See all versions
of this document

PetaLinux Tools Documentation

Reference Guide

UG1144 (v2020.1) July 24, 2020

XILINX

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
07/24/2020 Version 2020.1	
Appendix H: Partitioning and Formatting an SD Card	Added a new appendix.
06/03/2020 Version 2020.1	
Chapter 2: Setting Up Your Environment	Added the Installing a Preferred eSDK as part of the PetaLinux Tool section.
Chapter 4: Configuring and Building	Added the PetaLinux Commands with Equivalent devtool Commands section.
Chapter 6: Upgrading the Workspace	Added new sections: petalinux-upgrade Options, Upgrading Between Minor Releases (2020.1 Tool with 2020.2 Tool) , Upgrading the Installed Tool with More Platforms, and Upgrading the Installed Tool with your Customized Platform.
Chapter 7: Customizing the Project	Added new sections: Creating Partitioned Images Using Wic and Configuring SD Card ext File System Boot.
Chapter 8: Customizing the Root File System	Added the Appending Root File System Packages section.
Chapter 10: Advanced Configurations	Updated PetaLinux Menuconfig System.
Chapter 11: Yocto Features	Added the Adding Extra Users to the PetaLinux System section.
Appendix A: Migration	Added Tool/Project Directory Structure.

Table of Contents

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
`$PETALINUX/components/yocto/source/aarch64`	Zynq® UltraScale+™ MPSoC
`$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: Earlier, the eSDKs were extracted in the specified path but now they are in tar. For 2020.1 and future releases, 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`

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 (64-bit)
        - CentOS Workstation/Server 7.4, 7.5, 7.6, 7.7 (64-bit)
        - Ubuntu Linux Workstation/Server 16.04.5, 16.04.6, 18.04.1, 18.04.2, 18.04.3, 18.04.4 (64-bit)
- You need to have root access to install the required packages mentioned in the following table. 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 following table 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.

Table 2: **Packages and Linux Workstation Environments**

Tool / Library	**CentOS Workstation/ Server 7.4, 7.5, 7.6, 7.7 (64-bit)**	**Red Hat Enterprise Workstation/Server 7.4, 7.5, 7.6, 7.7, (64-bit)**	**Ubuntu Linux Workstation/Server 16.04.5, 16.04.6, 18.04.1, 18.04.2, 18.04.3, 18.04.4 (64-bit)**
ip	iproute	iproute	iproute2
gcc	gcc	gcc	gcc
g++ (gcc-c++)	gcc-c++	gcc-c++	g++
netstat	net-tools	net-tools	net-tools
ncurses devel	ncurses -devel	ncurses -devel	libncurses5 -dev
zlib devel (also, install 32-bit of this version)	zlib-devel	zlib-devel	zlib1g:i386
openssl devel	openssl -devel	openssl -devel 1.0	libssl -dev
flex	flex	flex	flex
bison	bison	bison	bison
libselinux	libselinux	libselinux	libselinux1
xterm	xterm	xterm	xterm
autoconf	autoconf	autoconf	autoconf
libtool	libtool	libtool	libtool
texinfo	texinfo	texinfo	texinfo
zlib1g-dev	-	-	zlib1g-dev
gcc-multilib	-	-	gcc-multilib
build-essential	-	-	build-essential
SDL-devel	SDL-devel	SDL-devel	-
glibc-devel	glibc-devel	glibc-devel	-
glibc	glibc.i686 glibc.x86_64	glibc.i686 glibc.x86_64	-
glib2-devel	glib2-devel	glib2-devel	-
automake	automake	automake	-
screen	screen	screen	screen
pax	pax	pax	pax
libstdc++	libstdc++.x86_64 libstdc++.i686	libstdc++.x86_64 libstdc++.i686	-
gawk	gawk	gawk	gawk
python3	python3	python3	python3
python3-pexpect			python3-pexpect
python3-pip	python3-pip	python3-pip	python3-pip
python3-GitPython	python3-GitPython	python3-GitPython	python3-git
python3-jinja2	python3-jinja2	python3-jinja2	python3-jinja2
perl	perl	perl	_

