required, for your custom board. Set **$ petalinux-config → U-Boot Configuration → u-boot config target** as required. Possible values for Xilinx evaluation boards which are default set are as follows:*

- For Zynq-7000 devices, xilinx_zynq_virt_defconfig
- For Zynq UltraScale+ MPSoC, xilinx_zynqmp_virt_defconfig
- For MicroBlaze processors, microblaze-generic_defconfig
- For Versal devices, xilinx_versal_virt_defconfig

***Note*:** Please make sure board and user specific dtsi entries are added to `project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi`.

Using template flow, for zcu102 and zcu106 boards, add the following line to `<plnx-proj-root>/project-spec/meta-user/recipes-bsp/embeddedsw/fsbl-firmware_%.bbappend` for FSBL initializations.

```
YAML_COMPILER_FLAGS_append = " -DXPS_BOARD_ZCU102" #for zcu102
YAML_COMPILER_FLAGS_append = " -DXPS_BOARD_ZCU106" # for zcu106
```

Device Tree Configuration

This section describes which files are safe to modify for the device tree configuration and how to add new information into the device tree.

Prerequisites

This section assumes that you have PetaLinux tools software platform ready for building a Linux system customized to your hardware platform. For more information, see Importing Hardware Configuration. Knowledge of DTS syntax is required to customize the default DTS.

Configuring Device Tree

User-modifiable PetaLinux device tree configuration is associated with following config files, that are located at `<plnx-proj-root>/project-spec/meta-user/recipes-bsp/device-tree/files/`:

- `system-user.dtsi`
- `xen.dtsi`
- `pl-custom.dtsi`
- `openamp.dtsi`
- `xen-qemu.dtsi`

The generated files are in the `<plnx-proj-root>/components/plnx_workspace/device-tree/device-tree/` directory.

CAUTION! *These dtsi files are auto-generated. Do not edit these files*

For more details on device tree files, see Appendix B: PetaLinux Project Structure.

If you wish to add information, like the Ethernet PHY information, this should be included in the `system-user.dtsi` file. In this case, device tree should include the information relevant for your specific platform as information (here, Ethernet PHY information) is board level and board specific.

The `system-user.dtsi` is automatically created when you configure your PetaLinux project. Once created, the tools do not update it automatically.

***Note*:** The need for this manual interaction is because some information is "board level" and the tools do not have a way of predicting what should be here. Refer to the Linux kernel Device Tree bindings documents (`Documentation/devicetree/bindings` from the root of the kernel source) for the details of bindings of each device.

An example of a well-formed device tree node for the `system-user.dtsi` is shown below:

```
/dts-v1/;
/include/ "system-conf.dtsi"
/ {
};
&gem0 {
    phy-handle = <&phy0>;
    ps7_ethernet_0_mdio: mdio {
        phy0: phy@7 {
            compatible = "marvell,88e1116r";
            device_type = "ethernet-phy";
            reg = <7>;
        };
    };
};
```

IMPORTANT! *Ensure that the device tree node name, MDIO address, and compatible strings correspond to the naming conventions used in your specific system.*

The following example demonstrates adding the `sample-user-1.dtsi` file:

1. Add `/include/ "system-user-1.dtsi"` in `project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi`. The file should look like the following:

   ```
   /include/ "system-conf.dtsi"
   /include/ "system-user-1.dtsi"
   / {
   };
   ```

2. Add `file://system-user-1.dtsi` to `project-spec/meta-user/recipes-bsp/device-tree/device-tree.bbappend`. The file should look like this:

   ```
   FILESEXTRAPATHS_prepend := "${THISDIR}/files:"

   SRC_URI += "file://system-user.dtsi"
   SRC_URI += "file://system-user-1.dtsi"
   ```

   It is not recommended to change anything in `<plnx-proj-root>/components/plnx_workspace/device-tree/device-tree/`.

It is recommended to use system user DTSIs for adding, modifying and deleting nodes or values. System user DTSIs are added at the end, which makes the values in it at higher priority.

You can overwrite any existing value in other DTSIs by defining in system user DTSIs.

U-Boot Configuration

This section describes which files are safe to modify for the U-Boot configuration and discusses about the U-Boot `CONFIG_` options/settings.

Prerequisites

This section assumes that you have PetaLinux tools software platform ready for building a Linux system customized to your hardware platform. Refer to section Importing Hardware Configuration for more information.

Configuring U-Boot

Universal boot loader (U-Boot) configuration is usually done using C pre-processor. It defines:

- **Configuration `_OPTIONS_`:** You can select the configuration options. They have names beginning with `CONFIG_`.
- **Configuration `_SETTINGS_`:** These depend on the hardware and other factors. They have names beginning with `CONFIG_SYS_`.

TIP: *Detailed explanation on `CONFIG_` options/settings documentation and README on U-Boot can be found at Denx U-Boot Guide.*

PetaLinux U-Boot configuration is associated with `config.cfg` and `platform-auto.h` configuration files which are located at `<plnx-proj-root>/project-spec/configs/u-boot-xlnx/` and `platform-top.h` located at `<plnx-proj-root>/project-spec/meta-user/recipes-bsp/u-boot/files/`.

***Note*:** `config.cfg` and `platform-auto.h` are generated only when **petalinux-config → Auto Config Settings → u-boot autoconfig** is enabled.

PetaLinux does not automate U-Boot configuration with respect to `CONFIG_` options/settings. You can add these `CONFIG_` options/settings into `platform-top.h` file.

IMPORTANT! *If **petalinux-config → Auto Config Settings → u-boot autoconfig** is disabled, the `platform-top.h` changes are not reflected directly. To resolve this, see Steps for Managing Image Size.*

Steps to add `CONFIG_` option (For example, CONFIG_CMD_MEMTEST) to `platform-top.h`:

- Change into the root directory of your PetaLinux project.

```
$ cd <plnx-proj-root>
```

- Open the file `platform-top.h`

```
$ vi project-spec/meta-user/recipes-bsp/u-boot/files/platform-top.h
```

- If you want to add CONFIG_CMD_MEMTEST option, add the following line to the file. Save the changes.

```
#define CONFIG_CMD_MEMTEST
```

TIP: *Defining CONFIG_CMD_MEMTEST enables the Monitor Command "mtest", which is used for simple RAM test.*

- Build the U-Boot image.

```
$ petalinux-build -c u-boot
```

- Generate `BOOT.BIN` using the following command.

```
$ petalinux-package --boot --fsbl <FSBL image> --fpga <FPGA bitstream> --
u-boot
```

- Boot the image either on hardware or QEMU and stop at U-Boot stage.
- Enter the `mtest` command in the U-Boot console as follows:

```
ZynqMP mtest
```

- Output on the U-Boot console should be similar to the following:

```
Testing 00000000 ... 00001000:
                      Pattern 00000000 Writing... Reading...Iteration:
20369
```

IMPORTANT! *If `CONFIG_CMD_MEMTEST` is not defined, output on U-Boot console is as follows:*

```
U-Boot-PetaLinux> mtest Unknown command 'mtest' - try 'help'
```

For more information on U-Boot, see https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842223/U-boot.

Yocto Features

SDK Generation (Target Sysroot Generation)

The OpenEmbedded build system uses BitBake to generate the Software Development Kit (SDK) installer script standard SDKs. PetaLinux builds and installs SDK. The installed SDK can be used as sysroot for the application development.

Building SDK

The following command builds SDK and copies it at `<proj_root>/images/linux/sdk.sh`.

```
petalinux-build --sdk
```

The following is the equivalent BitBake command.

```
bitbake petalinux-image-minimal -c do_populate_sdk
```

Installing SDK

The generated SDK has to be installed/extracted to a directory. The following command extracts the SDK to a specified directory. The default SDK is `<proj_proot>/images/linux/sdk.sh` and default installation directory is `<proj_proot>/images/linux/sdk/`.

```
petalinux-package --sysroot -s|--sdk <custom sdk path> -d|--dir <custom
directory path>
```

Examples

1. Adding a cross compiling qt toolchain

   To build SDK with qt toolchain:

   a. Create the `<plnx-proj-root>/project-spec/meta-user/recipes-core/images/petalinux-image-minimal.bbappend` file.

   b. Add `inherit populate_sdk_qt5` in the newly created file.

   c. Run `petalinux-config -c rootfs` and select **packagegroup-petalinux-qt**.

d. Run `petalinux-build -s`.

e. Run `petalinux-package --sysroot`.

To verify:

a. Open a new terminal.

b. Go to `<plnx-proj-root>/image/linux/sdk`.

c. Run `source environment-setup-aarch64-xilinx-linux`.

d. Run `which qmake`. This confirms that the qmake is coming from the SDK.

2. Building OpenCV applications

a. Create a PetaLinux project.

b. Add `packagegroup-petalinux-opencv` in the RootFS menu config.

c. Build SDK

```
petalinux-build --sdk
```

This command builds SDK and deploys it at `<plnx-proj-root>/images/linux/sdk.sh`.

d. Install SDK.

```
petalinux-package --sysroot
```

This command installs SDK at `<plnx-proj-root>/images/linux/sdk`.

e. Use the `images/linux/sdk` directory as sysroot for building the OpenCV applications.

Building an eSDK

The following command builds the eSDK and copies it at `<proj_root>/images/linux/esdk.sh`.

```
petalinux-build --esdk
```

Packaging Sources and Licenses

- To pack all the components of `petalinux-build`, issue the following commands.

```
petalinux-build --archiver
```

- To pack only the sysroot components, use the following command.

```
petalinux-build --sdk --archiver
```

Note: You can find the archiver tar in `<plnx-proj-root>/images/linux`.

Accessing BitBake/Devtool in a Project

BitBake is available only in the bash shell.

Steps to Access the BitBake Utility

1. Run `petalinux-config` or `petalinux-config --silentconfig` at least once after creating the project, so that the required environment is setup.
2. Source the PetaLinux tools script:

   ```
   source /opt/pkg/petalinux/settings.sh
   ```

3. Source the Yocto e-SDK:

   ```
   source <plnx-proj-root>/components/yocto/environment-setup-aarch64-xilinx-linux
   ```

4. Source the environment setup script to be redirected to the build directory:

   ```
   source <plnx-proj-root>/components/yocto/layers/core/oe-init-build-env
   ```

   Stay in the build directory to run BitBake.

5. Export XSCT:

   ```
   export PATH=/opt/pkg/petalinux/tools/xsct/bin:$PATH
   ```

6. Parse the PetaLinux variable to recipes:

   ```
   export BB_ENV_EXTRAWHITE="$BB_ENV_EXTRAWHITE PETALINUX"
   ```

7. To test if the BitBake is available, run:

   ```
   bitbake strace
   ```

   The generated images are placed in the deploy directory. You have to copy the generated images into `<plnx-proj-root>/images/linux` directory to work with the other commands.

Steps to Access the Devtool Utility

1. Follow steps 3-7 as described in Steps to Access the BitBake Utility.
2. Create a workspace for devtool:

   ```
   devtool create-workspace ../components/plnx_workspace/
   ```

3. Add the recipe to workspace directory:

   ```
   devtool add --version 1.0 gpio-demo ../project-spec/meta-user/recipes-apps/gpio-demo
   ```

4. Build the recipe:

```
devtool build gpio-demo
```

For more devtool commands, type `devtool --help`.

Shared State Cache

Yocto e-SDK contains minimal shared state (sstate) cache. Xilinx® hosts the full `petalinux-image` sstate cache at http://petalinux.xilinx.com/sswreleases/rel-v2020/.

During `petalinux-build`, BitBake searches for the sstate cache in http://petalinux.xilinx.com/sswreleases/rel-v2020/. If it fails to find the sstate cache, BitBake will build it from scratch. sstate is signature locked.

For a `.bbappend` file which you create for any root file system component, you must add `SIGGEN_UNLOCKED_RECIPES += "<recipe-name>"` or `SIGGEN_UNLOCKED_RECIPES += "u-boot-xlnx"` in `<plnx-proj-root>/project-spec/meta_user/conf/petalinuxbsp.conf`.

Sharing your State Cache

If you want to share/use your previously build sstate cache, you can follow either of the following approaches.

As an optimization, the Yocto Project optimizes downloads of the sstate cache items to only the minimal items required for the current build. This needs to be factored in when sharing your sstate cache with another user. The second user's configuration may be different causing a different set of sstate cache items to be required. There are two approaches to optimizing your downstream user and their usage of the sstate cache. The first approach is that the second user should include both the sstate cache directory you provided as well as the original Xilinx sstate cache directory in `<plnx-proj-root>/build/conf/plnxtool.conf`.

```
SSTATE_MIRRORS = "  \
file://.* file://<your-sstate-cache>/PATH \n \
file://.* http://petalinux.xilinx.com/sswreleases/rel-v2021/aarch64/sstate-cache/PATH;downloadfilename=PATH \n \"
```

The second approach is to fetch all of the sstate cache items that can be required for a particular build. This is required if you want to share your build sstate with the downstream user. There is an option called `--setscene-only` that will fetch all of the sstate objects that might be needed for a particular target recipe. For example, if you used `petalinux-build (bitbake petalinux-image-minimal)`, you should run the following command first to fetch all the required sstate from Xilinx provided sstate.

```
petalinux-build -c "petalinux-image-minimal --setscene-only"(bitbake
petalinux-image-minimal --setsecene-only)
```

Downloading Mirrors

Xilinx® hosts all source download tar files for each release at https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools.html. By default, PetaLinux points to pre-mirrors using`petalinux-config` command.

If any component is rebuilt from scratch, BitBake or devtool searches for its source in pre-mirrors and downloads the mirror URL. Later, it searches in SRC_URI of recipes for downloading the source of that component. If you configure any value through **petalinux-config → yocto settings → premirrors**, it will first search in the configured pre-mirrors, then on petalinux.xilinx.com, and finally in the SRC_URI in recipes.

You can add more mirrors by adding `SOURCE_MIRROR_URL += file:///home/you/your-download-dir/` in `<plnx-proj-root>/project-spec/meta-user/conf/petalinuxbsp.conf`.

For more information on how to set SSTATE and DL_DIR, see How to Reduce Build Time using SSTATE Cache.

Machine Support

The Yocto machine specifies the target device for which the image is built. The variable corresponds to a machine configuration file of the same name, through which machine-specific configurations are set. Currently, PetaLinux supports the user machine configuration file.

You can add your own machine configuration file under `<plnx-proj-root>/project-spec/meta-user/conf/machine/` or you can add your machine configuration file in any additional layers and add it into project through `petalinux-config`.

Follow these steps to specify the user machine configuration file name in the PetaLinux project:

1. Go to the PetaLinux project.

2. Select **petalinux-config → Yocto Settings → () MACHINE NAME**.
3. Specify your machine configuration file name.

   The BSPs are now updated with the meta-xilinx machines.

Table 29: **Machine Name Change for Templates**

Template	Machine
zynq	zynq-generic
zynqmp	zynqmp-generic
microblaze	microblazeel-v11.0-bs-cmp-mh-div-generic
versal	versal-generic

Board Name / Board Variant Support

The Yocto board or board variant variable adds board-specific recipes and board-specific Yocto variables for which the image is built. You can specify the Yocto board or board variant name using the `petalinux-config` command. If the board has multiple variants, use `board_variant config` to differentiate the Yocto flags or Yocto tasks, for example, the DT board name for device tree).

For example, specifying zcu102 board name defines the DT board `dtsi` file for zcu102 board.

Follow these steps to specify the Yocto board name in the PetaLinux project:

1. Go to the PetaLinux project.
2. Select **petalinux-config → Yocto Settings → Yocto Board Settings → () YOCTO_BOARD_NAME**.
3. Select **petalinux-config → Yocto Settings → Yocto Board Settings → () YOCTO_BOARD_VARIANT_NAME**.
4. Specify your Yocto board/Board variant variable in this config. By default, it is set to NULL

The BSPs are now updated with the board names

Table 30: **BSPs and Board Names**

BSP	Board Name	Yocto Board Variant Name
ZCU102	zcu102	NULL
ZCU104	zcu104	NULL
ZCU106	zcu106	NULL
ZCU111	zcu111	NULL
ZCU1275	zcu1275	NULL
ZCU1285	zcu1285	NULL
ZCU216	zcu216	NULL

Table 30: **BSPs and Board Names** *(cont'd)*

BSP	Board Name	Yocto Board Variant Name
ZCU208	zcu208	NULL
ZCU208-SDFEC	zcu208	NULL
ZC702	zc702	NULL
ZC706	zc706	NULL
AC701	ac701	NULL
KC705	kc705	NULL
KCU105	kcu105	NULL
VCU118	vcu118	NULL
SP701	sp701	NULL
VCK190	vck190	NULL
VMK180	vmk180	NULL
vc-p-a2197-00-reva-x-prc-02-reva	vc-p-a2197-00	NULL

SoC Variant Support

Xilinx® delivers multiple devices for each SoC product. Zynq® UltraScale+™ MPSoC is shipped in three device variants. For more information see here. Zynq-7000 devices are shipped in two variants. For more information, see here.

SOC_VARIANT extends overrides with ${SOC_FAMILY}${SOC_VARIANT}. It further extends overrides with components on the SoC (for example, Mali™, VCU). This makes reusing the component overrides depending on the SoC. This feature is mainly used to switch to hardware acceleration automatically if the hardware design has the corresponding IP (VCU or USP). Xilinx distributes SoCs with multiple variants as shown below.

1. Zynq-7000 devices are distributed under Zynq7000zs and Zynq7000z. The available SOC_VARIANTs are:
   - "7zs" - Zynq-7000 Single A9 Core

     "7z" - Zynq-7000 Dual A9 Core
   - Default SOC_VARIANT for Zynq-7000 devices is "7z". For 7000zs devices, add the SOC_VARIANT = "7zs" in `petalinuxbsp.conf`

   There are no additional overrides for Zynq-7000 devices.
2. Zynq UltraScale+ MPSoC is shipped in three device variants. The available SOC_VARIANTs are:
   - "cg" - Zynq UltraScale+ MPSoC CG Devices
   - "eg" - Zynq UltraScale+ MPSoC EG Devices

- "ev" - Zynq UltraScale+ MPSoC EV Devices
- "dr" - Zynq UltraScale+ MPSoC RFSoC devices

The default value is "eg". PetaLinux automatically assigns "ev" and "dr" based on the presence of IP in the XSA.

3. Versal ACAP is shipped only in one design variant currently. The SOC_VARIANT is s80.

***Note*:** You have to explicitly set SOC_VARIANT = "cg" in `petalinuxbsp.conf` for "CG" devices.

Image Features

The contents of images generated by the OpenEmbedded build system can be controlled by the IMAGE_FEATURES and EXTRA_IMAGE_FEATURES variables that you typically configure in your image recipes. Through these variables, you can add several different predefined packages such as development utilities or packages with debug information needed to investigate application problems or profile applications.