Table 2: **Packages and Linux Workstation Environments** *(cont'd)*

Tool / Library	CentOS Workstation/ Server 7.4, 7.5, 7.6, 7.7 (64-bit)	Red Hat Enterprise Workstation/Server 7.4, 7.5, 7.6, 7.7, (64-bit)	Ubuntu Linux Workstation/Server 16.04.5, 16.04.6, 18.04.1, 18.04.2, 18.04.3, 18.04.4 (64-bit)
patch	patch	patch	_
diffutils	diffutils	diffutils	_
cpp	cpp	cpp	_
perl-Data-Dumper	perl-Data-Dumper	perl-Data-Dumper	_
perl-Text-ParseWords	perl-Text-ParseWords	perl-Text-ParseWords	_
perl-Thread-Queue	perl-Thread-Queue	perl-Thread-Queue	_
xz	xz	xz	xz-utils
which	which	which	_
debianutils	_	_	debianutils
iputils-ping	_	_	iputils-ping
libegl1-mesa	_	_	libegl1-mesa
libsdl1.2-dev	_	_	libsdl1.2-dev
pylint3	_	_	pylint3
cpio	_	_	cpio

***Note*:** For the exact package versions, refer to the PetaLinux 2020.1 Release Notes and Master Answer Record: 73296.

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

IMPORTANT! *PetaLinux 2020.1 works only with hardware designs exported from Vivado® Design Suite 2020.1.*

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 are installed into the current working directory.

```
./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 3: **PetaLinux Installer Options**

Options	Description
--log <LOGFILE>	Specifies where the logfile 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. arm: sources for Zynq devices. microblaze_lite: sources for microblaze_lite microblaze_full: sources for microblaze_full

For example: To install PetaLinux tools under `/opt/pkg/petalinux/<petalinux-version>`:

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

***Note*:** Do not change the install directory permissions to CHMOD 775 as it might cause BitBake errors.

This installs the PetaLinux tool into the `/opt/pkg/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 `/opt/pkg/petalinux/<petalinux-version>` because the full path is stored in the Yocto e-SDK environment file.*

***Note*:** You cannot install the tool as a root user. While installing the software, ensure that `/opt/pkg/petalinux` is writeable 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 `/opt/pkg/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, 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 and Zynq UltraScale+ MPSoC 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 4: **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.

`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:** Give 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 2020.1
     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 5: **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.

Table 5: **PetaLinux Working Environment Troubleshooting** *(cont'd)*

Problem / Error Message	Description and Solution

`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 6: **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`

Creating a Project

PetaLinux BSP Installation

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`.

If `/tmp/<projname_timestamp>` is also on NFS, then it throws an error. You can change TMPDIR anytime through **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 install area and do not use the install area as a tmp build area.*

Troubleshooting

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

Table 7: **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

The following is a list of hardware requirements for a Zynq® UltraScale+™ MPSoC 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-7000 hardware 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.

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 (if auto config enabled for U-Boot), and enables the Linux kernel drivers (if auto config enabled for Linux) 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).

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:

    - `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.

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>
   ```

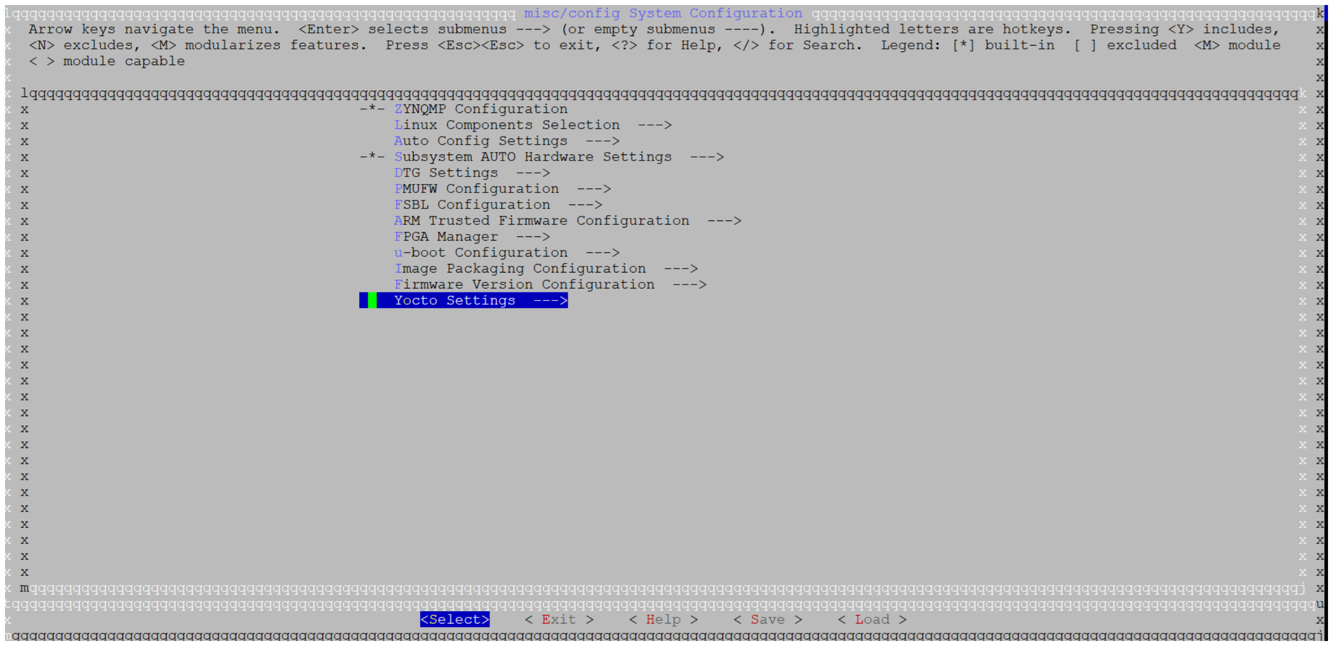
2. Import the hardware description with `petalinux-config` command by giving the path of the directory containing the .xsa file as follows:

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

   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>` 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**



```
misc/config System Configuration
 Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.  Pressing <Y> includes,
 <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module
 < > module capable

                                       -*- ZYNQMP Configuration
                                           Linux Components Selection  --->
                                           Auto Config Settings  --->
                                       -*- Subsystem AUTO Hardware Settings  --->
                                           DTG Settings  --->
                                           PMUFW Configuration  --->
                                           FSBL Configuration  --->
                                           ARM Trusted Firmware Configuration  --->
                                           FPGA Manager  --->
                                           u-boot Configuration  --->
                                           Image Packaging Configuration  --->
                                           Firmware Version Configuration  --->
                                           Yocto Settings  --->

                                   <Select>    < Exit >    < Help >    < Save >    < Load >
```

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

- The possible values are: ac701-full, ac701-lite, kc705-full, kcu105, zcu1275-revb, zcu1285-reva, zc1751-dc1, zc1751-dc2, zc702, zc706, avnet-ultra96-rev1, zcu100-revc, zcu102-rev1.0, zcu104-revc, zcu106-reva, zcu111-reva

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. To change the TMPDIR to a local storage, select* ***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  --->
[*]Advanced bootable images storage 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 and the kernel config files based on the "Auto Config Settings --->" and "Subsystem AUTO Hardware Settings --->" settings.

For example, if `ps7_ethernet_0` as the **Primary Ethernet** is selected and you enable the auto update for kernel configuration and U-Boot configuration, the tool automatically enables its kernel driver and also updates the U-Boot configuration headers for U-Boot to use the selected Ethernet controller.

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), and ATF (for Zynq UltraScale+ MPSoC), U-Boot, the Linux kernel, and a root file system image. 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 build tools
[INFO] building project
[INFO] generating Kconfig for project
[INFO] silentconfig project
[INFO] extracting yocto SDK to components/yocto
[INFO] sourcing build environment
[INFO] generating kconfig for Rootfs
[INFO] silentconfig rootfs
[INFO] generating plnxtool conf
[INFO] generating user layers
[INFO] generating workspace directory
INFO: bitbake petalinux-image-minimal
Parsing recipes: 100% |
############################################################################
##############################################################| Time:
0:00:35
Parsing of 2961 .bb files complete (0 cached, 2961 parsed). 4230 targets,
168 skipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
NOTE: Fetching uninative binary shim from file:///scratch/xilinx-
zcu102-2020.1/components/yocto/downloads/uninative/
9498d8bba047499999a7310ac2576d0796461184965351a56f6d32c888a1f216/x86_64-
nativesdk-
libc.tar.xz;sha256sum=9498d8bba047499999a7310ac2576d0796461184965351a56f6d32
c888a1f216
Initialising tasks: 100% |
############################################################################
###########################################################| Time: 0:00:03
Checking sstate mirror object availability: 100% |
############################################################################
###################################| Time: 0:00:21
Sstate summary: Wanted 1016 Found 803 Missed 213 Current 0 (79% match, 0%
complete)
NOTE: Executing Tasks
NOTE: Setscene tasks completed
NOTE: Tasks Summary: Attempted 3614 tasks of which 2619 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 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, `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: Skipping recipe linux-xlnx as it does not produce a package with the same name
  - **Description:** 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.
  - **Solution:** 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:
  - **Description:** 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.
  - **Solution:**
    - Method 1: Manually modify permissions of the license files coming from the layers

      ```
      $ chmod 666  <proj-root-dir>/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>/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.