To remove any default feature, add the following code in the `petalinuxbsp.conf`:

```
IMAGE_FEATURES_remove = "ssh-server-dropbear"
```

To add any new feature, add the following command in the `petalinuxbsp.conf`:

```
IMAGE_FEATURES_append = "  myfeature"
```

Filtering RootFS Packages Based on License

The INCOMPATBLE_LICENSE flag is used to control which packages are included in the final root file system configuration based on the license.

If you want to exclude packages based on license, you can edit the `<plnx-proj-root>/project-spec/conf/petalinuxbsp.conf` file. For example, set INCOMPATBLE_LICENSE = "GPLv3", then run the `petalinux-build` command.

Creating and Adding Patches For Software Components within a PetaLinux Project

To create and add patches for software components within a PetaLinux project, follow these steps:

1. Get the source code from git url specified in meta-layers:

   ```
   petalinux-devtool modify <recipe-name>
   ```

   For example:

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

   The previous command fetches the sources for the recipe and unpack them to a `<plnx-proj-root>/components/yocto/workspace/sources/<recipe-name>` directory and initialize it as a git repository if it is not already one.

2. Make the changes you want to make to the source.
3. Run a build to test your changes. You can just `petalinux-build -c <recipename>` or even build an entire image using `petalinux-build` incorporating the changes assuming a package produced by the recipe is part of an image. There is no need to force anything; the build system will detect changes to the source and recompile as necessary.
4. Optional: Test your changes on the target.
5. Place your changes in the form of a patch to the PetaLinux project. To commit your changes, use the following commands.

   ```
   git add <filename>
   git commit -s
   ```

6. `petalinux-devtool finish <recipe-name> <destination layer path>` creates a patch for the committed changes in recipe sources directory.

   For example:

   ```
   petalinux-devtool finish linux-xlnx <plnx-proj-dir>/project-spec/meta-user
   ```

   `petalinux-devtool update-recipe linux-xlnx -a <destination layer path>` creates a patch for the committed changes in recipe sources directory without removing the bbappend and source directory from the workspace directory.

   For example:

   ```
   petalinux-devtool update-recipe linux-xlnx -a <plnx-proj-dir>/project-spec/meta-user
   ```

7. Once you have finished working on the recipe, run `petalinux-devtool reset <recipe-name>` to remove the source directory for the recipe from workspace.

Known Issues for the Devtool Flow

When using `ext-local-src` for linux-xlnx and executing `petalinux-devtool finish linux-xlnx <layer path>` after making menuconfig changes, the file `bsp.cfg` is deleted from the `<plnx-proj-dir>/project/-spec/meta-user/recipes-kernel/linux/linux-xlnx` directory. However, the `<plnx-proj-dir>/project/-spec/meta-user/recipes-kernel/linux/linux-xlnx_%.bbappend` file still references this file in the 'KERNEL_FEATURES' variable. This causes an error when executing `petalinux-build`/ `petalinux-build -c kernel`. To overcome this issue, create a empty `bsp.cfg` file in the `<plnx-proj-dir>/project/-spec/meta-user/recipes-kernel/linux/linux-xlnx` directory before building the project/kernel.

Adding Extra Users to the PetaLinux System

You can make the changes using the following steps:

1. Go to project: **petalinux-config -c rootfs → PetaLinux Rootfs Settings → Add extra users**
2. Provide the users. To add extra users to the PetaLinux system, provide the user ID (userid) and password (passwd) separated by `:`; for multiple users, separate sets of user IDs and passwords using `;`.

   Examples:

   To add a passwd1 for user1:

   ```
   user1:passwd1; or user1:passwd1
   ```

   To add an empty passwd for the user1:

   ```
   user1:
   ```

   To add user1 and user2 with passwd1 and passwd2, respectively:

   ```
   user1:passwd1;user2:passwd2;
   ```

   To add an empty passwd for user1 and passwd2 for user2

   ```
   user1:;user2:passwd2
   ```

Adding Auto login Option in the PetaLinux Tool

The auto login option in PetaLinux ensures that you do not need to enter a user name as root and password as root after booting Linux.

If you want to use the user name and password as a root, disable the auto login option using the **petalinux-config -c rootfs → Image Features → [ ] auto-login**.

XILINX

Chapter 12

Technical FAQs

Troubleshooting

This section details the common errors that appear, while working with the PetaLinux commands, and also lists their recovery steps in detail.

For Yocto related information, please see https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842475/PetaLinux+Yocto+Tips.

TMPDIR on NFS

The error displayed is:

```
“ERROR: OE-core's config sanity checker detected a potential
misconfiguration”. Either fix the cause of this error or disable the
checker at your own risk (see sanity.conf). For the list of potential
problems or advisories.
```

The TMPDIR: `/home/user/xilinx-kc705-axi-full-2020.2/build/tmp` cannot be located on NFS.

When TMPDIR is on NFS, BitBake throws an error at the time of parsing. You can change it to local storage while creating the PetaLinux project. To do so, follow these steps:

1. Either run `petalinux-create -t project -s <PATH_TO_PETALINUX_PROJECT_BSP> --tmpdir <TMPDIR PATH>` or `petalinux-config`.
2. Provide any local storage by selecting **Yocto-settings → TMPDIR**.

CAUTION! *Do not configure the same TMPDIR for two different PetaLinux projects. This can cause build errors.*

Recipe Name has ' _ ' or Uppercase Letters or Starts with an Uppercase Letter

If the application name is `plnx_myapp`, BitBake throws an error. A version number has to be entered after ' _ '. For example, `myapp_1` is an accurate application and module name.

To recover, sstateclean the application and then delete it. Also, delete the following line in `<plnx-proj-root>/project-spec/meta-user/conf/user-rootfsconfig`.

```
CONFIG_plnx_myapp
```

If the application or library or module name has all uppercase letters or starting with an uppercase letter MYAPP/Myapp, then BitBake throws a do_package_qa error.

To recover, sstateclean the application and then delete it. Also, delete the line in `<plnx-proj-root>/project-spec/meta-user/conf/user-rootfsconfig`.

> **CAUTION!** *If the project path has special characters like +, *, ! etc., then the `petalinux-config` command fails to execute. For example: /opt/petalinux+/xilinx-zc702-2. To recover, do not use any special characters in the path.*

Recover from Corrupted Terminal

When PetaLinux is exited forcefully by pressing Ctrl+C twice, the following error appears:

```
NOTE: Sending SIGTERM to remaining 1 tasks
Error in atexit._run_exitfuncs:
Traceback (most recent call last):
  File
"<plnx-proj-root>/components/yocto/layers/core/bitbake/lib/bb/ui/k
notty.py", line 313, in finish
    self.termios.tcsetattr(fd, self.termios.TCSADRAIN, self.stdinbackup)
termios.error: (5, 'Input/output error')
```

After this error, the console is broken and you cannot see the text that you typed. To restore the console, enter `stty sane` and press Ctrl+J twice.

Python Language Settings

The following errors appear when the language settings are missing:

- `Could not find the /log/cooker/plnx_microblaze in the /tmp directory during petalinux-config`
- `Please use a locale setting which supports UTF-8 (such as LANG=en_US.UTF-8).`

Python cannot change the file system locale after loading. Therefore, you need a UTF-8 when Python starts, else it will not work.

```
ERROR: Failed to build project
```

To resolve the above errors, set the following:

```
export LC_ALL=en_US.UTF-8
export LANG=en_US.UTF-8
export LANGUAGE=en_US.UTF-8
```

Menuconfig Hang for Kernel and U-Boot

For `petalinux-config -c`, sometimes when the kernel and U-Boot BitBake try to open a new terminal inside, they fail. The following are the possible error messages:

1. `ERROR: Unable to spawn new terminal`
2. `ERROR: Continuing the execution without opening the terminal`

The solutions can be:

1. Use `ssh -X <hostname>`.
2. Uncomment the OE_TERMINAL line in `<plnx-proj-root>/project-spec/meta-user/conf/petalinuxbsp.conf`. You can set any terminal which suits you (possibles values could be auto, screen, tmux, xterm, and konsole). You have to change the OE_TERMINAL as it cannot get through default. For this, you must have the corresponding utility installed in your PC.

Menuconfig Not Seen for Kernel and U-Boot

Set `SHELL=/bin/bash` before issuing `petalinux-config -c kernel/ petalinux-config -c u-boot`.

Menuconfig Corruption for Kernel and U-Boot

When you issue `petalinux-config -c kernel/u-boot`, you might see a corrupted menu config. This is due to the terminal you are using.

To resolve, export other terminals like screen, xterm, konsole, putty, or gnome, and retry. For example:

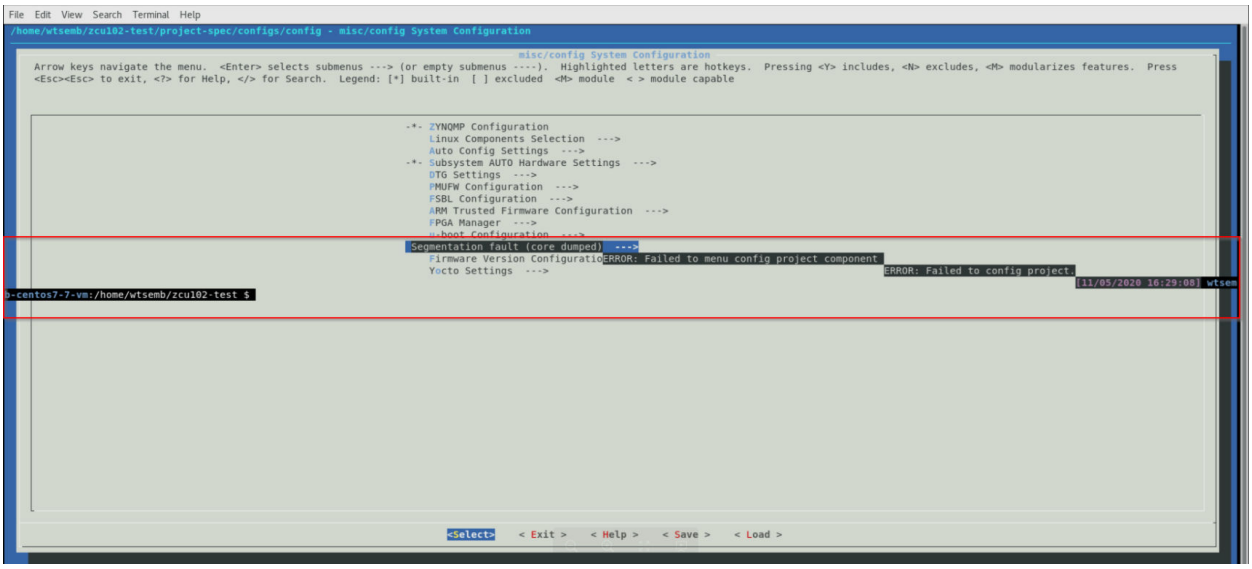
```
export TERM=screen
```

Note: You will not see this issue on Ubuntu 18.x host machines.

petalinux-config Menu Crashes with Segmentation Fault on CentOS 7.x

The `petalinux-config` menu crashes with segmentation fault on CentOS 7.x as shown below. This is a known issue in 2020.x version of PetaLinux tools.

Figure 20: **Segementation Fault on CentOS 7.x**




```
$ petalinux-config INFO: Sourcing build tools awk: /lib64/libm.so.6:
version `GLIBC_2.29' not found (required by awk) awk: /lib64/libm.so.6:
version `GLIBC_2.29' not found (required by awk) awk: /lib64/libm.so.6:
version `GLIBC_2.29' not found (required by awk) awk: /lib64/libm.so.6:
version `GLIBC_2.29' not found (required by awk) [INFO] Generating Kconfig
for project awk: /lib64/libm.so.6: version `GLIBC_2.29' not found (required
by awk) awk: /lib64/libm.so.6: version `GLIBC_2.29' not found (required by
awk) awk: /lib64/libm.so.6: version `GLIBC_2.29' not found (required by
awk) [INFO] Menuconfig project
```

This issue is seen when you run `petalinux-config` with non-interactive mode.

```
$ petalinux-config --silentconfig
INFO: sourcing build tools
awk: /lib64/libm.so.6: version `GLIBC_2.29' not found (required by awk)
awk: /lib64/libm.so.6: version `GLIBC_2.29' not found (required by awk)
awk: /lib64/libm.so.6: version `GLIBC_2.29' not found (required by awk)
awk: /lib64/libm.so.6: version `GLIBC_2.29' not found (required by awk)
[INFO] silentconfig project
awk: /lib64/libm.so.6: version `GLIBC_2.29' not found (required by awk)
ERROR: Failed to silentconfig project component
ERROR: Failed to config project.
$
```

Root Cause

In CentOS-7.x if you set LD_LIBRARY_PATH variable (example `export LD_LIBRARY_PATH=/lib64:/usr/lib64:/usr/local/lib64`) causes the `petalinux-config` menu to crash with segmentation fault. In Yocto, if you want to use any build host binaries or libraries you need to unset LD_LIBRARY_PATH. Once it goes to segmentation fault terminal, follow these steps to kill the seg fault terminal.

1. Type `reset` and enter key. Now terminal will be blank.
2. Press **Ctrl+J** same time.

   Now you should have a working terminal.

Solution

To work around this issue, follow these steps.

1. Unset the LD_LIBRARY_PATH by running below command.

   ```
   $ unset LD_LIBRARY_PATH
   ```

2. Source the PetaLinux tools.

   ```
   $ source /opt/xilinx/petalinux/petalinux-v2020.1/settings.sh
   ```

3. Run menu config.

   ```
   $ petalinux-config
   ```

External Source Configurations

The cfg or scc files are not applied with external source in the Yocto flow (upstream behavior). PetaLinux needs to handle external source with configurations applied. Therefore, it is always recommended to use cfgs instead of sccs.

Xen and openamp are handled through distro features. Adding distro features does not enable their corresponding configurations in kernel as they are handled in scc file. The solution is to edit `<plnx-proj-root>/project-spec/meta-user/recipes-kernel/linux/linux-xlnx_%.bbappend`.

Add the following lines:

```
SRC_URI += "file://xilinx-kmeta/bsp/xilinx/xen.cfg"
```

To work with the scc files, replace their respective cfg files using external source methodology.

do_image_cpio: Function Failed

CPIO format does not support sizes greater than 2 GB. Therefore, you cannot use INITRAMFS for larger sizes. The following steps describes the process for larger image sizes (greater than 2 GB).

1. Change the root file system type to EXT4 (SD/eMMC/SATA/USB).

   ```
   $ petalinux-config
   ```

   Select **Image Packaging Configuration → Root filesystem type → EXT4 (SD/eMMC/SATA/USB)**.

2. Add the following lines in the `<plnx-proj-root>/project-spec/meta-user/conf/petalinuxbsp.conf`.

   ```
   IMAGE_FSTYPES_remove = "cpio cpio.gz cpio.bz2 cpio.xz cpio.lzma cpio.lz4
   cpio.gz.u-boot"
   IMAGE_FSTYPES_DEBUGFS_remove = "cpio cpio.gz cpio.bz2 cpio.xz cpio.lzma
   cpio.lz4
   cpio.gz.u-boot"
   ```

3. Build the project.

   ```
   $ petalinux-build
   ```

   ***Note*:** Unlike earlier, currently PetaLinux does not generate the global DTS file. Use the following command to generate the global DTS file:

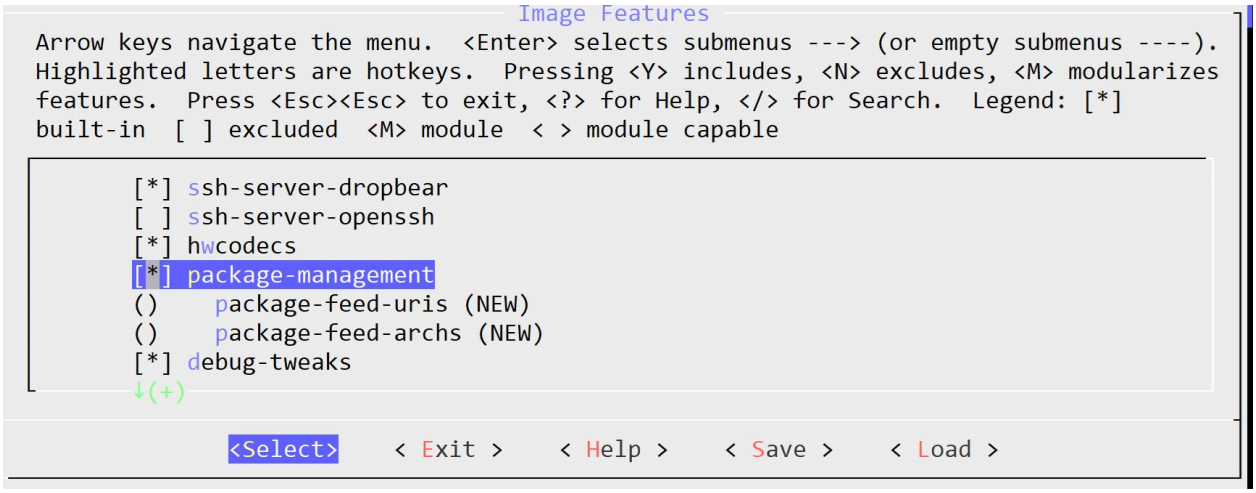
   ```
   dtc -I dtb -O dts -o system.dts system.dtb
   ```

CAUTION! *Do not use the symlinked path to the project directories for any build operations, including simply "cd"ing into the directory.*

Package Management

PetaLinux supports package management system for Zynq-7000, Zynq UltraScale+ MPSoC, and Versal™ devices. Use the following steps to configure and use the package management system:

Figure 21: **Package Management**



1. Enable DNF through `petalinux-config -c rootfs`. Enable the following configs to use DNF.
   - **Image Features → [*] package management**
   - Set the base package feed url in **Image features → package-management → package-feed-uris**.

     For BSPs, http://petalinux.xilinx.com/sswreleases/rel-v2021.1/generic/rpm/.

     Check the available architecture in the above URLs.
   - Set the package feed architecture in **Image features → package management → package-feed-archs**.

     ***Note*:** From 2021.1, specifying package-feed-archs is optional and PetaLinux or Yocto set the archs based on the project.

     The possible archs are below:

     - zcu102
       - noarch
       - aarch64
       - zynqmp
       - zynqmp_generic
       - zynqmpeg
     - zcu104 and zcu106
       - noarch
       - aarch64
       - zynqmp

    - zynqmp_generic
    - zynqmpev
  - zcu111, zcu 208, and RFSoC BSPs
    - noarch
    - aarch64
    - zynqmp
    - zynqmp_generic
    - zynqmpdr
  - vck190
    - noarch
    - aarch64
    - versal
    - versal_generic
    - versal-ai-core
  - vmk180
    - noarch
    - aarch64
    - versal
    - versal_generic
    - versal-prime
  - zc702/zc706/ac701
    - cortexa9hf_neon
    - cortexa9t2hf_neon
    - zynq
    - zynq7z
    - zynq_generic
2. Build the project.

   ```
   #petalinux-build
   ```

3. Boot Linux in SD or in JTAG boot mode.

4. Check for `.repo` file on target in `/etc/yum.repos.d/` as shown below.

```
[oe-remote-repo-sswreleases-rel-v2021-generic-rpm-noarch]
name=OE Remote Repo: sswreleases rel-v<PETALINUX_VER> generic rpm noarch
baseurl=http://petalinux.xilinx.com/sswreleases/rel-v<PETALINUX_VER>/
generic/rpm/noarch
gpgcheck=0

[oe-remote-repo-sswreleases-rel-v<PETALINUX_VER>-generic-rpm-aarch64]
name=OE Remote Repo: sswreleases rel-v<PETALINUX_VER> generic rpm aarch64
baseurl=http://petalinux.xilinx.com/sswreleases/rel-v<PETALINUX_VER>/
generic/rpm/aarch64
gpgcheck=0

[oe-remote-repo-sswreleases-rel-v<PETALINUX_VER>-generic-rpm-zynqmp]
name=OE Remote Repo: sswreleases rel-v<PETALINUX_VER> generic rpm zynqmp
baseurl=http://petalinux.xilinx.com/sswreleases/rel-v<PETALINUX_VER>/
generic/rpm/zynqmp
gpgcheck=0

[oe-remote-repo-sswreleases-rel-v<PETALINUX_VER>-generic-rpm-zynqmpeg]
name=OE Remote Repo: sswreleases rel-v<PETALINUX_VER> generic rpm
zynqmpeg
baseurl=http://petalinux.xilinx.com/sswreleases/rel-v<PETALINUX_VER>/
generic/rpm/zynqmpeg
gpgcheck=0

[oe-remote-repo-sswreleases-rel-v<PETALINUX_VER>-generic-rpm-
zynqmp_generic]
name=OE Remote Repo: sswreleases rel-v<PETALINUX_VER> generic rpm
zynqmp_generic
baseurl=http://petalinux.xilinx.com/sswreleases/rel-v2021/generic/rpm/
zynqmp_generic
gpgcheck=0
```

5. List all available packages.

```
#dnf repoquery
```

6. Install a specific package.

```
#dnf install <pkg name>
```

Example: `#dnf install packagegroup-petalinux-matchbox`

Once the matchbox package is installed, reboot the target and you should get the desktop environment.

Common Tar Usage

Get the `BOOT.BIN` from the Vitis software platform or the PetaLinux BSP prebuilt to boot Linux with common tar images. You may see runtime errors when hardware is not supported for the utilities/packages enabled in the common rootFS with respect to each platform. Example cases are as follows:

1. An error occurs when the AI Engine is accessed on the VMK180 board using the common rootFS because the VMK180 board does not support AI Engine.
2. An error occurs when the VCU is accessed on the ZCU102 board using the common rootFS because the VCU is not supported on the ZCU102. Use the rootFS utilities/packages based on the hardware supported on the board.

Linux Boot Hang with Large INITRAMFS Image in Zynq-7000 Devices and Zynq UltraScale+ MPSoC

When the `petalinux-boot` command is issued, the following warning message is displayed:

```
"Linux image size is large (${imgsize}). It can cause boot issues. Please
refer to Technical FAQs. Storage based RootFilesystem is recommended for
large images."
```

If your INITRAMFS image size is large, use storage based boot.

Generating Image for Flash/QSPI Boot in MicroBlaze Devices

1. Create the project using the BSP.
2. Go to the project directory using the `cd` command.
3. Run the `petalinux-config --silentconfig` command.
4. Open the `platform-auto.h` file and add the following command in the `bootcmd` options of the U-Boot environment (`project-spec/configs/u-boot-xlnx/platform-auto.h`). The values can vary based on hardware.

   ```
   $ "bootcmd=sf probe 0;sf read 0x82000000 0xAA0000 0xC00000;bootm
   0x82000000\0"
   ```

5. `$save` and exit.
6. Run the `petalinux-build` command.
7. Create the mcs file with build images using the following command.

   ```
   $ petalinux-package --boot --fpga images/linux/system.bit --u-boot --
   kernel --flash-size 128
   ```

Detailed Component-based Logs

The `<plnx-proj-root>/build/build.log` log provides information on all the tasks that are running and the tasks that have failed.

For each component and for each task, there are detailed logs that you can find in the build or in the TMPDIR(NFS builds) directory.

- If you build on local space, you can find the component task wise logs in the build directory. For example: For the device-tree component, do_configure task, the log is located at `<plnx-proj-root>/build/tmp/work/versal_generic-xilinx-linux/device-tree/xilinx-v2020.2+gitAUTOINC+f725aaecff-r0/temp/log.do_configure.32730`.
- If you build on the NFS, you can find the log in `<TMPDIR>/work/versal_generic-xilinx-linux/device-tree/xilinx-<PETALINUX_VERSION>+gitAUTOINC+f725aaecff-r0/temp/log.do_configure.32730`. You can check the TMPDIR location in the `<plnx-proj-root>/project-spec/configs/config` file.

XILINX

Appendix A

Migration

This section describes the migration details of the current release versus the previous release.

Deprecating microblaze_lite Designs

microblaze_lite designs are deprecated in future releases.

1. Create a project using the template flow:

```
petalinux-create -t project -n <name> --template microblaze
```

2. Run the following command:

```
petalinux-config --get-hw-description <PATH-TO-XSA DIRECTORY>/--get-hw-description=<PATH-TO-XSA DIRECTORY>/--get-hw-description=<PATH-TO-XSA>
```

Note: Importing the XSA design as microblaze_lite in the <PATH-TO-XSA directory> gives the following warning: *WARNING: microblaze_lite design support will be deprecated in next releases warning for all subsequent executions of the `petalinux-config`/`petalinux-build` commands. This means that your XSA file has microblaze_lite design.*

U-Boot Image Changes

From 2021.1, PetaLinux removed DTB from `u-boot.elf`. U-Boot now uses the DTB from `BOOT.BIN` instead of `u-boot.elf`. If you are using a custom BIF file to generate `BOOT.BIN`, ensure to include the DTB in the file so that the boot doesn't fail.

Table 31: **U-Boot Files Deployed by Petalinux**

File name	Description
`u-boot.elf`	Self-extractable U-Boot elf containing the U-Boot binary symbols with a larger size than previous versions
`u-boot-dtb.elf`	Self-extractable U-Boot elf that has the U-Boot binary with DTB
`u-boot.bin`	Contains the U-Boot binary
`u-boot-dtb.bin`	Contains the U-Boot binary with DTB

By default, PetaLinux uses `u-boot.elf` to create `JTAG/BOOT.BIN`. During `BOOT.BIN` creation, bootgen removes the symbols from `u-boot.elf` so that there is no size difference in the final `BOOT.BIN`. You can also use `u-boot-dtb.elf` for `BOOT.BIN`. In that case, U-Boot uses the DTB from `u-boot-dtb.elf` instead of `BOOT.BIN`.

Common Tar Files for Platforms

Common tar files are provided for Zynq®, Zynq® UltraScale+™ MPSoC, and Versal™ platforms. Each common TAR file contains a prebuilt Linux kernel and a root file system and can be used the respective Zynq, Zynq UltraScale+ MPSoC, and Versal boards for embedded developers.

Copy these files to an SD flash card along with the platform specific boot image (BOOT.BIN) available from the Vitis™ IDE or from the board-specific PetaLinux BSP prebuilts. For more information, see the readme file inside the common tar.

***Note*:** Installing PetaLinux tools is not necessary to use the common images.

Removed Kernel and U-Boot Autoconfig

petalinux-config → Auto configs → kernel autoconfig and **petalinux-config → Auto configs → u-boot autoconfig** have been removed for aarch64 and arm. If you want to add any U-Boot or kernel configs, use `bsp.cfg`.

Removed petalinux-build -b Option

Yocto does not recommend using the `-b` option for the `petalinux-build` command as it cannot guarantee that all dependencies have been built or that they are up to date. Support for the `-b` option has been removed from PetaLinux.

Using Bitbake over Devtool for petalinux-config -c

Using the devtool backend for `petalinux-config` causes delay in menuconfig and failures in applying local patches as seen in the 2020.x releases. For 2021.1 and future releases, use the bitbake backend for the `petalinux-config`. This is similar to 2019.1 and earlier releases.

Added Distroboot Support for MicroBlaze Processors

Distroboot support for MicroBlaze processors has been added. Be careful when loading the `boot.scr` file into the DDR/Flash as per the bootmode. You can check the images offsets in the `boot.scr`.

Note: Use the `cat` command to check the `boot.scr` file. Using the `vi`, `vim`, and `gvim` commands to check might corrupt the file causing the boot to fail.

U-Boot Configuration Changes

The following changes were made:

- Added a new menuconfig option to check/modify the default bootscript images offsets or images names at **petalinux-config → u-boot Configuration → u-boot script configuration**. The modified offsets are added into the final `boot.scr` that is generated using `petalinux-build` command.
- Added a new menuconfig, u-boot-ext-dtb, to build a separate dtb for U-Boot. You can build a U-Boot dtb from design or by specifying the dts file.
- Changed **petalinux-config → u-boot configuration → u-boot config**.

Added Linux Configuration

Added a new menu config, **petalinux-configuration → Linux configuration**, to specify the Linux defconfig.

Removed the Advanced Bootable Images Storage Settings

Until 2019.x, **petalinux-config → Subsystem Auto Hardware Settings → Advanced bootable image storage settings** configuration was used to pass the image information to U-Boot through the `platform-auto.h` file that is used in the U-Boot boot commands. To align with Yocto from 202x.x, this image information will no longer be passed to the U-Boot and using the u-boot static env for the boot commands. The previous configuration has been removed and a new menuconfig has been introduced. From this release onwards, you can pass the image names or load address to load the U-Boot using `boot.scr` through **petalinux-config → u-boot configuration → u-boot configuration → u-boot script configuration**.

Changes to petalinux-boot Command

The following `petalinux-boot -qemu` / `--jtag` command line options which are broken in past releases have been fixed in this release.

- `petalinux-boot -qemu/--jtag --kernel/--u-boot`
- `petalinux-boot -qemu/--jtag -dtb`
- `petalinux-boot -qemu/--jtag --kernel/--u-boot -pmufw` for Zynq UltraScale+ MPSoC.
- `petalinux-boot --qemu/--jtag --kernel --rootfs` <specify custom `cpio.gz.u-boot` rootfs path>

***Note*:** `--rootfs` with `petalinux-boot --qemu` does not work for Versal platfroms in 2021.1. Since rootfs will be part of the `qemu_boot.img` file you need to repack the .img file using `petalinux-package --boot --u-boot --qemu-rootfs rootfs.cpio.gz.u-boot`and then run the `petalinux-boot --qemu --kernel` command to using the rootfs.cpio.gz.u-boot.

DTG Settings Changes

The following changes were made to the DTG settings:

- Provide the extra device tree path in **petalinux-config → DTG Settings → Extra dts/dtsi files menu with absolute/full path** to build as part of the device-tree and deploy into `/boot/devicetree`.

- Earlier **petalinux-config → DTG settings → Devicetree flags** used the `CONFIG_SUBSYSTEM_DEVICETREE_FLAGS="-@"` configuration. The new configuration is `CONFIG_SUBSYSTEM_DEVICETREE_COMPILER_FLAGS="-@"`.

Changes to the Ethernet Settings

The default **petalinux-config → Subsystem Auto Hardware Settings → Ethernet Settings → Ethernet MAC address** now points to `ff:ff:ff:ff:ff:ff` invalid address to read the MAC from EEPROM. Earlier, it pointed `00:0a:35:00:22:01`.

Image Package Configuration

The default menuconfig options for the `petalinux-initramfs-image` is **petalinux-config → Image Package Configuration → INITRAMFS/INITRD image name**. The config option is packaged in the `image.ub` file and is also copied into the `images/linux` folder with `ramdisk` prefixed to the filename (for example, `ramdisk.cpio.gz.u-boot`), if the image specified in the config option name contains the initramfs. This is a tiny-based rootfile system search for the EXT partition in SD/eMMC that is used as a real time rootfs file system irrespective of boot mode. If the init script in the tiny rootfs system does not find any rootfs in real time, it lands up into the tiny-based rootfs with the prompt '/#'. For example, if you are in the QSPI boot mode and you have flashed the `BOOT.BIN`, `boot.scr`, and `image.ub` files, it will boot up to tiny rootfs and search for the ext2/3/4 to mount and use it as real rootfs.

The default prebuilt/built images are configured as mentioned above to facilitate the larger rootfs support in PetaLinux.

Use Environment Variables in petalinux-config Option

From this release, PetaLinux supports using all Yocto variables in `petalinux-config`, for example, TOPDIR. To use the shell variables, you need to export the shell variable to BB_ENV_EXTRAWHITE. You cannot use normal shell variables directly to align with Yocto like the previous versions.

Note: In the above example, the HOST machine exports the USER environment variable by default.

```
export BB_ENV_EXTRAWHITE="$BB_ENV_EXTRAWHITE USER"
```

```
export MYENV="myenv"
export BB_ENV_EXTRAWHITE="$BB_ENV_EXTRAWHITE MYENV"
```

Note: In the above example, MYENV is a new environment variable defined and exported to the BB_ENV_EXTRAWHITE Yocto variable. After it is exported to BB_ENV_EXTRAWHITE, you can use MYENV for the `petalinux-config` options.

Auto login Option for PetaLinux Images

The auto login option is enabled by default for all PetaLinux BSPs. You do not need to enter a username or password to log in to the board using PetaLinux images. To disable this option, go to **petalinux-config -c rootfs → Image Features → [ ] auto-login** and disable **auto-login**.

FPGA Manager Changes

When you create fpgamanager or `fpgamanager_dtg` template apps using the `petalinux-create` command, ensure to enable FPGA Manager. Not enabling FPGA Manager could cause issues in loading the dtbo and bin files on target.

petalinux-config → FPGA Manager → [*] Fpga Manager

```
petalinux-create -t apps --template fpgamanager_dtg -n gpio --enable --
srcuri "<path>/gpio.xsa <path>/shell.json"
```

(or)

```
petalinux-create -t apps --template fpgamanager -n gpio --enable --srcuri
"<path>/pl.dtsi <path>/system.bit <path>/shell.json"
```

You can find the dtbo and bin files of the generated base and template applications in `/lib/firmware/xilinx/`

Yocto Recipe Name Changes

Yocto updated the following recipe names in 2021.1:

Table 32: **Updated Yocto Recipe Names**

Old Name	New Name
fsbl.bb	fsbl-firmware.bb
plm.bb	plm-firmware.bb

If you are creating or using any bbappends for these components, you need to update them.

For example:

```
$ mkdir <plnx-proj-root>/project-spec/meta-user/recipes-bsp/emebeddedsw/
$ touch <plnx-proj-root>/project-spec/meta-user/recipes-bsp/emebeddedsw/
fsbl-firmware_%.bbappend
Add recipe changes into fsbl-firmware_%.bbappend file.
```

Host GCC Version Upgrade

From this release, the gcc version should be greater than 6. Using lower versions of gcc causes build issues. You can also enable the **Enable buildtools extended** option from **petalinux -config → Yocto** settings, which uses the pre-compiled gcc binaries from the PetaLinux tool.

Image Selector

The Image Selector (ImgSel) is a bare-metal application running out of OCM after a POR/SRST to select the correct boot image. Image Selector is a generic application designed to share across different ZU+ boards.

Image Selector currently supports the selection of boot images based on the below parameters for different boards. You can enable this option in PetaLinux using the following:

```
petalinux-config → Linux Components Selection → []Image Selector
```

- Board EEPROM data
- QSPI A/B update register data

***Note*:**

Image Selector is optional and only required if the boot image has to be selected based on Board EEPROM data or QSPI A/B mechanism.

XILINX

Appendix B

PetaLinux Project Structure

This section provides a brief introduction to the file and directory structure of a PetaLinux project. A PetaLinux project supports development of a single Linux system development at a time. A built Linux system is composed of the following components:

- Device tree
- First stage boot loader (optional)
- U-Boot
- Linux kernel
- The root file system is composed of the following components:
  - Prebuilt packages
  - Linux user applications (optional)
  - User modules (optional)

A PetaLinux project directory contains configuration files of the project, the Linux subsystem, and the components of the subsystem. The `petalinux-build` command builds the project with those configuration files. You can run `petalinux-config` to modify them. The following is an example of a PetaLinux project:

```
project-spec
   hw-description
   configs
   meta-user
pre-built
   linux
      implementation
      images
      xen
hardware
  <project-name>
components
   plnx_workspace
      device-tree
config.project
README
```

Table 33: **PetaLinux Project Description**

File / Directory in a PetaLinux Project	Description
`/.petalinux/`	Directory to hold tools usage and WebTalk data.
`/config.project/`	Project configuration file.
`/project-spec`	Project specification.
`/project-spec/hw-description`	Hardware description imported from Vivado® design tools.
`/project-spec/configs`	Configuration files of top level config and RootFS config.
`/project-spec/configs/config`	Configuration file used to store user settings.
`/project-spec/configs/rootfs_config`	Configuration file used for root file system.
`/project-spec/configs/busybox`	Configuration file for busybox.
`/project-spec/configs/init-ifupdown`	Configuration file for Ethernet.
`/components/plnx_workspace/device-tree/device-tree/`	Device tree files used to build device tree. The following files are auto generated by `petalinux-config`: • `skeleton.dtsi` (Zynq-7000 devices only) • `zynq-7000.dtsi` (Zynq-7000 devices only) • `zynqmp.dtsi` (Zynq UltraScale+ MPSoC only) • `pcw.dtsi` (Zynq-7000 devices and Zynq UltraScale+ MPSoC only) • `pl.dtsi` • `system-conf.dtsi` • `system-top.dts` • `<bsp name>.dtsi` It is not recommended to edit these files, as these files are regenerated by the tools.
`/project-spec/meta-user/recipes-bsp/device-tree/files/`	`system-user.dtsi` is not modified by any PetaLinux tools. This file is safe to use with revision control systems. In addition, you can add your own DTSI files to this directory. You have to edit the `<plnx-proj-root>/project-spec/meta-user/recipes-bsp/device-tree/device-tree.bbappend` by adding your DTSI file.
`/project-spec/meta-user/recipes-bsp/u-boot/files/platform-top.h` (only for MicroBlaze processors)	`platform-top.h` is copied to `include/configs/` directory in the U-Boot source
`/project-spec/meta-user/conf/petalinuxbsp.conf`	This configuration file contains all the local user configurations for your build environment. It is a substitute for `local.conf` in the Yocto meta layers.