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 build tools
INFO: File in BOOT BIN: "/scratch/petalinux/xilinx-zcu102-2020.1/images/
linux/zynqmp_fsbl.elf"
INFO: File in BOOT BIN: "/scratch/petalinux/xilinx-zcu102-2020.1/images/
linux/pmufw.elf"
INFO: File in BOOT BIN: "/scratch/petalinux/xilinx-zcu102-2020.1/images/
linux/bl31.elf"
INFO: File in BOOT BIN: "/scratch/petalinux/xilinx-zcu102-2020.1/images/
linux/system.dtb"
INFO: File in BOOT BIN: "/scratch/petalinux/xilinx-zcu102-2020.1/images/
linux/u-boot.elf"
INFO: Generating zynqmp binary package BOOT.BIN...


****** Xilinx Bootgen v2020.1
  **** Build date : May 26 2020-14:07:15
    ** Copyright 1986-2020 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 Tools Documentation: PetaLinux Command Line Reference* (UG1157).

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 Tools Documentation: PetaLinux Command Line Reference* (UG1157).

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 /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 Tools Documentation: PetaLinux Command Line Reference* (UG1157).

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 the *PetaLinux Tools Documentation: PetaLinux Command Line Reference* (UG1157).

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**

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

Note**:** If the FSBL and PMU 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

Table 8: **Local Mirror Servers** *(cont'd)*

Server	Description
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.

***Note*:** This is set to http://petalinux.xilinx.com/sswreleases/rel-v2020/aarch64/sstate-cache, by default.

PetaLinux Commands with Equivalent devtool Commands

The following table lists the PetaLinux commands with equivalent devtool commands. To execute these commands, change the `petalinux-build` flow from bitbake to devtool by selecting **petalinux-config → Yocto Settings → Build tool (devtool)**.

Table 9: **Using Devtool**

PetaLinux Command	Respective Devtool Command
petalinux-build	devtool build-image petalinux-image-minimal
petalinux-build -c <component> -x build	devtool build <component>
petalinux-build -c <component> -x clean	devtool build --clean <component>

Table 9: **Using Devtool** *(cont'd)*

PetaLinux Command	Respective Devtool Command
petalinux-build -c <component> -x finish	devtool finish <component>
petalinux-build -c <component> -x modify	devtool modify <component>
petalinux-build -c <component> -x reset	devtool reset <component> ; rm -rf <workspace>/sources/<component>
petalinux-build -c <component> -x update-recipe	devtool update-recipe <component>
petalinux-build -c <component>	devtool modify <component>; devtool build <component>
petalinux-build -s	devtool build-sdk (This was not working for current flow)
petalinux-config -c <component>	devtool modify <component>; devtool menuconfig <component>
petalinux-config -c <component> --silentconfig	devtool modify <component>; devtool configure <component>

Chapter 5

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 and Zynq-7000 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 Tools Documentation: PetaLinux Command Line Reference* (UG1157).

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.
- 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.

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 PetaLinux BSP Installation) 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.