Notes:

1. All the paths are relative to `<plnx-proj-root>`.

When the project is built, three directories are auto generated:

- `<plnx-proj-root>/build` for the files generated for build.
- `<plnx-proj-root>/images` for the bootable images.

- `<plnx-proj-root>/build/tmp` for the files generated by Yocto. This directory is configurable through `petalinux-config`.
- `<plnx-proj-root>/components/yocto` has Yocto eSDK. This file is generated when execute `petalinux-config/petalinux-build`.
- When `petalinux-package --prebuilt` is run, it creates a pre-built/directory in `<PROJECT>/` and copies the build image files from the `<PROJECT>/images/linux` directory to the `<PROJECT>/pre-built/linux/images/` directory.

Here is an example:

```
├── build
│   ├── bitbake-cookerdaemon.log
│   ├── build.log
│   ├── cache
│   ├── conf
│   ├── downloads
│   ├── misc
│   ├── sstate-cache
│   └── tmp
├── components
│   ├── plnx_workspace
│   └── yocto
├── config.project
├── hardware
│   └── project-name
├── images
│   └── linux
├── pre-built
│   └── linux
├── project-spec
│   ├── attributes
│   ├── configs
│   ├── hw-description
│   └── meta-user
└── README
```

Note: `<plnx-proj-root>/build/` are automatically generated. Do not manually edit files in this directory. Contents in this directory are updated when you run `petalinux-config` or `petalinux-build`. `<plnx-proj-root>/images/` are also automatically generated. Files in this directory are updated when you run `petalinux-build`.

The table below is an example for Zynq® UltraScale+™ MPSoC.

By default the build artifacts are removed to preserve space after `petalinux-build`. To preserve the build artifacts, you have to add the `INHERIT_remove = "rm_work"` in `<plnx-proj-root>/project-spec/meta-user/conf/petalinuxbsp.conf`, but it increases the project space.

Table 34: **Build Directory in a PetaLinux Project**

Build Directory in a PetaLinux Project	Description
`<plnx-proj-root>/build/build.log`	Logfile of the build.
`<plnx-proj-root>/build/misc/config/`	Directory to hold files related to the Linux subsystem build.

Table 34: **Build Directory in a PetaLinux Project** *(cont'd)*

Build Directory in a PetaLinux Project	Description
`<plnx-proj-root>/build/misc/rootfs_config/`	Directory to hold files related to the RootFS build.
`${TMPDIR}/work/zynqmp_generic-xilinx-linux/petalinux-image-minimal/1.0-r0/rootfs`	RootFS copy of target. This is the staging directory.
`${TMPDIR}/zynqmp_generic-xilinx-linux`	Stage directory to hold the libs and header files required to build user apps/libs.
`${TMPDIR}/work/zynqmp_generic-xilinx-linux/linux-xlnx/`	Directory to hold files related to the kernel build.
`${TMPDIR}/work/zynqmp_generic-xilinx-linux/u-boot-xlnx`	Directory to hold files related to the U-Boot build.
`<plnx-proj-root>/components/plnx_workspace/device-tree/device-tree"`	Directory to hold files related to the device tree build.
`<plnx-proj-root>/components/yocto`	Directory to hold Yocto eSDK content.

Table 35: **Image Directory in a PetaLinux Project**

Image Directory in a PetaLinux Project	Description
`<plnx-proj-root>/images/linux/`	Directory to hold the bootable images for Linux subsystem
`<plnx-proj-root>/images/linux`	Directory to hold the bootable images for xen hyperviser

Project Layers

The PetaLinux project has the following layer under `<plnx-proj-root>/project-spec`.

meta-user

This layer is a place holder for all user-specific changes. You can add your own bbappend and configuration files in this layer.

XILINX

Appendix C

Generating Boot Components

Platform Loader and Manager Firmware (PLM)

This is for Versal™ ACAP. This is mandatory. By default, the top-level system settings are set to generate the PLM.

If you had disabled PLM from menuconfig previously, you can configure the project to build PLM as follows:

1. Launch top level system settings configuration menu and configure:

   ```
   $ petalinux-config
   ```

   a. Select **Linux Components Selection --->** sub-menu.

   b. Select PLM.

   c. Enter your settings.

   d. Exit the menu and save the change.

2. Build the PLM when building the project:

   ```
   $ petalinux-build
   ```

3. Build the PLM only:

   ```
   $ petalinux-build -c plm
   ```

The PLM ELF file is installed as `plm.elf` for Versal ACAP in `images/linux` inside the project root directory.

For more information on PLM, see *Versal ACAP System Software Developers Guide* (UG1304).

Processing System Management Firmware (PSM)

This is for Versal™ ACAP. This is mandatory. By default, the top-level system settings are set to generate the PSM.

If you had disabled PSM from menuconfig previously, you can configure the project to build PSM as follows:

1. Launch top level system settings configuration menu and configure:

```
$ petalinux-config
```

   a. Select the **Linux Components Selection** sub-menu.
   b. Select PSM firmware.
   c. Enter your settings.
   d. Exit the menu and save the change.
2. Build the PSM when building the project:

```
$ petalinux-build
```

3. Build the PSM only:

```
$ petalinux-build -c psm-firmware
```

The PSM ELF file is installed as `psmfw.elf` for Versal ACAP in `images/Linux` inside the project root directory.

For more information on PSM, see *Versal ACAP System Software Developers Guide* (UG1304).

Image Selector

***Note*:** This section is only for Zynq® UltraScale+™ MPSoCs.

By default, the top-level system settings are not set to generate the Image Selector. If you want enable Image Selector from menuconfig, configure the project to build Image Selector as follows:

1. Launch top level system settings configuration menu and configure:

```
$ petalinux-config
```

   a. Select the Linux Components Selection sub-menu.
   b. Select Image Selector.

c. Enter your settings.

d. Exit the menu and save the change.

2. Build the Image Selector when building the project:

```
$ petalinux-build
```

3. Build the Image Selector only:

```
petalinux-build -c imgsel
```

The image selector ELF file is installed as `imgsel.elf` for Zynq UltraScale+ MPSoCs in the `images/Linux` inside the project root directory.

First Stage Boot Loader for Zynq UltraScale+ and Zynq-7000 Devices

By default, the top level system settings are set to generate the first stage boot loader. This is optional.

Note: If you do not want the PetaLinux build FSBL/FS-BOOT, then you will need to manually build it on your own. Else, your system will not boot properly.

If you had disabled first stage boot loader from menuconfig previously, You can configure the project to build first stage boot loader as follows:

1. Launch top level system settings configuration menu and configure:

```
$ petalinux-config
```

a. Select **Linux Components Selection --->** sub-menu.

b. Select **First Stage Boot Loader** option.

```
[*] First Stage Bootloader
```

c. Select the **FSBL Configuration --->** submenu.

d. For application compiler flags, select **FSBL Configuration → FSBL compiler flags**.

e. For BSP compiler flags, select **FSBL Configuration → FSBL BSP extra compiler flags**.

f. Enter your compilation flags.

g. Exit the menu and save the change.

2. Launch `petalinux-build` to build the FSBL:

Build the FSBL when building the project:

```
$ petalinux-build
```

Build the FSBL only:

```
$ petalinux-build -c fsbl (for MicroBlaze, it is fs-boot)
```

The boot loader ELF file is installed as `zynqmp_fsbl.elf` for Zynq UltraScale+ MPSoC, `zynq_fsbl.elf` for Zynq®-7000 devices and `fs-boot.elf` for MicroBlaze™ processors in `images/linux` inside the project root directory.

For more information on FSBL, see https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842019/FSBL.

Arm Trusted Firmware (ATF)

This is for Zynq UltraScale+ MPSoC and Versal ACAP. This is mandatory. By default, the top level system settings are set to generate the ATF.

You can set the ATF configurable options as follows:

1. Launch top level system settings configuration menu and configure:

   ```
   $ petalinux-config
   ```

   a. Select the **Arm Trusted Firmware Compilation Configuration --->** submenu.

   b. Enter your settings.

   c. Exit the menu and save the change.

2. Build the ATF when building the project:

   ```
   $ petalinux-build
   ```

   Build the ATF only:

   ```
   $ petalinux-build -c arm-trusted-firmware
   ```

   The ATF ELF file is installed as `bl31.elf` for Zynq UltraScale+ MPSoC and Versal ACAP in `images/linux` inside the project root directory.

For more information on ATF, see https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842107/Arm+Trusted+Firmware.

PMU Firmware

This is for Zynq UltraScale+ MPSoC only. This is optional. By default, the top level system settings are set to generate the PMU firmware.

CAUTION! *If you do not want PetaLinux to build the PMU firmware, you have to manually build it on your own. Else, your system will not boot properly.*

You can configure the project to build PMU firmware as follows:

1. Launch top level system settings configuration menu and configure:

   ```
   $ petalinux-config
   ```

   a. Select **Linux Components Selection**.

   b. Select **PMU Firmware** option.

   ```
   [*] PMU Firmware
   ```

   c. Select the **PMUFW Configuration → PMUFW compiler flags** submenu.

   d. Enter your compilation flags.

   e. Exit the menu and save the change.

2. Build the PMU firmware when building the project:

   ```
   $ petalinux-build
   ```

   Build the PMU firmware only:

   ```
   $ petalinux-build -c pmufw
   ```

   The PMU firmware ELF file is installed as `pmufw.elf` for Zynq UltraScale+ MPSoC in `images/linux` inside the project root directory.

   For more information on PMU Firmware, see http://www.wiki.xilinx.com/PMU+Firmware.

FS-Boot for MicroBlaze Platform Only

FS-Boot in PetaLinux is a first stage boot loader demo for MicroBlaze™ platform only. It is to demonstrate how to load images from flash to the memory and jump to it. If you want to try FS-Boot, you must have a minimum of 8 KB block RAM.

FS-Boot supports parallel flash and SPI flash in standard SPI mode and Quad SPI mode only.

In order for FS-Boot to know where in the flash should get the image, macro `CONFIG_FS_BOOT_START` needs to be defined. This is done by the PetaLinux tools. PetaLinux tools set this macro automatically from the `boot` partition settings in the menuconfig primary flash partition table settings. For parallel flash, it is the start address of boot partition. For SPI flash, it is the start offset of `boot` partition.

The image in the flash requires a wrapper header followed by a BIN file. FS-Boot gets the target memory location from wrapper. The wrapper needs to contain the following information:

Table 36: **Wrapper Information**

Offset	Description	Value
0×0	FS-Boot bootable image magic code	×0b8b40008
0×4	BIN image size	User-defined
0×100	FS-Boot bootable image target memory address	User-defined. The PetaLinux tool automatically calculates it from the U-Boot text base address offset from the Memory Settings from the menuconfig.
0×10c	Where the BIN file start	None

The FS-Boot ignores other fields in the wrapper header. The PetaLinux tool generates the wrapper header to wrap around the U-Boot BIN file.

***Note*:** PetaLinux only supports 32-bit MicroBlaze processors.

The FS-Boot supports symmetric multi processing (SMP) from the 2020.1 release onwards. You can have multiple MicroBlaze processors in your design. A maximum of eight cores is supported.

The same FS-Boot which is built as part of the `petalinux-build/petalinux-build -c fsboot` works for all the cores. XSDB is needed to flash the FS-Boot on all the cores. The following is an example for four cores. `xsdb > ta` lists all the available cores. To boot your target with SMP support, follow these steps:

```
<plnx-tool>/tools/xsct/bin/xsdb
xsdb > connect -url <target-url>
xsdb > fpga -f <plnx-proj-root>/images/linux/system.bit
xsdb > ta
xsdb > ta <core number>
xsdb > dow -f <plnx-proj-root>/images/linux/fs-boot.elf
the above two steps for all available cores.
xsdb > dow -f <plnx-proj-root>/images/linux/u-boot.elf
xsdb > dow -f <plnx-proj-root>/images/linux/image.ub
```

XILINX

Appendix D

QEMU Virtual Networking Modes

There are two execution modes in QEMU: non-root (default) and root requires sudo or root permission). The difference in the modes relates to virtual network configuration.

In non-root mode QEMU sets up an internal virtual network which restricts network traffic passing from the host and the guest. This works similar to a NAT router. You can not access this network unless you redirect tcp ports.

In root mode QEMU creates a subnet on a virtual Ethernet adapter, and relies on a DHCP server on the host system.

The following sections detail how to use the modes, including redirecting the non-root mode so it is accessible from your local host.

Specifying the QEMU Virtual Subnet

By default, PetaLinux uses `192.168.10.*` as the QEMU virtual subnet in `--root` mode. If it has been used by your local network or other virtual subnet, you may wish to use another subnet. You can configure PetaLinux to use other subnet settings for QEMU by running `petalinux-boot` as follows on the command console:

***Note*:** This feature requires sudo access on the local machine, and must be used with the `--root` option.

```
$ petalinux-boot --qemu --root --u-boot --subnet <subnet gateway IP>/
<number of the bits of the subnet mask>
```

For example, to use subnet `192.168.20.*`:

```
$ petalinux-boot --qemu --root --u-boot --subnet 192.168.20.0/24
```

Appendix E

Xilinx IP Models Supported by QEMU

Note: By default, QEMU disables any devices for which there is no model available. For this reason it is not possible to use QEMU to test your own customized IP cores (unless you develop C/C++ models for them according to QEMU standard).

For more information on Xilinx® IP models supported by QEMU, see *Xilinx Quick Emulator User Guide: QEMU*.

XILINX

Appendix F

Xen Zynq UltraScale+ MPSoC and Versal ACAP Example

This section details on the Xen Zynq® UltraScale+™ MPSoC and Versal™ ACAP example. It describes how to get Linux to boot as dom0 on top of Xen on Zynq UltraScale+ MPSoC and Versal ACAP.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have PetaLinux tools software platform ready for building a Linux system customized to your hardware platform. For more information, see Importing Hardware Configuration.
- You have created a PetaLinux project from the reference BSP.
- There are Xen related prebuilts in the `pre-built/linux/xen` directory, which are `xen.dtb`, `xen-openamp.dtb`, `xen-qemu.dtb`, `xen-Image`, and `xen-rootfs.cpio.gz`.

Boot Prebuilt Linux as dom0

1. Copy the prebuilt Xen images to your TFTP directory so that you can load them from U-Boot with TFTP.

```
$ cd <plnx-proj-root>
$ cp pre-built/linux/xen/xen.dtb <tftpboot>/
$ cp pre-built/linux/xen/xen-openamp.dtb <tftpboot>/
$ cp pre-built/linux/xen/xen-qemu.dtb <tftpboot>/
$ cp pre-built/linux/xen/xen-Image <tftpboot>/
$ cp pre-built/linux/xen/xen-rootfs.cpio.gz <tftpboot>/
$ cp pre-built/linux/xen/xen_boot_tftp.scr <tftpboot>/
$ cp pre-built/linux/xen/xen_boot_sd.scr  <tftpboot>/
$ cp pre-built/linux/xen/xen  <tftpboot>/
```

2. Boot the prebuilt U-Boot image on the board with either JTAG boot or boot from SD card.

   Note: For SD card boot, see Boot a PetaLinux Image on Hardware with an SD Card and for JTAG boot, see Boot a PetaLinux Image on Hardware with JTAG.

3. Setup TFTP server IP from U-Boot:

```
platform> setenv serverip <TFTP SERVERIP>
```

4. Load Xen images from U-Boot.
   - **TFTP Boot:** `xen_boot_tftp.scr`, to be loaded at address `0xC00000` as shown:

```
tftpb 0xC00000 xen_boot_tftp.scr; source 0xC00000
```

   - **SD Boot:** `xen_boot_sd.scr`, to be loaded at address `0xC00000` as shown:

```
load mmc 0:1 0xC00000 xen_boot_sd.scr; source 0xC00000
```

Rebuild Xen

After creating a PetaLinux project for Zynq UltraScale+ MPSoC and Versal™ ACAP, follow the below steps to build Xen images:

1. Go to `cd <proj root directory>`.
2. In the `petalinux-config` command, select **Image Packaging Configuration → Root filesystem type (INITRD)**.
3. In `petalinux-config -c rootfs`, select **PetaLinux Package Groups → Packagegroup-petalinux-xen → [*] packagegroup-petalinux-xen**.

   ***Note*:** If you enable Xen when `/switch_root` is enabled, you will see build failures as Xen only supports ramfs boot. ext4-based boot is enable if you enable `switch_root`. To resolve the issue, change the above config to `petalinux-image-minimal` from `petalinux-initramfs-image`.
4. Edit the device tree to build in the extra Xen related configs. Edit this file: `project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi` and add this line: `/include/ "xen.dtsi"`

   It should look like the following:

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

5. Edit the file: `project-spec/meta-user/recipes-bsp/device-tree/device-tree.bbapp end` and add this line to it: `SRC_URI += "file://xen.dtsi"`

   The file should look like this:

```
FILESEXTRAPATHS_prepend := "${THISDIR}/files:"

SRC_URI += "file://system-user.dtsi"
SRC_URI += "file://xen.dtsi"
```

6. Run `petalinux-build`.

7. The build artifacts are in `images/linux` in the project directory.

Note: By default, the `petalinux-build` command does not build Xen. The default root file system does not contain the Xen tools. You have to use Xen RootFS.

IMPORTANT! *You are required to update dom0 memory in xen-bootargs in the `xen.dtsi` file based on the image/RootFS size. Also, adjust the above load addresses based on the image/RootFS size without overlapping.*

Boot Built Linux as dom0

1. Copy built Xen images to your TFTP directory so that you can load them from U-Boot with TFTP.

```
$ cd <plnx-proj-root>
$ cp images/linux/system.dtb <tftpboot>/
$ cp images/linux/Image <tftpboot>/
$ cp images/linux/xen_boot_tftp.scr <tftpboot>/
$ cp images/linux/xen_boot_sd.scr  <tftpboot>/
$ cp images/linux/xen  <tftpboot>/
$ cp images/linux/rootfs.cpio.gz  <tftpboot>/
```

2. Boot built U-Boot image on the board with either JTAG boot or boot from SD card.

   Note: For SD card boot, see Boot a PetaLinux Image on Hardware with an SD Card and for JTAG boot, see Boot a PetaLinux Image on Hardware with JTAG.

   Note: You can also point the dom1 and dom2 to the domU kernels in the configuration itself so that the Xen boot files are updated with the images that are being pointed to. Edit the configuration file as shown"

```
$ vi images/linux/xen.cfg
$ export XEN_CONFIG="<Absolute path for xen.cfg>"
$ export XEN_CONFIG_SKIP="1"
$ export BB_ENV_EXTRAWHITE="$BB_ENV_EXTRAWHITE XEN_CONFIG
XEN_CONFIG_SKIP"
$ petalinux-build -c kernel -x do_deploy
```

   Note: Xen boot files are generated in the `<plnx-proj-root>/images/linux` folder.

3. Setup TFTP server IP from U-Boot:

```
Platform> setenv serverip <TFTP SERVERIP>
```

   Note: Platform refers to Versal™ or Zynq® UltraScale+™ MPSoC.

4. Load Xen images from U-Boot:

   - **TFTP Boot:** `xen_boot_tftp.scr`, to be loaded at address `0xC00000` as shown::

```
tftpb 0xC00000 xen_boot_tftp.scr; source 0xC00000
```

- **SD Boot:** `xen_boot_sd.scr`, to be loaded at address `0xC00000` as shown:

```
load mmc 0:1 0xC00000 xen_boot_sd.scr; source 0xC00000
```

***Note*:** For more information, see http://www.wiki.xilinx.com/XEN+Hypervisor.

XILINX

Appendix G

Booting Prebuilt OpenAMP

Use the following steps to execute OpenAMP:

To boot prebuilt Linux for Versal™ ACAP, follow these steps:

1. Generate BOOT.BIN for Versal ACAP.

```
petalinux-package --boot --plm pre-built/linux/images/plm.elf --psmfw
pre-built/linux/images/psmfw.elf --dtb pre-built/linux/images/
openamp.dtb --u-boot -o pre-built/linux/images/BOOT.BIN --force.
```

2. Boot Linux

```
petalinux-boot --jtag --prebuilt 3 --hw_server-url <hostname:3121>
```

To boot prebuilt Linux for Zynq® UltraScale+™ MPSoC, follow these steps:

```
$ cd <plnx-proj-root>
$ cp pre-built/linux/images/openamp.dtb pre-built/linux/images/system.dtb
$ petalinux-boot --jtag --prebuilt 3 --hw_server-url <hostname:3121>
```

Once Linux is booted, run the following commands for Versal devices only:

1. `modprobe virtio_rpmsg_bus`
2. `modprobe zynqmp_r5_remoteproc`

To load OpenAMP firmware and run OpenAMP test application, run the following command:

```
$ echo <echo_test_firmware> > /sys/class/remoteproc/remoteproc0/firmware
```

For example, to load image_echo_test, run:

```
$ echo image_echo_test > /sys/class/remoteproc/remoteproc0/firmware
$ echo start > /sys/class/remoteproc/remoteproc0/state
$ echo_test
$ echo stop > /sys/class/remoteproc/remoteproc0/state
```

To stop running, run the following command:

```
$ echo stop > /sys/class/remoteproc/remoteproc0/state
```

For more examples, see *Libmetal and OpenAMP for Zynq Devices User Guide* (UG1186).

Partitioning and Formatting an SD Card

For partitioning and formatting an SD card, the following tools are required:

- fdisk
- mkfs

The steps and logs for partitioning are as follows:

- `sudo fdisk /dev/sdb`

```
Welcome to fdisk (util-linux 2.31.1).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.
```

- `Command (m for help): n`

  Partition type

  - p primary (0 primary, 0 extended, 4 free)
  - e extended (container for logical partitions)

- `Select (default p): p`

```
Partition number (1-4, default 1):
First sector (2048-62333951, default 2048):
```

- `Last sector, +sectors or +size{K,M,G,T,P} (2048-62333951, default 62333951): 21111220`

  Creates a new partition 1 of type 'Linux' and of size 10.1 GB. Partition #1 contains a vfat signature.

- `Do you want to remove the signature? [Y]es/[N]o: y`

  The signature will be removed by a write command.

- `Command (m for help): n`

  Partition type

  - p primary (1 primary, 0 extended, 3 free)
  - e extended (container for logical partitions)

- `Select (default p): p`

```
Partition number (2-4, default 2):
First sector (21111221-62333951, default 21112832):
Last sector, +sectors or +size{K,M,G,T,P} (21112832-62333951, default
62333951):
Created a new partition 2 of type 'Linux' and of size 19.7 GB.
```

- `Command (m for help): w`

```
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.
Steps and log for formatting:
```

- `$ sudo mkfs.vfat /dev/sdb1`

  mkfs.fat 4.1 (2017-01-24)

- `$ sudo mkfs.ext4 /dev/sdb2`

```
mke2fs 1.44.1 (24-Mar-2018)
Creating file system with 5152640 4k blocks and 1289280 inodes
File system UUID: ad549f34-ee6e-4efc-ab03-fba390e98ede
Superblock backups stored on blocks:
32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
4096000
Allocating group tables: done
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and file system accounting information: done
```

- `SD EXT ROOTFS BOOT:`

```
Mount the fat partition and copy BOOT.BIN, boot.scr, Image, and
system.dtb files on it.
Mount the EXT partition and untar rootfs.tar.gz to it.
Finally unmount the SD card and use it for booting.
```

Auto-mounting an SD Card

Auto-mounting an SD Card during the Build

To auto-mount an SD card during the build, follow the instructions in https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842475/PetaLinux+Yocto+Tips#PetaLinuxYoctoTips-HowtoAutoMountSDcardinYoctoRecipes.

Auto-mounting SD Partitions after Linux is Running

To mount the SD partitions to a user-defined path, follow these steps:

1. Boot the target to the Linux prompt.
2. Create the direct path, for example:

```
mkdir /media/card
```

3. Edit the file as shown below:

```
vim /etc/fstab
```

Once the editor opens, add the following line:

```
/dev/mmcblk0p2 /media/card ext4 defaults 0 1 4
```

Save and exit.

```
$ reboot
```

4. Check the SD mount point using the `mount` command:

```
root@xilinx-vck190-2021_1:~# mount
/dev/root on / type ext4 (rw,relatime)
devtmpfs on /dev type devtmpfs
(rw,relatime,size=8053972k,nr_inodes=2013493,mode=755)
proc on /proc type proc (rw,relatime)
sysfs on /sys type sysfs (rw,relatime)
debugfs on /sys/kernel/debug type debugfs (rw,relatime)
configfs on /sys/kernel/config type configfs (rw,relatime)
tmpfs on /run type tmpfs (rw,nosuid,nodev,mode=755)
tmpfs on /var/volatile type tmpfs (rw,relatime)
/dev/mmcblk0p2 on /media/sd-mmcblk0p2 type ext4 (rw,relatime)
/dev/mmcblk0p1 on /media/sd-mmcblk0p1 type vfat
(rw,relatime,fmask=0022,dmask=0022,codepage=437,iocharset=iso8859-1,short
name=mixed,errors=remount-ro)
devpts on /dev/pts type devpts (rw,relatime,gid=5,mode=620,ptmxmode=000)
root@xilinx-vck190-2021_1:~#
```

XILINX

Appendix J

PetaLinux Commands

There are eight independent commands that make up the PetaLinux design flow. They are:

- petalinux-create
- petalinux-config
- petalinux-build
- petalinux-boot
- petalinux-package
- petalinux-util
- petalinux-upgrade
- petalinux-devtool

In most cases, the PetaLinux commands are flexible such that the specific options passed to the tools present you with a unique use model, compared to other options for the same tool.

For the purposes of this document, command line arguments that behave as modifiers for workflows are referred to as "options." User-specified values that are accepted by options are shown in italics. In some cases, omitting the user-specified value might result in a built-in default behavior. See the "Default Value" column in the tables for details about relevant default values.

petalinux-create

The `petalinux-create` tool creates objects that are part of a PetaLinux project. This tool provides two separate workflows. In the `petalinux-create -t project` workflow, the tool creates a new PetaLinux project directory structure. In the `petalinux-create -t COMPONENT` workflow, the tool creates a component within the specified project.

These workflows are executed with `petalinux-create -t project` or `petalinux-create -t COMPONENT`, respectively.

petalinux-create Command Line Options

The following table details the command line options that are common to all `petalinux-create` workflows.

Table 37: **petalinux-create Command Line Options**

Option	Functional Description	Value Range	Default Value
`-t,--type TYPE`	Specify the TYPE of object to create. This is required.	• project • apps • modules	None
`-n,--name NAME`	Create object with the specified NAME. This is optional when creating a project from a BSP source. Otherwise, this is required.	User-specified	When creating a project from a BSP source, the project takes the name of the source BSP.
`-p,--project PROJECT`	PetaLinux project directory path for component creation in a project. This is optional.	User-specified	Current Directory
`--force`	Overwrite existing files on disk. This is optional.	None	None
`-h,--help`	Display usage information. This is optional.	None	None
--tmpdir	Specify the local drive path as TMPDIR location when creating the project. Default TMPDIR cannot be under NFS. By default, PetaLinux sets the TMPDIR under /tmp when project is on NFS. You can set your own local drive as TMPDIR PATH using the `--tmpdir` option	None	None

petalinux-create -t project

The `petalinux-create -t project` command creates a new PetaLinux project at the specified location with a specified name. If the specified location is on the Network File System (NFS), it changes the TMPDIR automatically to `/tmp/<projname_timestamp>`. If `/tmp/<projname_timestamp>` is also on NFS, it throws an error. You can change the TMPDIR through `petalinux-config`. Do not configure the same location as TMPDIR for two different PetaLinux projects as this can cause build errors.

petalinux-create -t project Options

The following table details options used when creating a project. These options are mutually exclusive and one of them must be used when creating a new project.

Table 38: **petalinux-create -t project Options**

Option	Functional Description	Value Range	Default Value
`--template TEMPLATE`	Assumes the specified CPU architecture, and is only required when `--source` is not provided.	• microblaze • zynqMP • zynq • versal	None
`-s,--source SOURCE`	Creates project based on specified BSP file. SOURCE is the full path on disk to the BSP file. This is optional.	User-specified	None

Note: For Xilinx® boards, the `-s, --source` BSP flows are suggested. For custom boards, the `--template` flow is required.

petalinux-create -t project Examples

The following examples demonstrate proper usage of the `petalinux-create -t project` command.

- Create a new project from a reference BSP file:

```
$ petalinux-create -t project -s <PATH-TO-BSP>
```

- Create a new project based on the MicroBlaze™ processor template:

```
$ petalinux-create -t project -n <NAME> --template microblaze
```

- Create a project from the PetaLinux project BSP and specify the TMPDIR PATH:

```
$ petalinux-create -t project -s <PATH_TO_PETALINUX_PROJECT_BSP> --tmpdir
<TMPDIR PATH>
```

- Create a project from a template and specify the TMPDIR PATH:

```
$ petalinux-create -t project -n <PROJECT> --template <TEMPLATE> --tmpdir
<TMPDIR PATH>
```

By default, the directory structure created by `--template` is minimal, and is not useful for building a complete system until initialized using the `petalinux-config --get-hw-description` command. Projects created using a BSP file as their source are suitable for building immediately.

petalinux-create -t COMPONENT

The `petalinux-create -t COMPONENT` command allows you to create various components within the specified PetaLinux project. These components can then be selectively included or excluded from the final system by toggling them using the `petalinux-config -c rootfs` workflow.

petalinux-create -t COMPONENT Options

The `petalinux-create -t apps` command allows you to customize how application components are created. The following table details options that are common when creating applications within a PetaLinux project

Table 39: **petalinux-create -t apps Options**

Option	Functional Description	Value Range	Default Value
`-s,--source SOURCE`	Create the component from pre-existing content on disk. Valid formats are .tar.gz, .tar.bz2, .tar, .zip, and source directory (uncompressed). This is optional.	User-specified	None
`--template TEMPLATE`	Create the component using a pre-defined application template. This is optional.	• c • c++ • autoconf, for GNU autoconfig • fpgamanager • install, for applications which have prebuilt binary only	c
`--enable`	Upon creating the component, enable it in the project's root file system. You can also enable using the `petalinux-config -c rootfs`. This is optional.	None	Disabled
--srcuri	Creates an application with local sources or from remote source.	None	None

petalinux-create -t COMPONENT Examples

The following examples demonstrate proper usage of the `petalinux-create -t COMPONENT` command.

- Create an application component that is enabled in the root file system.

```
$ petalinux-create -t apps -n <NAME> --template <template> --enable
```

- Create a new install-only application component. In this flow, nothing is compiled.

```
$ petalinux-create -t apps -n <NAME> --template install
```

- Create a new kernel module and enable it.

```
$ petalinux-create -t modules -n <name> --template <template> --enable
```

- Create an application with multiple source files.

```
$ petalinux-create -t apps --template install --name mylibs --srcuri
"<path-to-dir>/mylib1.so <path-to-dir>/mylib2.so"
```

- Create an app with remote sources. The following examples will create applications with specified git/http/https pointing to the srcuri.

```
$ petalinux-create -t apps -n myapp --enable --srcuri http://
example.tar.gz
```

```
$ petalinux-create -t apps -n myapp --enable --srcuri git://example.git
\;protocol=https
```

```
$ petalinux-create -t apps -n myapp --enable --srcuri https://
example.tar.gz
```

Note: This is applicable for applications and modules.

petalinux-config

The `petalinux-config` tool allows you to customize the specified project. This tool provides two separate workflows. In the `petalinux-config --get-hw-description` workflow, a project is initialized or updated to reflect the specified hardware configuration. In the `petalinux-config -c COMPONENT` workflow, the specified component is customized using a menuconfig interface.

petalinux-config Command Line Options

The following table details the available options for the `petalinux-config` tool.

Table 40: **petalinux-config Command Line Options**

Option	Functional Description	Value Range	Default Value
`-p, --project <path to project directory>`	Specifies path to the project to be configured.	User-specified	Current Directory
`--get-hw-description <DIR containing XSA>/--get-hw-description=<DIR containing XSA>/--get-hw-description=<PATH-TO-XSA>`	Initializes or updates the hardware configuration for the PetaLinux project. Mutually exclusive with -c. This is required.	User-specified	Current Directory

Table 40: **petalinux-config Command Line Options** *(cont'd)*

Option	Functional Description	Value Range	Default Value
`-c,--component COMPONENT`	Configures the specified system component. Mutually exclusive with `--get-hw-description`. This is required.	• kernel • rootfs • u-boot • bootloader (for Zynq® UltraScale+™ MPSoC, Zynq architecture, and MicroBlaze™ CPU) • pmufw, for Zynq UltraScale+ MPSoC only • device-tree • plm, for Versal™ ACAP • psmfw, for Versal ACAP	None
`--defconfig DEFCONFIG`	Initializes the Linux kernel/U-Boot configuration using the specified `defconfig` file. Valid for Linux kernel and U-Boot. This is optional.	User-specified. For example, for Linux kernel, the file name of a file in `<kernel_source>/arch/<ARCH>/configs/` is `XXX_defconfig`. For U-Boot, the file name of a file in `<uboot_ source> / configs` is `XXX_defconfig`.	None
`--silentconfig`	Allows you to restore a prior configuration. Example: Execute the following command after enabling or disabling different configs by editing `<proj-root>/project-spec/configs/config` `$ petalinux-config --silentconfig`	None	None
`-v,--verbose`	Displays additional output messages. This is optional.	None	None
`-h,--help`	Displays tool usage information. This is optional.	None	None

petalinux-config --get-hw-description

The `petalinux-config --get-hw-description` command allows you to initialize or update a PetaLinux project with hardware-specific information from the specified Vivado® Design Suite hardware project. The components affected by this process can include FSBL configuration, U-Boot options, Linux kernel options, and the Linux device tree configuration. This workflow should be used carefully to prevent accidental and/or unintended changes to the hardware configuration for the PetaLinux project. The path used with this workflow is the directory that contains the XSA file rather than the full path to the XSA file itself. This entire option can be omitted if run from the directory that contains the XSA file.

petalinux-config --get-hw-description Examples

The following examples demonstrate proper usage of the `petalinux-config --get-hw-description` command.

- Initialize a PetaLinux project within the project directory with an external XSA.

```
$ petalinux-config --get-hw-description <PATH-TO-XSA DIRECTORY>/--get-hw-description=<PATH-TO-XSA DIRECTORY>/--get-hw-description=<PATH-TO-XSA>
```

- Initialize a PetaLinux project from within the directory containing an XSA.

```
$ petalinux-config --get-hw-description -p <PATH-TO-PETALINUX-PROJECT>
```

- Initialize a PetaLinux project from a neutral location.

```
$ petalinux-config --get-hw-description <PATH-TO-XSA DIRECTORY>/--get-hw-description=<PATH-TO-XSA DIRECTORY>/--get-hw-description=<PATH-TO-XSA> -p <PATH-TO-PETALINUX-PROJECT>
```

petalinux-config -c COMPONENT

The `petalinux-config -c COMPONENT` command allows you to use a standard menuconfig interface to control how the embedded Linux system is built, and also generates the source code for embedded software applications. When `petalinux-config` is executed with no other options, it launches the system-level or "generic" menuconfig. This interface allows you to specify information such as the desired boot device or metadata about the system such as default hostname. The `petalinux-config -c kernel`, `petalinux-config -c u-boot`, and `petalinux-config -c rootfs` workflows launch the menuconfig interfaces for customizing the Linux kernel, U-Boot, and the root file system, respectively.

The `--silentconfig` option allows you to restore a prior configuration.

Example:

Execute the following command after enabling or disabling different configs by editing `<proj-root>/project-spec/configs/rootfs_config`

```
$ petalinux-config -c rootfs --silentconfig
```

Use this command when you want to use the existing configurations without editing it. In this case, the menuconfig will not launch.

petalinux-config -c COMPONENT Examples

The following examples demonstrate proper usage of the `petalinux-config -c COMPONENT` command:

- Start the menuconfig for the system-level configuration.

```
$ petalinux-config
```

- Enable different rootfs packages without opening the menuconfig. Execute below command after enabling or disabling different packages by editing `<proj-root>/project-spec/configs/rootfs_config`

```
$ petalinux-config -c rootfs --silentconfig
```

- Load the Linux kernel configuration with a specific default configuration.

```
$ petalinux-config -c kernel --defconfig xilinx_zynq_base_trd_defconfig
```

- Load the U-Boot configuration with a specific default configuration.

```
$ petalinux-config -c u-boot --defconfig xilinx_zynqmp_zcu102_defconfig
```

- Generate the source code for FSBL/fs-boot.

```
$ petalinux-config -c bootloader
```

The following warning message appears when `petalinux-config` or `petalinux-build` for components (for example: `petalinux-build -c u-boot`) is run. This message can be ignored.