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:

```
[    4.841731] TI DP83867 ff0e0000.ethernet-ffffffff:0c: attached PHY
driver [TI DP83867] (mii_bus:phy_addr=ff0e0000.ethernet-ffffffff:0c,
irq=POLL)
[    4.854771] macb ff0e0000.ethernet eth0: Cadence GEM rev 0x50070106
at 0xff0e0000 irq 30 (00:0a:35:00:22:01)
[    4.864857] xilinx-axipmon ffa00000.perf-monitor: Probed Xilinx APM
[    4.871379] xilinx-axipmon fd0b0000.perf-monitor: Probed Xilinx APM
[    4.877847] xilinx-axipmon fd490000.perf-monitor: Probed Xilinx APM
[    4.884328] xilinx-axipmon ffa10000.perf-monitor: Probed Xilinx APM
[    4.892194] dwc3 fe200000.dwc3: Failed to get clk 'ref': -2
[    4.897976] xilinx-psgtr fd400000.zynqmp_phy: Lane:2 type:0
protocol:3 pll_locked:yes
[    4.908242] xhci-hcd xhci-hcd.0.auto: xHCI Host Controller
[    4.913734] xhci-hcd xhci-hcd.0.auto: new USB bus registered,
assigned bus number 1
[    4.921486] xhci-hcd xhci-hcd.0.auto: hcc params 0x0238f625 hci
version 0x100 quirks 0x0000000202010810
[    4.930903] xhci-hcd xhci-hcd.0.auto: irq 54, io mem 0xfe200000
[    4.937042] usb usb1: New USB device found, idVendor=1d6b,
idProduct=0002, bcdDevice= 5.04
[    4.945313] usb usb1: New USB device strings: Mfr=3, Product=2,
SerialNumber=1
[    4.952526] usb usb1: Product: xHCI Host Controller
[    4.957396] usb usb1: Manufacturer: Linux 5.4.0-xilinx-v2020.1 xhci-
hcd
[    4.964001] usb usb1: SerialNumber: xhci-hcd.0.auto
[    4.969138] hub 1-0:1.0: USB hub found
[    4.972904] hub 1-0:1.0: 1 port detected
[    4.977017] xhci-hcd xhci-hcd.0.auto: xHCI Host Controller
[    4.982500] xhci-hcd xhci-hcd.0.auto: new USB bus registered,
assigned bus number 2
[    4.990157] xhci-hcd xhci-hcd.0.auto: Host supports USB 3.0 SuperSpeed
[    4.996882] usb usb2: New USB device found, idVendor=1d6b,
idProduct=0003, bcdDevice= 5.04
[    5.005145] usb usb2: New USB device strings: Mfr=3, Product=2,
SerialNumber=1
[    5.012362] usb usb2: Product: xHCI Host Controller
[    5.017231] usb usb2: Manufacturer: Linux 5.4.0-xilinx-v2020.1 xhci-
hcd
[    5.023839] usb usb2: SerialNumber: xhci-hcd.0.auto
[    5.028941] hub 2-0:1.0: USB hub found
[    5.032703] hub 2-0:1.0: 1 port detected
[    5.037687] pca953x 0-0020: 0-0020 supply vcc not found, using dummy
regulator
[    5.045615] pca953x 0-0021: 0-0021 supply vcc not found, using dummy
regulator
[    5.061251] i2c i2c-0: Added multiplexed i2c bus 3
[    5.072158] i2c i2c-0: Added multiplexed i2c bus 4
[    5.088893] random: fast init done
[    5.097250] ata1: SATA link down (SStatus 0 SControl 330)
[    5.123459] i2c i2c-0: Added multiplexed i2c bus 5
[    5.128360] i2c i2c-0: Added multiplexed i2c bus 6
[    5.133154] pca954x 0-0075: registered 4 multiplexed busses for I2C
mux pca9544
[    5.140480] cdns-i2c ff020000.i2c: 400 kHz mmio ff020000 irq 32
[    5.148171] at24 7-0054: 1024 byte 24c08 EEPROM, writable, 1 bytes/
write
[    5.154899] i2c i2c-1: Added multiplexed i2c bus 7
[    5.159894] i2c i2c-1: Added multiplexed i2c bus 8
[    5.167399] si570 9-005d: registered, current frequency 300000000 Hz
[    5.173780] i2c i2c-1: Added multiplexed i2c bus 9
[    5.193293] si570 10-005d: registered, current frequency 148500000 Hz
[    5.199762] i2c i2c-1: Added multiplexed i2c bus 10
[    5.204837] si5324 11-0069: si5328 probed
[    5.263052] ata2: SATA link up 3.0 Gbps (SStatus 123 SControl 330)
[    5.268669] si5324 11-0069: si5328 probe successful
[    5.274124] i2c i2c-1: Added multiplexed i2c bus 11
[    5.279126] i2c i2c-1: Added multiplexed i2c bus 12
[    5.284115] i2c i2c-1: Added multiplexed i2c bus 13
[    5.289112] i2c i2c-1: Added multiplexed i2c bus 14
[    5.293993] pca954x 1-0074: registered 8 multiplexed busses for I2C
switch pca9548
[    5.301875] i2c i2c-1: Added multiplexed i2c bus 15
[    5.306869] i2c i2c-1: Added multiplexed i2c bus 16
[    5.307741] ata2.00: ATA-7: ST3160812AS, 3.ADH, max UDMA/133
[    5.311866] i2c i2c-1: Added multiplexed i2c bus 17
[    5.317389] ata2.00: 312500000 sectors, multi 0: LBA48 NCQ (depth 32)
[    5.328880] i2c i2c-1: Added multiplexed i2c bus 18
[    5.333886] i2c i2c-1: Added multiplexed i2c bus 19
[    5.338881] i2c i2c-1: Added multiplexed i2c bus 20
[    5.343880] i2c i2c-1: Added multiplexed i2c bus 21
[    5.348874] i2c i2c-1: Added multiplexed i2c bus 22
[    5.353752] pca954x 1-0075: registered 8 multiplexed busses for I2C
switch pca9548
[    5.361338] cdns-i2c ff030000.i2c: 400 kHz mmio ff030000 irq 33
[    5.366050] ata2.00: configured for UDMA/133
[    5.370930] cdns-wdt fd4d0000.watchdog: Xilinx Watchdog Timer with
timeout 60s
[    5.371767] scsi 1:0:0:0: Direct-Access     ATA      ST3160812AS
H    PQ: 0 ANSI: 5
[    5.378957] cdns-wdt ff150000.watchdog: Xilinx Watchdog Timer with
timeout 10s
[    5.387232] sd 1:0:0:0: [sda] 312500000 512-byte logical blocks: (160
GB/149 GiB)
[    5.394415] cpufreq: cpufreq_online: CPU0: Running at unlisted freq:
1199880 KHz
[    5.401532] sd 1:0:0:0: [sda] Write Protect is off
[    5.408945] cpufreq: cpufreq_online: CPU0: Unlisted initial frequency
changed to: 1199999 KHz
[    5.422230] sd 1:0:0:0: [sda] Write cache: enabled, read cache:
enabled, doesn't support DPO or FUA
[    5.431292] usb 2-1: new SuperSpeed Gen 1 USB device number 2 using
xhci-hcd
[    5.464062]  sda: sda1
[    5.467101] sd 1:0:0:0: [sda] Attached SCSI disk
[    5.467629] mmc0: SDHCI controller on ff170000.mmc [ff170000.mmc]
using ADMA 64-bit
[    5.482267] input: gpio-keys as /devices/platform/gpio-keys/input/
input0
[    5.489346] rtc_zynqmp ffa60000.rtc: setting system clock to
2020-05-27T01:17:28 UTC (1590542248)
[    5.498217] of_cfs_init
[    5.500678] of_cfs_init: OK
[    5.503618] cfg80211: Loading compiled-in X.509 certificates for
regulatory database
```

```
[    5.551387] mmc0: new high speed SDHC card at address aaaa
[    5.557013] usb 2-1: New USB device found, idVendor=054c,
idProduct=09c2, bcdDevice= 1.00
[    5.565192] usb 2-1: New USB device strings: Mfr=1, Product=2,
SerialNumber=3
[    5.572322] usb 2-1: Product: Storage Media
[    5.576496] usb 2-1: Manufacturer: Sony
[    5.580325] usb 2-1: SerialNumber: 5C07104BE28C15CF00
[    5.585676] mmcblk0: mmc0:aaaa SL16G 14.8 GiB
[    5.592325] usb-storage 2-1:1.0: USB Mass Storage device detected
[    5.598452]  mmcblk0: p1 p2
[    5.601833] scsi host2: usb-storage 2-1:1.0
[    5.640394] cfg80211: Loaded X.509 cert 'sforshee: 00b28ddf47aef9cea7'
[    5.646922] clk: Not disabling unused clocks
[    5.651189] ALSA device list:
[    5.654141]   #0: DisplayPort monitor
[    5.658099] platform regulatory.0: Direct firmware load for
regulatory.db failed with error -2
[    5.666706] cfg80211: failed to load regulatory.db
[    5.841844] EXT4-fs (mmcblk0p2): recovery complete
[    5.850361] EXT4-fs (mmcblk0p2): mounted filesystem with ordered data
mode. Opts: (null)
[    5.855064] [drm] Cannot find any crtc or sizes
[    5.858490] VFS: Mounted root (ext4 filesystem) on device 179:2.
[    5.869011] devtmpfs: mounted
[    5.872150] Freeing unused kernel memory: 704K
[    5.876626] Run /sbin/init as init process

INIT: version 2.88 booting

Starting udev
[    6.231116] udevd[171]: starting version 3.2.8
[    6.239333] random: udevd: uninitialized urandom read (16 bytes read)
[    6.245844] random: udevd: uninitialized urandom read (16 bytes read)