WARNING! *SRC_URI is conditionally overridden in this recipe, thus several devtool-override-* branches have been created, one for each override that makes changes to SRC_URI. It is recommended that you make changes to the devtool branch first, then checkout and rebase each devtool-override-* branch and update any unique patches there (duplicates on those branches will be ignored by devtool finish/update-recipe).*

petalinux-build

The `petalinux-build` tool builds either the entire embedded Linux system or a specified component of the Linux system. This tool uses the Yocto Project underneath. Whenever `petalinux-build` is invoked, it internally calls `bitbake`. While the tool provides a single workflow, the specifics of its operation can be dictated using the `petalinux-build -c` and `petalinux-build -x` options.

petalinux-build Command Line Options

The following table outlines the valid options for the `petalinux-build` tool.

Table 41: **petalinux-build Command Line Options**

Option	Functional Description	Value Range	Default Value
`-p,--project PROJECT`	PetaLinux project directory path. This is optional.	User-specified	None
`-c, --component COMPONENT`	Builds specified component. These are the default values which are supported. You can build against your own target (such as your application or module). This is optional.	• bootloader (Zynq® UltraScale+™ MPSoC, Zynq architecture, and MicroBlaze™ CPU) • kernel • u-boot • rootfs • pmufw, only for Zynq UltraScale+ MPSoC • arm-trusted-firmware, for Zynq UltraScale+ MPSoC and Versal™ ACAP. • device-tree • plm, only for Versal ACAP • psmfw, only for Versal ACAP • apps • modules	None
`-x, --execute STEP`	Executes specified build step. All Yocto tasks can be passed through this option. To get all tasks of a component, use “listtasks”. This is optional.	• build • clean • cleanall • cleansstate • distclean • install • listtasks • populate_sysroot • package • mrproper	Build
`-v,--verbose`	Displays additional output messages. This is optional.	None	None
`-s, --sdk`	Builds Yocto SDK. This is optional.	None	None
--esdk	Builds Yocto e-SDK.This is optional.	None	Nnone
`-h`	Lists all the sub-components of a component. Valid only for rootfs. This is optional.	rootfs	None
`-f, --force`	Forces a specific task to run against a component, or a single task in the component, ignoring the stamps. This is optional.	None	None

petalinux-build --component

The `petalinux-build -c` option builds the specified component of the embedded system. When no components are specified, the `petalinux-build` tool operates on the project as a whole. User-created components for the root file system can be built by targeting those components by name (for example, with `-c <APP-NAME>`). This is equivalent to `bitbake <COMPONENT>`. Each recipe can be specified as a component for `petalinux-build -c <component>`. The component can be a user created app or package/package group in the root filesystem.

The `petalinux-build` command with no arguments runs `bitbake petalinux-user-image` internally. The default image target is `petalinux-image-minimal`. There is no restriction on the components, and you can build your own packages. For the names of the packages, search in `petalinux-config -c rootfs`.

An example to build base-files is as follows:

```
petalinux-build -c base-files
```

petalinux-build -c component options

The following table summarizes the available components that can be targeted with this command:

Table 42: **petalinux-build -c Components**

Component	Equivalent Bitbake Commands	Description
bootloader	`bitbake virtual/fsbl` `bitbake virtual/fsboot` (for MicroBlaze™ processor)	Build only the boot loader elf image and copy it into `<plnx-proj-root>/images/linux/`. For Zynq® UltraScale+™ MPSoC and Zynq-7000 devices, it is FSBL and for MicroBlaze™ processor, it is fs-boot.
device tree	`bitbake virtual/dtb`	Build only the device tree DTB file and copy it into `<plnx-proj-root>/images/linux/`. The device tree source is in `<plnx-proj-root>/components/plnx_workspace/device-tree/device-tree/`.
arm-trusted-firmware	`bitbake virtual/arm-trusted-firmware`	Build only the ATF image and copy it into `<plnx-proj-root>/images/linux`.
pmufw	`bitbake virtual/pmufw`	Build only the PMU firmware image and copy it into `<plnx-proj-root>/images/linux`.
kernel	`bitbake virtual/kernel`	Build only the Linux kernel image and copy it into `<plnx-proj-root>/images/linux`.
rootfs	`bitbake petalinux-user-image -c do_image_complete`	Build only the root file system. It generates the target rootfs in `${TMPDIR}/work/${MACHINE}/petalinux-user-image/1.0-r0/rootfs/` and the sysroot in `${TMPDIR}/tmp/sysroots/${MACHINE}`.
u-boot	`bitbake virtual/bootloader`	Build only the U-Boot elf image and copy it into `<plnx-proj-root>/images/linux`.

Table 42: **petalinux-build -c Components** *(cont'd)*

Component	Equivalent Bitbake Commands	Description
plm	`virtual/plm`	Build only the PLM image and copy it into `<plnx-proj-root>/images/linux`.
psmfw	`virtual/psm-firmware`	Build only the PSM firmware image and copy it into `<plnx-proj-root>/image/linux`.
image selector	imgsel	Build only the Image Selector firmware image and copy it into `<plnx-proj-root>/image/linux`.

petalinux-build --execute

The `petalinux-build -x` option allows you to specify a build step to the `petalinux-build` tool to control how the specified components are manipulated. The Yocto task name has a `do_` prefixed to the `petalinux-build` step. All Yocto tasks can be passed through this option. To get all tasks of a component, use `listtasks`.

Commands for petalinux-build -x

The following table summarizes some of the available commands that can be used with this option:

Table 43: **petalinux-build -x options**

Component	Description
`clean`	Cleans build data for the target component.
`cleansstate`/ `distclean`	Removes the shared state cache of the corresponding component.
`cleanall`	Removes the downloads and shared state cache. Cleans the work directory of a component.
`mrproper`	Cleans the build area. This removes the `<plnx-proj-root>/build/`, `<TMPDIR>`, and `<plnx-proj-root>/images/` directories. This is the recommended way of cleaning the entire project.
`build`	Builds the target component.
`install`	Installs the target component. For bootloader, ATF, Linux kernel, U-Boot, and device tree, it copies the generated binary into `<plnx-proj-root>/images/linux/`. For the root file system and root file system component, it copies the generated binary to target the root file system host copy `${TMPDIR}/work/${MACHINE}/petalinux-user-image/1.0-r0/rootfs/`.
`package`	Generates FIT image `image.ub` from build area and copies it into `<plnx-proj-root>/images/linux/`. Valid for -c all or when no component is specified only.
`listtasks`	Gets all tasks of a specific component.

petalinux-build Examples

The following examples demonstrate proper usage of the `petalinux-build` command.

- Clear the build area of the PetaLinux project for archiving as a BSP or for revision control. This example retains the images directory of the project.

```
$ petalinux-build -x distclean
```

- Clean all build collateral from the U-Boot component of the PetaLinux project.

```
$ petalinux-build -c u-boot -x cleansstate
```

- Clean all build collateral. It removes build/, ${TMPDIR} and images. This brings the project to its initial state.

```
$ petalinux-build -x mrproper
```

- Create an updated FIT image from the current contents of the deploy area.

```
$ petalinux-build -x package
```

- Build the entire PetaLinux project.

```
$ petalinux-build
```

- Build the kernel forcefully by ignoring the stamps (output of tasks from last successful build).

```
$ petalinux-build -c kernel -f
```

- Compile kernel forcefully by ignoring do_compile task stamp.

```
$ petalinux-build -c kernel -x compile -f
```

- Build the eSDK and copy it to `<proj_root>/images/linux/esdk.sh`

```
petalinux-build --esdk
```

- Pack all the components of `petalinux-build`.

```
petalinux-build --archiver
```

- Pack only the sysroot components.

```
petalinux-build --sdk --archiver
```

***Note*:** You can find the archiver tar in `<plnx-proj-root>/images/linux`.

petalinux-boot

The `petalinux-boot` command boots MicroBlaze™ CPU, Zynq® devices, Versal™ ACAP, and Zynq® UltraScale+™ MPSoC with PetaLinux images through JTAG/QEMU. This tool provides two distinct workflows:

- In `petalinux-boot --jtag` workflow, images are downloaded and booted on a physical board using a JTAG cable connection.

- In `petalinux-boot --qemu` workflow, images are loaded and booted using the QEMU software emulator.

Either the `--jtag` or the `--qemu` is mandatory for the `petalinux-boot` tool. By default, the `petalinux-boot` tool loads binaries from the `<plnx-proj-root>/images/linux/` directory.

petalinux-boot Command Line Options

The following table details the command line options that are common to all `petalinux-boot` workflows.

Table 44: **petalinux-boot Command Line Options**

Option	Functional Description	Value Range	Default Value
`--jtag`	Use the JTAG workflow. Mutually exclusive with the QEMU workflow. One of the two, `--jtag` or `--qemu` is required.	None	None
`--qemu`	Use the QEMU workflow. Mutually exclusive with the JTAG workflow. One of the two, `--jtag` or `--qemu` is required.	None	None
`--prebuilt`	Boot a prebuilt image. This is optional.	• 1 (bitstream /FSBL) (1) • 2 (U-Boot) • 3 (Linux kernel)	None
`--boot-addr BOOT_ADDR`	Boot address. This is optional.	None	None
`-i,--image IMAGEPATH`	Image to boot. This is optional. To specify U-Boot/Kernel image from an external path, use this option. Example: `$ petalinux-boot --qemu --image ./images/linux/zImage --dtb ./images/linux/system.dtb`	User-specified	None
`--u-boot`	This option can be use to download specified U-Boot binary along with dependent files to boot into the U-Boot. It will select an U-Boot ELF image from `<plnx-proj-root>/images/linux/`. This is optional.	User-specified	`<plnx-proj-root>/images/linux/u-boot.elf`
`--kernel`	This option can be use to download specified kernel binary along with dependent files to boot kernel. This option will pick kernel image from `<plnx-proj-root>/images/linux/`. This is optional.	User-specified	• zImage for Zynq®-7000 devices • Image for Zynq® UltraScale+™ MPSoC, and Versal™ ACAP • image.elf for MicroBlaze™ CPU The default image is in `<plnx-proj-root>/images/linux`.

Table 44: **petalinux-boot Command Line Options** *(cont'd)*

Option	Functional Description	Value Range	Default Value
`-v,--verbose`	Displays additional output messages. This is optional.	None	None
`-h,--help`	Displays tool usage information. This is optional.	None	None

Notes:

1. `--prebuilt` 1 is not a valid option for the QEMU workflow.

petalinux-boot --jtag

The `petalinux-boot --jtag` command boots the MicroBlaze™ CPUs, the Zynq® UltraScale+™ MPSoCs, Zynq-7000 devices, or Versal™ ACAPs with a PetaLinux image using a JTAG connection.

Note: The `petalinux-boot --jtag` command might not work as expected when executed within a virtual machine since virtual machines often have problems with JTAG cable drivers.

petalinux-boot --jtag Options

The following table contains details of options specific to the JTAG boot workflow.

Table 45: **petalinux-boot --jtag Options**

Option	Functional Description	Value Range	Default Value
`--xsdb-conn COMMAND`	Customised XSDB connection command to run prior to boot. This is optional.	User-specified	None
`--hw_server-url URL`	URL of the hw_server to connect to. This is optional.	User-specified	None
`--tcl OUTPUTFILE`	Log JTAG Tcl commands used for boot. This is optional.	User-specified	None

Table 45: **petalinux-boot --jtag Options** *(cont'd)*

Option	Functional Description	Value Range	Default Value
`--fpga` (1)	Program FPGA bitstream. This is optional.	None	If no bitstream is specified with the `--bitstream` option, it uses the bitstream from one of the following locations: • If you are using build images to boot, it will pick the bitstream from `<plnx-proj-root>/images/linux/system.bit` • If you are using prebuilt images to boot, it will pick the bitstream pick `<plnx-proj-root>/prebuilt/linux/implementation/download.bit`
`--bitstream BITSTREAM`	Specify a bitstream. This is optional.	User-specified	None
`--pmufw PMUFW-ELF`	PMU firmware image. This is optional and applicable for Zynq® UltraScale+™ MPSoC. PMU firmware image is loaded by default, unless it is specified otherwise. To skip loading PMU firmware, use `--pmufw no`.	None	`<plnx-proj-root>/images/linux/pmufw.elf`
`before-connect <CMD>`	Extra command to run before XSDB connect command. Ensure the command is properly quoted in your shell. This is optional and can be used multiple times.	None	None
`after-connect <CMD>`	Extra commands to run after XSDB connect command. Ensure the command is properly quoted in your shell. This is optional and can be used multiple times.	None	None

Notes:

1. The `--fpga` option looks for download.bit in `<plnx-proj-root>/pre-built/linux/implementation` by default.

petalinux-boot --jtag Examples

Images for loading on target can be selected from the following:

1. Prebuilt directory: `<plnx-proj-root>/pre-built/linux/images`. These are prebuilt images packed along with the BSP.
2. Images directory: `<plnx-proj-root>/images/linux`. These are the images built by the user.

The following examples demonstrate some use-cases of the `petalinux-boot --jtag` command.

- Download bitstream and FSBL for Zynq-7000 devices, and FSBL and PMU firmware for Zynq UltraScale+ MPSoC

```
$ petalinux-boot --jtag --prebuilt 1
```

Note: Images are taken from `<plnx-proj-root>/pre-built/linux/images` directory.

- Boot U-Boot on target board after loading bitstream/boot loader.

```
$ petalinux-boot --jtag --prebuilt 2
```

Note: Images are taken from `<plnx-proj-root>/pre-built/linux/images` directory.

```
$ petalinux-boot --jtag --u-boot --fpga
```

Note: Images are taken from `<plnx-proj-root>/images/linux` directory.

  - For MicroBlaze™ processors, the above commands download the bitstream to the target board, and then boot the U-Boot on the target board.
  - For Zynq-7000 devices, they download the bitstream and FSBL to the target board, and then boot the U-Boot on the target board.
  - For Zynq UltraScale+ MPSoC, they download the bitstream, PMU firmware, and FSBL, and then boot the U-Boot on the target board.
  - For Versal™ ACAP, they download BOOT.BIN (which contains the PDI, PLM firmware, PSM firmware, U-boot, and DTB) and then boot the U-Boot on the target board.

- Boot prebuilt kernel on target board after loading bitstream, boot loader, and U-Boot.

```
$ petalinux-boot --jtag --prebuilt 3
```

Note: Images are taken from `<plnx-proj-root>/pre-built/linux/images` directory.

```
$ petalinux-boot --jtag --kernel
```

Note: Images are taken from `<plnx-proj-root>/images/linux` directory.

  - For MicroBlaze processors, the above commands download the bitstream to the target board, and then boot the kernel image on the target board.
  - For Zynq-7000 devices, they download the bitstream and FSBL to the target board, and then boot the U-Boot and then the kernel on the target board.
  - For Zynq UltraScale+ MPSoC, they download the bitstream, PMU firmware, and FSBL, and then boot the kernel with help of `linux-boot.elf` to set kernel start and DTB addresses.
  - For Versal™ ACAP, they download the BOOT.BIN (which contains the PDI, PLM firmware, PSM firmware, U-Boot, and DTB) and then boot the kernel (Image) with the help of U-Boot script (`boot.scr`).

petalinux-boot --qemu

The `petalinux-boot --qemu` command boots the MicroBlaze™ CPU, Zynq® UltraScale+™ MPSoC, Versal™ ACAP, or Zynq-7000 devices with a PetaLinux image using the QEMU emulator. Many QEMU options require superuser (root) access to operate properly. The `--root` option enables root mode and prompts you for sudo credentials.

***Note*:** For Versal ACAP, you require BOOT.BIN to boot on QEMU. Check petalinux-package on how to create BOOT.BIN for Versal ACAP.

petalinux-boot --qemu Options

The following table contains details of options specific to the QEMU boot workflow:

Table 46: **petalinux-boot --qemu Options**

Otion	Functional Description	Value Range	Default Value
`--root`	Boot in root mode	None	None
`--iptables-allowed`	Whether to allow to implement iptables commands. This is optional and applicable only in root mode.	None	None
`--net-intf`	Network interface on the host to bridge with the QEMU subnet. This option applies for root mode only.	User-specified	eth0
`--qemu-args`	Extra arguments to QEMU command. This is optional.	None	None
`--subnet SUBNET`	Specifies subnet gateway IP and the number of valid bit of network mask. This option applies for root mode only.	User-specified	192.168.10.1/24
`--dhcpd`	Enable or disable dhcpd. This is optional and applicable only for root mode.	Enable Disable	Enable
`--tftp`	Path to tftp boot directory	User-specified	None
`--pmu-qemu-args`	Extra arguments for PMU instance of QEMU. This is optional.	User-specified	None

petalinux-boot --qemu Examples

The following examples demonstrate proper usage of the `petalinux-boot --qemu` command.

- Load and boot a prebuilt U-Boot elf using QEMU.

```
$ petalinux-boot --qemu --prebuilt 2
```

- Load and boot a prebuilt U-Boot elf using QEMU in root mode.

```
$ petalinux-boot --qemu --root --prebuilt 2
```

petalinux-package

The `petalinux-package` tool packages a PetaLinux project into a format suitable for deployment. The tool provides several workflows whose operations vary depending on the target package format. The supported formats/workflows are `boot`, `bsp`, and `pre-built`. The `petalinux-package` tool is executed using the package type name to specify a specific workflow in the format `petalinux-package --PACKAGETYPE`.

- The `boot` package type creates a file (.BIN or .MCS) that allows the target device to boot.
- The `bsp` package type creates a .bsp file which includes the entire contents of the target PetaLinux project. This option allows you to export and re-use your bsp.
- The `pre-built` package type creates a new directory within the target PetaLinux project called "pre-built" and contains prebuilt content that is useful for booting directly on a physical board. This package type is commonly used as a precursor for creating a `bsp` package type.
- The `image` package type packages image for component with the specified format.
- The `sysroot` package type installs the sysroot for the Vitis™ software platform. It can specify the installer path and also install directory path.

You are required to install Vivado® Design Suite on the same machine as PetaLinux to use `petalinux-boot` for the MCS format for MicroBlaze™ processor. By default, the `petalinux-package` tool loads default files from the `<plnx-proj-root>/images/linux/` directory.

petalinux-package Command Line Options

The following table details the command line options that are common to all of the `petalinux-package` workflows.

Table 47: **petalinux-package Command Line Options**

Option	Functional Description	Value Range	Default Value
`-p, --project PROJECT`	PetaLinux project directory path. This is optional.	User-specified	Current Directory
`-h, --help`	Display usage information. This is optional.	None	None

petalinux-package --boot

The `petalinux-package --boot` command generates a bootable image that can be used directly with Versal™ ACAP, Zynq® UltraScale+™ MPSoC and Zynq-7000 devices, and also with MicroBlaze™-based FPGA designs. For devices in the Zynq series, bootable format is BOOT.BIN which can be booted from an SD card. For MicroBlaze-based designs, the default format is an MCS PROM file suitable for programming using Vivado® Design Suite or other PROM programmer.