[    6.252367] random: udevd: uninitialized urandom read (16 bytes read)
[    6.287438] udevd[172]: starting eudev-3.2.8
[    6.658640] EXT4-fs (sda): ext4_check_descriptors: Block bitmap for
group 880 not in group (block 1838176491)!
[    6.668737] EXT4-fs (sda): group descriptors corrupted!
[    6.684560] scsi 2:0:0:0: Direct-Access     Sony     Storage Media
PMAP PQ: 0 ANSI: 6
[    6.728123] FAT-fs (mmcblk0p1): Volume was not properly unmounted.
Some data may be corrupt. Please run fsck.
[    6.835063] cramfs: Unknown parameter 'umask'
[    6.847623] FAT-fs (sda1): Volume was not properly unmounted. Some
data may be corrupt. Please run fsck.
[    6.881627] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)
Wed May 27 01:23:03 UTC 2020

Configuring packages on first boot....

 (This may take several minutes. Please do not power off the machine.)

Running postinst /etc/rpm-postinsts/100-sysvinit-inittab...

update-rc.d: /etc/init.d/run-postinsts exists during rc.d purge
(continuing)

 Removing any system start
INIT: Entering runlevel: 5


Configuring network interfaces... [    7.345608] pps pps0: new PPS
```

```
source ptp0
[    7.349678] macb ff0e0000.ethernet: gem-ptp-timer ptp clock
registered.
udhcpc: started, v1.31.0

udhcpc: sending discover

[    7.411155] sd 2:0:0:0: [sdb] 15199296 512-byte logical blocks: (7.78
GB/7.25 GiB)
[    7.419833] sd 2:0:0:0: [sdb] Write Protect is off
[    7.425751] sd 2:0:0:0: [sdb] No Caching mode page found
[    7.431064] sd 2:0:0:0: [sdb] Assuming drive cache: write through
[    7.473694]  sdb: sdb1
[    7.485182] sd 2:0:0:0: [sdb] Attached SCSI removable disk
[    7.644305] cramfs: Unknown parameter 'umask'
[    7.653863] FAT-fs (sdb1): Volume was not properly unmounted. Some
data may be corrupt. Please run fsck.
[    8.347604] macb ff0e0000.ethernet eth0: link up (1000/Full)
[    8.353280] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
udhcpc: sending discover

udhcpc: sending select for 10.10.70.2

udhcpc: lease of 10.10.70.2 obtained, lease time 600

done.

Starting haveged: haveged: listening socket at 3

haveged: haveged starting up






Starting Dropbear SSH server: [   11.609617] urandom_read: 5 callbacks
suppressed
[   11.609623] random: dropbearkey: uninitialized urandom read (32 bytes
read)
[   11.623891] random: dropbearkey: uninitialized urandom read (32 bytes
read)
Generating 2048 bit rsa key, this may take a while...

haveged: haveged: ver: 1.9.5; arch: generic; vend: ; build: (gcc 9.2.0
CTV); collect: 128K





haveged: haveged: cpu: (VC); data: 16K (D); inst: 16K (D); idx: 11/40;
sz: 15456/64452





haveged: haveged: tot tests(BA8): A:1/1 B:1/1 continuous tests(B):  last
entropy estimate 8.00051





haveged: haveged: fills: 0, generated: 0
```

```
[   12.347988] random: crng init done
Public key portion is:

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCobZGi0OV/
ajCzeowFZ6TeEcJERNytVMWW2+F/
cHeqKnAQWrBrU4Wd7VxS4i5er5CVCUs59isTG9WidFUaVuBYJGLsC6lK/
lkHBojGuAa4Tsll4CkcpemSC2ERknqvOctRWEGJUJCKTc3lxmsYg9qklG/
dpUltXte5xedFQjt9DX4QRbgcHus1MXNGa9ARzqOz5oYPKTU6ZOAOoWdQcPHkCfnCPnMfURWj
zgeFH73MnMkJfxrnr+5c6n/H69mL/btzXUEtN4IZYQodkZrx/XMn33Ac/
d6Dg2JuvUOr4BivpEUpwS0Sw+jPi0vrFPQMYSg52Evm+Weie25t5uFA6xkh root@xilinx-
zcu102-2020_1

Fingerprint: sha1!!
e6:30:61:30:67:11:cf:5a:92:48:64:ca:ed:e5:02:f0:a1:ed:35:4f

dropbear.

Starting internet superserver: inetd.

Starting syslogd/klogd: done

Starting tcf-agent: OK


PetaLinux 2020.1 xilinx-zcu102-2020_1 /dev/ttyPS0



xilinx-zcu102-2020_1 login: rroot
oot

Password:
root


root@xilinx-zcu102-2020_1:~#
root@xilinx-zcu102-2020_1:~#
```