For devices in the Zynq series, this workflow is a wrapper around the bootgen utility provided with the Vitis software platform. For MicroBlaze-based FPGA designs, this workflow is a wrapper around the corresponding Vivado Tcl commands and generates an MCS formatted programming file. This MCS file can be programmed directly to a target board and then booted.

petalinux-package --boot Command Options

The following table details the options that are valid when creating a bootable image with the `petalinux-package --boot` command:

Table 48: **petalinux-package --boot Command Options**

Option	Functional Description	Value Range	Default Value
`--format FORMAT`	Image file format to generate. This is optional.	• BIN • MCS • DOWNLOAD.BIT	BIN
`--fsbl FSBL`	Path on disk to FSBL elf binary. This is required. To skip loading FSBL, use `--fsbl no` or `--fsbl none`. This is optional.	User-specified	• `zynqmp_fsbl.elf` for Zynq® UltraScale+™ MPSoC • `zynq_fsbl.elf` for Zynq-7000 device • `fs-boot.elf` for MicroBlaze™ processor The default image is in `<plnx-proj-root>/images/linux`.
`--force`	Overwrite existing files on disk. This is optional.	None	None
`--fpga BITSTREAM`[1]	Path on disk to bitstream file. This is optional.	User-specified	`<plnx-proj-root>/images/linux/system.bit`
`--atf ATF-IMG`	Path on disk to Arm® trusted firmware elf binary. This is optional. To skip loading ATF, use `--atf no` or `--atf none`	User-specified	`<plnx-proj-root>/images/linux/bl31.elf`

Table 48: **petalinux-package --boot Command Options** *(cont'd)*

Option	Functional Description	Value Range	Default Value
`--u-boot UBOOT-IMG`	Path on disk to U-Boot binary. This is optional.	User-specified	• `u-boot.elf` for Zynq device • `u-boot-s.bin` for MicroBlaze CPUs The default image is in `<plnx-proj-root>/images/linux`
`--kernel KERNEL-IMG`	Path on disk to Linux kernel image. This is optional.	User-specified	`<plnx-proj-root>/images/linux/image.ub`
`--boot-script BOOT-SCRIPT`	Path to the `boot.scr` file location. This is optional.	User-specified	`<plnx-proj-root>/images/linux/boot.scr`
`--qemu-rootfs ROOTFS-CPIO-FILE`	Path to the rootfs file location to create `qemu_boot.img` (`cpio.gz.u-boot`). Only valid for the Versal ACAP to generate the QEMU SD image.	User specified	`<plnx-proj-root>/images/linux/rootfs.cpio.gz.u-boot`
`--pmufw PMUFW-ELF`	Optional and applicable only for Zynq® UltraScale+™ MPSoC. By default, prebuilt PMU firmware image is packed. Use this option to either specify a path for PMU firmware image or to skip packing of PMU firmware. To skip packing PMU firmware, use `--pmufw no`.	User-specified	`<plnx-proj-root>/images/linux/pmufw.elf`
`--plm PLM-ELF`	Optional and applicable only for Versal™ ACAP. By default, prebuilt PLM image is packed. Use this option to either specify a path for PLM image or to skip packing of PLM. To skip packing PLM, use `--plm no`.	User-specified	`<plnx-proj-root>/images/linux/plm.elf`
`--psmfw PSMFW-ELF`	Optional and applicable only for Versal ACAP. By default, prebuilt PSM firmware image is packed. Use this option to either specify a path for PSM firmware image or to skip packing of PSM firmware. To skip packing PSM firmware, use `--psmfw no`.	User-specified	`<plnx-proj-root>/images/linux/psmfw.elf`
`--addcdo CDOFILE`	Path on disk to add `.cdo` file pack into BOOT.BIN.	User-specified	None
`--add DATAFILE`	Path on disk to arbitrary data to include. This is optional.	User-specified	None
`--offset OFFSET`	Offset at which to load the prior data file. Only the `.elf` files are parsed. This is optional.	User-specified	None
`--load <LOADADDR>`	Load address for specified data file. The RAM address where to load the specified data file. Example: `[ partition_type=raw, load=0x01000 ] <image>`	User-specified	None

Table 48: **petalinux-package --boot Command Options** *(cont'd)*

Option	Functional Description	Value Range	Default Value
`--mmi MMIFILE`	Bitstream MMI file, valid for MicroBlaze CPUs only. It will be used to generate the `download.bit` with boot loader in the block RAM. Default will be the MMI file in the same directory as the FPGA bitstream. This is optional	User-specified	MMI in directory with FPGA bitstream
`--flash-size SIZE`	Flash size in MB. Must be a power-of-2. Valid for MicroBlaze processor only. Not needed for parallel flash types. Ensure you just pass digit value to this option. Do not include MB in the value. This is optional.	User-specified	Auto-detect from system configuration. If it is not specified, the default value is 16.
`--flash-intf INTERFACE`	Valid for MicroBlaze processor only. This is optional.	• SERIALx1 • SPIx1 • SPIx2 • SPIx4 • BPIx8 • BPIx16 • SMAPx8 • SMAPx16 • SMAPx32	Auto-detect
`-o, --output OUTPUTFILE`	Path on disk to write output image. This is optional.	User-specified	None
`--cpu DESTINATION CPU`	Zynq UltraScale+ MPSoC only. The destination CPU of the previous data file. This is optional.	• a53-0 • a53-1 • a53-2 • a53-3	None
`--file-attribute DATA File ATTR`	Zynq-7000, Zynq® UltraScale+™ MPSoC, and Versal ACAP only. Data file file-attribute. This is optional. Example: `petalinux-package --boot --u-boot --kernel images/linux/Image --offset 0x01e40000 --file-attribute partition_owner=uboot --add images/linux/system.dtb --offset 0x3AD1200 --file-attribute partition_owner=uboot --fpga`	User-specified	None
`--bif-attribute ATTRIBUTE`	Zynq-7000, Zynq® UltraScale+™ MPSoC, and Versal ACAP only. Example: `petalinux-package --boot --bif-attribute fsbl_config --bif-attribute-value a53_x64 --u-boot`	User-specified	None
`--bif-attribute-value VALUE`	Zynq-7000, Zynq® UltraScale+™ MPSoC, and Versal ACAP only. The value of the attribute specified by `--bif-attribute` argument. This is optional. Example: `petalinux-package --boot --bif-attribute fsbl_config --bif-attribute-value a53_x64 --u-boot`	User-specified	None

Table 48: **petalinux-package --boot Command Options** *(cont'd)*

Option	Functional Description	Value Range	Default Value
`--fsblconfig BIF_FSBL_CONFIG`	Zynq® UltraScale+™ MPSoC only. BIF FSBL config value. Example: `petalinux-package --boot --fsblconfig a53_x64 --u-boot`	User-specified	None
`--bif BIF FILE`	Zynq-7000 devices, Zynq UltraScale+ MPSoC, and Versal ACAP. BIF file. For Zynq-7000 devices and Zynq UltraScale+ MPSoC, it overrides the following settings: • `-fsbl` • `-fpga` • `-u-boot` • `-add` • `-fsblconfig` • `-file-attribute` • `-bif-attribute` • `-bif-attribute-value` For Versal ACAP, it overrides the following settings: • `-fpga` • `-u-boot` • `-add` • `-file-attribute` • `-bif-attribute` • `-bit-attribute-value` This is optional.	User-specified	None
`--boot-device BOOT-DEV`	Zynq-7000, Zynq UltraScale+ MPSoC, and Versal ACAP. The boot device is updated in bootargs to boot. This is optional.	• sd • flash	Default value is the one selected from the system select menu of boot image settings.
`--bootgen-extra-args ARGS`	Zynq-7000, Zynq UltraScale+ MPSoC, and Versal ACAP only. Extra arguments to be passed while invoking bootgen command. This is optional.	User-specified	None

Notes:

1. When the `petalinux-config` option is enabled in the FPGA manager, the `--fpga` option cannot be used. Bitstream will not be included in the BOOT.BIN.

petalinux-package --boot Examples

The following examples demonstrate proper usage of the `petalinux-package --boot` command.

- Create a BOOT.BIN file for a Versal™ device.

```
$ petalinux-package --boot --format BIN --plm --psmfw --u-boot --dtb -o
<PATH-TO-OUTPUT-WITH-FILE-NAME>
```

It will generate `BOOT.BIN`, `BOOT_bh.bin`, and `qemu_boot.img` in `images/linux` directory. The default DTB load address will be 0x1000. For more information, see *Bootgen User Guide* (UG1283).

```
$ petalinux-package --boot --plm <PLM_ELF> --psmfw <PSMFW_ELF> --u-boot --
dtb --load <load_address>
```

It will generate a BOOT.BIN with a specified load address for DTB.

Note: The files `versal-qemu-multiarch-pmc.dtb` and `versal-qemu-multiarch-ps.dtb` are qemu dtbs required to boot multi arch qemu. You need to use `system.dtb` for `--dtb` option, generated in `image/linux/` directory or simply use `--dtb` option it will pick from there.

- Create a BOOT.BIN file for a Zynq® device (including Zynq-7000 and Zynq® UltraScale+™ MPSoC).

```
$ petalinux-package --boot --format BIN --fsbl <PATH-TO-FSBL> --u-boot -o
<PATH-TO-OUTPUT-WITH-FILE-NAME>
```

- Create a BOOT.BIN file for a Zynq device that includes a PL bitstream and FITimage.

```
$ petalinux-package --boot --format BIN --fsbl <PATH-TO-FSBL> --u-boot --
fpga <PATH-TO-BITSTREAM> --kernel -o <PATH-TO-OUTPUT>
```

- Create a x8 SMAP PROM MCS file for a MicroBlaze™ CPU design.

```
$ petalinux-package --boot --format MCS --fsbl <PATH-TO-FSBL> --u-boot --
fpga <PATH-TO-BITSTREAM> --flash-size <SIZE> --flash-intf SMAPx8 -o
<PATH-TO-OUTPUT-WITH-FILE-NAME>
```

- Create a BOOT.BIN file for a Zynq UltraScale+ MPSoC that includes PMU firmware.

```
$ petalinux-package --boot --u-boot --kernel --pmufw <PATH_TO_PMUFW>
```

- Create bitstream file download.bit for a MicroBlaze CPU design.

```
$ petalinux-package --boot --format DOWNLOAD.BIT --fpga <BITSTREAM> --fsbl
<FSBOOT_ELF>
```

petalinux-package --bsp

The `petalinux-package --bsp` command compiles all contents of the specified PetaLinux project directory into a BSP file with the provided file name. This .bsp file can be distributed and later used as a source for creating a new PetaLinux project. This command is generally used as the last step in producing a project image that can be distributed to other users. All Xilinx® reference BSPs for PetaLinux are packaged using this workflow.

petalinux-package --bsp Command Options

The following table details the options that are valid when packaging a PetaLinux BSP file with the `petalinux-package --bsp` command.

Table 49: **petalinux-package --bsp Command Options**

Option	Functional Description	Value Range	Default Value
-o, --output BSPNAME	Path on disk to store the BSP file. File name is of the form BSPNAME.bsp. This is required.	User-specified	None
-p,--project PROJECT	PetaLinux project directory path. In the BSP context, multiple project areas can be referenced and included in the output BSP file. This is optional.	User-specified	None
--force	Overwrite existing files on disk. This is optional.	None	None
--clean	Clean the hardware implementation results to reduce package size. This is optional.	None	None
--hwsource HWPROJECT	Path to a Vivado® design tools project to include in the BSP file. Vivado® hardware project will be added to the hardware directory of the output BSP. This is optional.	None	None
--exclude-from-file EXCLUDE_FILE	Excludes the files mentioned in EXCLUDE_FILE from BSP.	User-specified	None

petalinux-package --bsp Command Examples

The following examples demonstrate the proper usage of the `petalinux-package --bsp` command.

- Clean the project and then generate the BSP installation image (.bsp file).

```
$ petalinux-package --bsp --clean -o <PATH-TO-BSP> -p <PATH-TO-PROJECT>
```

- Generate the BSP installation image that includes a reference hardware definition.

```
$ petalinux-package --bsp --hwsource <PATH-TO-HW-EXPORT> -o <PATH-TO-BSP>
-p <PATH-TO-PROJECT>
```

- Generate the BSP installation image from a neutral location.

```
$ petalinux-package --bsp -p <PATH-TO-PROJECT> -o <PATH-TO-BSP>
```

- Generate the BSP installation image excluding some files.

```
$ petalinux-package --bsp -p <path_to_project> -o <path_to_bsp> --exclude-
from-file <EXCLUDE_FILE>
```

petalinux-package --prebuilt

The `petalinux-package --prebuilt` command creates a new directory named “pre-built” inside the directory hierarchy of the specified PetaLinux project. This directory contains the required files to facilitate booting a board immediately without completely rebuilding the project. This workflow is intended for those who will later create a PetaLinux BSP file for distribution using the `petalinux-package --bsp` workflow. All Xilinx® reference PetaLinux BSPs contain a prebuilt directory.

petalinux-package --prebuilt Command Options

The following table details the options that are valid when including prebuilt data in the project with the `petalinux-package --prebuilt` workflow.

Table 50: **petalinux-package --prebuilt Command Options**

Options	Functional Description	Value Range	Default Value
`-p,--project PROJECT`	PetaLinux project directory path. This is optional.	User-specified	Current Directory
`--force`	Overwrite existing files on disk. This is optional.	None	None
`--clean`	Remove all files from the `<plnx-proj-root>/prebuilt` directory. This is optional.	None	None
`--fpga BITSTREAM`	Include the BITSTREAM file in the prebuilt directory. This is optional.	User-specified	`<project>/images/linux/*.bit`
`-a,--add src:dest`	Add the file/directory specified by src to the directory specified by dest in the prebuilt directory. This is optional and can be used multiple times.	User-specified	`The default dest path is <project>/prebuilt/linux`

petalinux-package --prebuilt Command Examples

The following examples demonstrate proper usage of the `petalinux-package --prebuilt` command.

- Include a specific bitstream in the prebuilt area.

```
$ petalinux-package --prebuilt --fpga <BITSTREAM>
```

- Include a specific data file in the prebuilt area. For example, add a custom readme to the prebuilt directory.

```
$ petalinux-package --prebuilt -a <Path to readme>:images/<custom readme>
```

petalinux-package --sysroot

The `petalinux-package --sysroot` command installs an SDK to a specified directory in publish mode. This directory can be used as sysroot for application development.

petalinux-package --sysroot Command Options

The following table details the options that are valid when installing an SDK with the `petalinux-package --sysroot` workflow. The SDK must previously have been published using the `petalinux-build --sdk` command.

Table 51: **petalinux-package --sysroot Command Options**

Options	Functional Description	Value Range	Default Value
`-p,--project PROJECT`	PetaLinux project directory path. This is optional.	User-specified	Current Directory
`-s, --sdk SDK`	SDK path on disk to SDK .sh file. This is optional.	None	`<plnx-proj-root>/images/linux/sdk.sh`
`-d, --dir DIRECTORY`	Directory path on disk to install SDK. This is optional.	None	`<plnx-proj-root>/images/linux/sdk`

petalinux-package --sysroot Command Examples

The following examples demonstrate the proper usage of the `petalinux-package --sysroot` command.

- Install default SDK to default directory.

```
$ petalinux-package --sysroot
```

- Install specified SDK to default directory.

```
$ petalinux-package --sysroot -s <PATH-TO-SDK>
```

- Install specified SDK to specified directory.

```
$ petalinux-package --sysroot -s <PATH-to-SDK> -d <PATH-TO-INSTALL-DIR>
```

petalinux-package --wic

The following command generates partitioned images from the `images/linux` directory. Image generation is driven by partitioning commands contained in the kickstart file (.wks). The default .wks file is FAT32 with 1 GB and EXT4 with 3 GB. You can find the default kickstart file in `<project-proot>/build/wic/rootfs.wks` after the `petalinux-package --wic` command is executed.

```
$ petalinux-package --wic
```

petalinux-package --wic Command Options

Table 52: **petalinux-package --wic Command Options**

Options	Functional Description	Value Range	Default Value
`-w, --wks <WKS_FILE>`	Specify the `wic` file to be used to create partitions of SD card.	None	None
`-o, --outdir <OUTPUT_DIR>`	Specify the output directory to create `petalinux-sdimage.wic`.	None	Default: `<proot>/images/linux`.
`-i, --images-dir <IMAGES_DIR>`	Specify the images directory the boot files were located.	None	Default: `<proot>/images/linux`.
`-c, --rootfs-file <ROOTFS_FILE>`	Specify the compressed root filesystem file that will be extracted into the `/rootfs` directory.	None	Default: `<proot>/images/linux/rootfs.tar.gz` Supports only the tar.gz format.
`-r, --rootfs-dir <ROOTFS_DIR>`	Specify the extracted root filesystem directory that will be copied to `/rootfs` directory.	None	None
`-b, --bootfiles <BOOT_FILES>`	Specify boot files which should be copied into `/boot` directory.	None	Default boot files: For Zynq-7000 devices: BOOT.BIN, uImage, boot.scr. For Zynq UltraScale+ MPSoC: BOOT.BIN,Image, boot.scr, and ramdisk.cpio.gz For Versal ACAP: BOOT.BIN, Image, boot.scr, and ramdisk.cpio.gz.u-boot
`-e, --extra-bootfiles <EXTRA_FILES>`	Specify extra boot files which should be copied into `/boot` directory. Make sure these are part of images directory.	None	None

petalinux-package --wic Command Examples

Packaging the WIC Image using Default Images

The following command generates the wic image, `petalinux-sdimage.wic`, in the `images/linux` folder with the default images from the `images/linux` directory.

```
$ petalinux-package --wic
```

Packaging the WIC Image in a Specific Folder

The following command generates the wic image, `petalinux-sdimage.wic`, in the `wicimage/` folder.

```
$ petalinux-package --wic --outdir wicimage/
```

Packaging the WIC Image with Specified Images Path

The following command packs all bootfiles from the `custom-imagespath/` directory.

```
$ petalinux-package --wic --images-dir custom-imagespath/
```

Packaging Custom Bootfiles into the /boot Directory

- To copy `boot.bin userfile1 userfile2` files from the `<plnx-proj-root>/images/linux` directory to the `/boot` of media, use the following command:

  ```
  $ petalinux-package --wic --bootfiles "boot.bin userfile1 userfile2"
  ```

  This generates the wic image with specified files copied into the `/boot` directory.

  Note: Ensure that these files are part of the `images` directory.

- To copy the `uImage` file named kernel to the `/boot` directory, use the following command:

  ```
  $ petalinux-package --wic --extra-bootfiles "uImage:kernel"
  ```

- To copy the default bootfiles and specified bootfiles by user files into the `/boot` directory, use the following command:

  ```
  $ petalinux-package --wic --bootfiles "userfiles/*"
  ```

- To copy all the files in the `userfiles/` directory to the `/boot/user_boot` directory, use the following command:

  ```
  $ petalinux-package --wic --extra-bootfiles "userfiles/*:user_boot"
  ```

  Note: Ensure that these files are part of the `images` directory.