2. Log in to target with the default user name `root` and password `root`.

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, 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.

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 --uboot --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:

```
[    4.841731] TI DP83867 ff0e0000.ethernet-ffffffff:0c: attached PHY
driver [TI DP83867] (mii_bus:phy_addr=ff0e0000.ethernet-ffffffff:0c,
irq=POLL)
[    4.854771] macb ff0e0000.ethernet eth0: Cadence GEM rev 0x50070106 at
0xff0e0000 irq 30 (00:0a:35:00:22:01)
[    4.864857] xilinx-axipmon ffa00000.perf-monitor: Probed Xilinx APM
[    4.871379] xilinx-axipmon fd0b0000.perf-monitor: Probed Xilinx APM
[    4.877847] xilinx-axipmon fd490000.perf-monitor: Probed Xilinx APM
[    4.884328] xilinx-axipmon ffa10000.perf-monitor: Probed Xilinx APM
[    4.892194] dwc3 fe200000.dwc3: Failed to get clk 'ref': -2
[    4.897976] xilinx-psgtr fd400000.zynqmp_phy: Lane:2 type:0 protocol:3
pll_locked:yes
[    4.908242] xhci-hcd xhci-hcd.0.auto: xHCI Host Controller
[    4.913734] xhci-hcd xhci-hcd.0.auto: new USB bus registered, assigned
bus number 1
[    4.921486] xhci-hcd xhci-hcd.0.auto: hcc params 0x0238f625 hci version
0x100 quirks 0x0000000202010810
[    4.930903] xhci-hcd xhci-hcd.0.auto: irq 54, io mem 0xfe200000
[    4.937042] usb usb1: New USB device found, idVendor=1d6b,
idProduct=0002, bcdDevice= 5.04
[    4.945313] usb usb1: New USB device strings: Mfr=3, Product=2,
```

```
SerialNumber=1
[    4.952526] usb usb1: Product: xHCI Host Controller
[    4.957396] usb usb1: Manufacturer: Linux 5.4.0-xilinx-v2020.1 xhci-hcd
[    4.964001] usb usb1: SerialNumber: xhci-hcd.0.auto
[    4.969138] hub 1-0:1.0: USB hub found
[    4.972904] hub 1-0:1.0: 1 port detected
[    4.977017] xhci-hcd xhci-hcd.0.auto: xHCI Host Controller
[    4.982500] xhci-hcd xhci-hcd.0.auto: new USB bus registered, assigned
bus number 2
[    4.990157] xhci-hcd xhci-hcd.0.auto: Host supports USB 3.0 SuperSpeed
[    4.996882] usb usb2: New USB device found, idVendor=1d6b,
idProduct=0003, bcdDevice= 5.04
[    5.005145] usb usb2: New USB device strings: Mfr=3, Product=2,
SerialNumber=1
[    5.012362] usb usb2: Product: xHCI Host Controller
[    5.017231] usb usb2: Manufacturer: Linux 5.4.0-xilinx-v2020.1 xhci-hcd
[    5.023839] usb usb2: SerialNumber: xhci-hcd.0.auto
[    5.028941] hub 2-0:1.0: USB hub found
[    5.032703] hub 2-0:1.0: 1 port detected
[    5.037687] pca953x 0-0020: 0-0020 supply vcc not found, using dummy
regulator
[    5.045615] pca953x 0-0021: 0-0021 supply vcc not found, using dummy
regulator
[    5.061251] i2c i2c-0: Added multiplexed i2c bus 3
[    5.072158] i2c i2c-0: Added multiplexed i2c bus 4
[    5.088893] random: fast init done
[    5.097250] ata1: SATA link down (SStatus 0 SControl 330)
[    5.123459] i2c i2c-0: Added multiplexed i2c bus 5
[    5.128360] i2c i2c-0: Added multiplexed i2c bus 6
[    5.133154] pca954x 0-0075: registered 4 multiplexed busses for I2C mux
pca9544
[    5.140480] cdns-i2c ff020000.i2c: 400 kHz mmio ff020000 irq 32
[    5.148171] at24 7-0054: 1024 byte 24c08 EEPROM, writable, 1 bytes/write
[    5.154899] i2c i2c-1: Added multiplexed i2c bus 7
[    5.159894] i2c i2c-1: Added multiplexed i2c bus 8
[    5.167399] si570 9-005d: registered, current frequency 300000000 Hz
[    5.173780] i2c i2c-1: Added multiplexed i2c bus 9
[    5.193293] si570 10-005d: registered, current frequency 148500000 Hz
[    5.199762] i2c i2c-1: Added multiplexed i2c bus 10
[    5.204837] si5324 11-0069: si5328 probed
[    5.263052] ata2: SATA link up 3.0 Gbps (SStatus 123 SControl 330)
[    5.268669] si5324 11-0069: si5328 probe successful
[    5.274124] i2c i2c-1: Added multiplexed i2c bus 11
[    5.279126] i2c i2c-1: Added multiplexed i2c bus 12
[    5.284115] i2c i2c-1: Added multiplexed i2c bus 13
[    5.289112] i2c i2c-1: Added multiplexed i2c bus 14
[    5.293993] pca954x 1-0074: registered 8 multiplexed busses for