Packaging Custom Root File System

The following command unpacks your `custom-rootfs.tar.gz` file and copies it to the `/rootfs` directory.

```
$ petalinux-package --wic --rootfs-file custom-rootfs.tar.gz
```

Customizing WIC Partitions

PetaLinux uses the kickstart (.wks) file to define the partitions to create the WIC image. When the `petalinux-package --wic` command is executed, the default .wks file is placed in `<plnx-proj-root>/build/wic/rootfs.wks`. This can be modified and provided as an input to create the WIC images as per the requirement. Following is the default `rootfs.wks` file:

```
part /boot --source bootimg-partition --ondisk mmcblk0 --fstype=vfat --
label boot --active --align 4 --size 800
part / --source rootfs --ondisk mmcblk0 --fstype=ext4 --label root --align
4 --size 2400
```

You can refer to https://www.yoctoproject.org/docs/latest/ref-manual/ref-manual.html#ref-kickstart for each argument specified in the wks file.

Once you have the updated .wks file, use the following command to create WIC image:

```
$ petalinux-package --wic --wks <path to the wks file>
```

petalinux-util

The `petalinux-util` tool provides various support services to the other PetaLinux workflows. The tool itself provides several workflows depending on the support function needed.

petalinux-util --gdb

The `petalinux-util --gdb` command is a wrapper around the standard GNU GDB debugger and simply launches the GDB debugger in the current terminal. Executing `petalinux-util --gdb --help` at the terminal prompt provides verbose GDB options that can be used.

For GDB GUI-based debugging, use the Vitis™ software platform. For more information regarding GDB, see *Vitis Unified Software Platform Documentation: Embedded Software Development* (UG1400).

petalinux-util --gdb command Examples

The following example demonstrates proper usage of the `petalinux-util --gdb` command. To launch the GNU GDB debugger, use the following command:

```
$ petalinux-util --gdb
```

petalinux-util --dfu-util

The `petalinux-util --dfu-util` command is a wrapper around the standard `dfu-util`, and launches `dfu-util` in the current terminal. Executing `petalinux-util --dfu-util --help` at the terminal prompt provides verbose dfu-util options that can be used.

petalinux-util --dfu-util Command Examples

The following example demonstrates proper usage of the `petalinux-util --dfu-util` command. To launch the `dfu-util`, use the following command:

```
$ petalinux-util --dfu-util
```

petalinux-util --xsdb-connect

The `petalinux-util --xsdb-connect` command provides XSDB connection to QEMU. This is for Zynq® UltraScale+™ MPSoC and Zynq-7000 devices only.

For more information regarding XSDB, see *Vitis Unified Software Platform Documentation: Embedded Software Development* (UG1400).

petalinux-util --xsdb-connect Options

The following table details the options that are valid when using the `petalinux-util --xsdb-connect` command.

Table 53: **petalinux-util --xsdb-connect Options**

Option	Functional Description	Value Range	Default Value
`--xsdb-connect HOST:PORT`	Host and the port XSDB should connect to. This should be the host and port that QEMU has opened for GDB connections. It can be found in the QEMU command line arguments from: --gdb tcp: <QEMU_HOST>: <QEMU_PORT>. This is required.	User-specified	None

petalinux-util --jtag-logbuf

The `petalinux-util --jtag-logbuf` command logs the Linux kernel printk output buffer that occurs when booting a Linux kernel image using JTAG. This workflow is intended for debugging the Linux kernel for review and debug. This workflow can be useful when the Linux kernel is not producing output using a serial terminal. For details on how to boot a system using JTAG, see the `petalinux-boot --jtag` command. For MicroBlaze™ CPUs, the image that can be debugged is `<plnx-proj-root>/image/linux/image.elf`. For Arm® cores, the image that can be debugged is `<plnx-proj-root>/image/linux/vmlinux`.

petalinux-util --jtag-logbuf Options

The following table details the options that are valid when using the `petalinux-util --jtag-logbuf` command.

Table 54: **petalinux-util --jtag-logbuf Options**

Option	Functional Description	Value Range	Default Value
`-i,--image IMAGEPATH`	Linux kernel ELF image. This is required.	User-specified	None
`--hw_server-url URL`	URL of the hw_server to connect to. This is optional.	User-specified	None
`-p,--project PROJECT`	PetaLinux project directory path. This is optional.	User-specified	Current Directory

Table 54: **petalinux-util --jtag-logbuf Options** *(cont'd)*

Option	Functional Description	Value Range	Default Value
`--noless`	Do not pipe output to the less command. This is optional.	None	None
`-v,--verbose`	Displays additional output messages. This is optional.	None	None
`-h,--help`	Displays tool usage information. This is optional.	None	None
`--dryrun`	Prints the commands required to extract the kernel log buffer, but do not run them.	None	None

petalinux-util --jtag-logbuf Examples

The following examples demonstrate proper usage of the `petalinux-util --jtag-logbuf` command.

- Launch a specific Linux kernel image

```
$ petalinux-util --jtag-logbuf -i <PATH-TO-IMAGE>
```

- Launch the JTAG logger from a neutral location. This workflow is for Zynq®-7000 devices only

```
$ petalinux-util --jtag-logbuf -i <PATH-TO-IMAGE> -p <PATH-TO-PROJECT>
```

petalinux-util --find-xsa-bitstream

The `petalinux-util --find-xsa-bitstream` gives the name of bitstream packed in the XSA file.

petalinux-util --find-xsa-bitstream Options

The following table details the options that are valid when using the `petalinux-util --find-xsa-bitstream` command.

Table 55: **petalinux-util --find-xsa-bitstream Options**

Option	Functional Description	Value Range	Default Value
`--xsa-file <XSA>`	Argument to specify the XSA file to use. This is optional.	None	system.xsa file in the `<project>/project-spec/hw-description` directory

petalinux-util -- find-xsa-bitstream Examples

The following examples demonstrate proper usage of the `petalinux-util --find-xsa-bitstream` command.

- To find the default bitstream of a project

```
petalinux-util --find-xsa-bitstream
```

- To find the bitstream of a xsa

```
petalinux-util --find-xsa-bitstream --xsa-file <path to xsa file>
```

petalinux-util --webtalk

The `petalinux-util --webtalk` command sets the Xilinx® WebTalk feature ON or OFF. Xilinx WebTalk provides anonymous usage data about the various PetaLinux tools to Xilinx. A working internet connection is required for this feature to work when enabled.

petalinux-util --webtalk Options

The following table details the options that are valid when using the `petalinux-util --webtalk` command.

Table 56: **petalinux-util --webtalk Options**

Option	Functional Description	Value Range	Default Value
`--webtalk`	Toggle WebTalk. This is required.	• On • Off	Off
`-h,--help`	Display usage information. This is optional.	None	None

petalinux-util --webtalk Examples

The following examples demonstrate proper usage of the `petalinux-util --webtalk` command.

- Turn the WebTalk feature off

```
$ petalinux-util --webtalk off
```

- Turn the WebTalk feature on

```
$ petalinux-util --webtalk on
```

petalinux-upgrade

To upgrade the workspace, use the `petalinux-upgrade` command.

petalinux-upgrade Options

Table 57: **petalinux-upgrade Options**

Options	Functional description	Value Range	Default Range
`-h --help`	Displays usage information.	None	None
`-f --file`	Local path to target system software components.	User-specified.	None
`-u --url`	URL to target system software components.	User-specified.	None
`-w, --wget-args`	Passes additional wget arguments to the command.	Additional wget options.	None
`-p|--platform`	Specifies the architecture name to upgrade.	aarch64: sources for Zynq UltraScale+ MPSoC and Versal arm: sources for Zynq devices microblaze_lite: sources for microblaze_lite microblaze_full: sources for microblaze_full	None

petalinux-devtool

The `petalinux-devtool` is a utility that uses the Yocto devtool to enable you to build, test, and package software. The following table details the available options for the `petalinux-devtool` command.

petalinux-devtool Command Line Options

Table 58: **petalinux-devtool Command Line Options**

Option	Functional Description
add	Add a new recipe
modify	Modify the source for an existing recipe
upgrade	Upgrade an existing recipe
status	Show workspace status
search	Search available recipes
latest-version	Report the latest version of an existing recipe
check-upgrade-status	Report upgradability for multiple (or all) recipes
build	Build a recipe
rename	Rename a recipe file in the workspace
edit-recipe	Edit a recipe file
find-recipe	Find a recipe file
configure-help	Get help on configure script options

Table 58: **petalinux-devtool Command Line Options** *(cont'd)*

Option	Functional Description
update-recipe	Apply changes from external source tree to recipe
reset	Remove a recipe from your workspace
finish	Finish working on a recipe in your workspace
deploy-target	Deploy recipe output files to live target machine
undeploy-target	Undeploy recipe output files in live target machine
build-image	Build image including workspace recipe packages
create-workspace	Set up workspace in an alternative location
configure	Runs configure command
export	Export workspace into a tar archive
extract	Extract the source for an existing recipe
sync	Synchronize the source tree for an existing recipe
import	Import exported tar archive into workspace
menuconfig	Alter build-time configuration for a recipe

petalinux-devtool Examples

Adding a New Recipe to the Workspace Layer

To add a new recipe to the workspace layer, use the `petalinux-devtool add` command. For `$ petalinux-devtool add bbexample https://github.com/whbruce/bbexample.git` This command fetches the source from the specified URL and creates the recipe bbexample in the Devtool Workspace directory.

petalinux-devtool add

```
$.  petalinux-devtool modify linux-xlnx
[INFO] Sourcing buildtools
[INFO] Sourcing build environment
[INFO] Generating workspace directory
[INFO] devtool modify linux-xlnx
NOTE: Starting bitbake server...
NOTE: Started PRServer with DBfile: <plnx-proj-root>/xilinx-vck190-2021.1/
build/cache/prserv.sqlite3, IP: 127.0.0.1, PORT: 34999, PID: 30957
NOTE: Reconnecting to bitbake server...
NOTE: Retrying server connection (#1)...
NOTE: Started PRServer with DBfile: <plnx-proj-root>/xilinx-vck190-2021.1/
build/cache/prserv.sqlite3, IP: 127.0.0.1, PORT: 44777, PID: 31500
Loading cache: 100% |
##################################################################################
##############################################################| Time:
0:00:01
Loaded 5103 entries from dependency cache.
Parsing recipes: 100% |
##################################################################################
##############################################################| Time: 0:00:01
Parsing of 3477 .bb files complete (3468 cached, 9 parsed). 5112 targets,
243 skipped, 0 masked, 0 errors.
```

```
NOTE: Resolving any missing task queue dependencies
Initialising tasks: 100% |
##################################################################################
##########################################################| Time: 0:00:00
Sstate summary: Wanted 2 Found 0 Missed 2 Current 103 (0% match, 98%
complete)
NOTE: Executing Tasks
NOTE: Tasks Summary: Attempted 474 tasks of which 474 didn't need to be
rerun and all succeeded.
INFO: Copying kernel config to workspace
INFO: Recipe linux-xlnx now set up to build from <plnx-proj-root>/xilinx-
vck190-2021.1/components/yocto/workspace/sources/linux-xlnx
```

Getting the Status of the Recipes in Your Workspace

Use the `petalinux-devtool status` command to list the recipes currently in your workspace. Information includes the paths to their respective external source trees.

petalinux-devtool status

```
$ petalinux-devtool build linux-xlnx
[INFO] Sourcing buildtools
[INFO] Sourcing build environment
[INFO] Generating workspace directory
[INFO] devtool build linux-xlnx
NOTE: Starting bitbake server...
NOTE: Started PRServer with DBfile <plnx-proj-root>/xilinx-vck190-2021.1/
build/cache/prserv.sqlite3, IP: 127.0.0.1, PORT: 38785, PID: 19093
NOTE: Reconnecting to bitbake server...
NOTE: Retrying server connection (#1)...
NOTE: Started PRServer with DBfile: <plnx-proj-root>/xilinx-vck190-2021.1/
build/cache/prserv.sqlite3, IP: 127.0.0.1, PORT: 38405, PID: 19133
Loading cache: 100% |
##################################################################################
##########################################################| Time:
0:00:01
Loaded 5103 entries from dependency cache.
Parsing recipes: 100% |
##################################################################################
##########################################################| Time: 0:00:40
Parsing of 3477 .bb files complete (3467 cached, 10 parsed). 5112 targets,
243 skipped, 0 masked, 0 errors.
Removing 1 recipes from the versal_generic sysroot: 100% |
##################################################################################
#########################| Time: 0:00:07
NOTE: Started PRServer with DBfile: <plnx-proj-root>/xilinx-vck190-2021.1/
build/cache/prserv.sqlite3, IP: 127.0.0.1, PORT: 35773, PID: 24137
Loading cache: 100% |
##################################################################################
##########################################################| Time:
0:00:03
Loaded 5103 entries from dependency cache.
Parsing recipes: 100% |
##################################################################################
##########################################################| Time: 0:00:28
Parsing of 3477 .bb files complete (3467 cached, 10 parsed). 5112 targets,
243 skipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
Initialising tasks: 100% |
##################################################################################
##########################################################| Time: 0:00:00
```

```
Sstate summary: Wanted 6 Found 0 Missed 6 Current 140 (0% match, 95%
complete)
NOTE: Executing Tasks
NOTE: linux-xlnx: compiling from external source tree <plnx-proj-root>/
xilinx-vck190-2021.1/components/yocto/workspace/sources/linux-xlnx
NOTE: Tasks Summary: Attempted 658 tasks of which 624 didn't need to be
rerun and all succeeded.
```

Modifying an Existing Recipe

Use the `petalinux-devtool modify` command to begin modifying the source of an existing recipe. This command extracts the source for a recipe to the Devtool Workspace directory, checks out a branch for development, and applies the patches, if any, from the recipe as commits on top.

Use the following command to modify the linux-xlnx recipe:

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

Figure 22: **petalinux-devtool modify**

```
:petalinux-devtool modify linux-xlnx
INFO: Sourcing build tools
[INFO] Sourcing build environment
[INFO] Generating workspace directory
[INFO] devtool modify linux-xlnx
NOTE: Starting bitbake server...
...
INFO: Adding local source files to srctree...
INFO: Copying kernel config to srctree
INFO: Source tree extracted to <plnx-proj-dir>/components/yocto/workspace/sources/linux-xlnx
WARNING: SRC_URI is conditionally overridden in this recipe, thus several devtool-override-* branches have been created,
one for each override that makes changes to SRC_URI. It is recommended that you make changes to the devtool branch first,
then checkout and rebase each devtool-override-* branch and update any unique patches there (duplicates on those branches
will be ignored by devtool finish/update-recipe)
INFO: Recipe linux-xlnx now set up to build from <plnx-proj-dir>/components/yocto/workspace/sources/linux-xlnx
```

Building the Recipe

Use the `petalinux-devtool build` command to build your recipe. This command is equivalent to the `bitbake -c populate_sysroot` command. You must supply the root name of the recipe (i.e., do not provide versions, paths, or extensions), as shown:

```
$ petalinux-devtool build linux-xlnx
```

Figure 23: **petalinux-devtool build**

```
:petalinux-devtool build linux-xlnx
INFO: Sourcing build tools
[INFO] Sourcing build environment
[INFO] Generating workspace directory
[INFO] devtool build linux-xlnx
NOTE: Starting bitbake server...
...
NOTE: Executing Tasks
NOTE: Setscene tasks completed
NOTE: linux-xlnx: compiling from external source tree <plnx-proj-dir>/components/yocto/workspace/sources/linux-xlnx
NOTE: Tasks Summary: Attempted 2762 tasks of which 2598 didn't need to be rerun and all succeeded.
```

Building Your Image

Use the `petalinux-devtool build-image` command to build an image using the devtool flow. This will include the recipes which are the workspace directory. When running this command you must specify the image name to be built.

```
$ petalinux-devtool build-image petalinux-image-minimal
```

Figure 24: **petalinux-devtool build-image**

```
:petalinux-devtool build-image petalinux-image-minimal
INFO: Sourcing build tools
[INFO] Sourcing build environment
[INFO] Generating workspace directory
[INFO] devtool build-image petalinux-image-minimal
NOTE: Starting bitbake server...
...
NOTE: Executing Tasks
NOTE: Setscene tasks completed
NOTE: bbexample: compiling from external source tree <plnx-proj-dir>/components/yocto/workspace/sources/bbexample
NOTE: Tasks Summary: Attempted 3623 tasks of which 2805 didn't need to be rerun and all succeeded.
INFO: Successfully built petalinux-image-minimal. You can find output files in <plnx-proj-dir>/build/tmp/deploy/images/zynqmp-generic
```

Resetting the Recipe

Use the `petalinux-devtool reset` command to remove the recipe and its configurations from Devtool Workspace directory. This command does not add/append changes in the Devtool Workspace to the any layers, you must need to update the recipe append file before running the reset command.

```
$ petalinux-devtool reset linux-xlnx
```

Figure 25: **petalinux-devtool reset**

```
:petalinux-devtool reset linux-xlnx
INFO: Sourcing build tools
[INFO] Sourcing build environment
[INFO] Generating workspace directory
[INFO] devtool reset linux-xlnx
NOTE: Starting bitbake server...
INFO: Cleaning sysroot for recipe linux-xlnx...
INFO: Leaving source tree <plnx-proj-dir>/components/yocto/workspace/sources/linux-xlnx as-is; if you no longer need it then please delete it manually
```

***Note*:** You can find a list of examples at https://www.yoctoproject.org/docs/latest/ref-manual/ref-manual.html#ref-devtool-reference. Make sure to use `petalinux-devtool` in place of devtool in the examples.

XILINX

Appendix K

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the Vivado® IDE, select **Help → Documentation and Tutorials**.
- On Windows, select **Start → All Programs → Xilinx Design Tools → DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on DocNav, see the Documentation Navigator page on the Xilinx website.

References

These documents provide supplemental material useful with this guide:

1. PetaLinux Documentation (www.xilinx.com/petalinux)
2. Xilinx Answer Record (55776)
3. *Zynq UltraScale+ MPSoC: Software Developers Guide* (UG1137)
4. *Versal ACAP System Software Developers Guide* (UG1304)
5. *Bootgen User Guide* (UG1283)
6. Xilinx Quick Emulator User Guide (QEMU) (UG1169)
7. *Libmetal and OpenAMP for Zynq Devices User Guide* (UG1186)
8. www.wiki.xilinx.com/Linux
9. PetaLinux Yocto Tips
10. Yocto Project Technical FAQ
11. *Vitis Unified Software Platform Documentation: Embedded Software Development* (UG1400)

Please Read: Important Legal Notices

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright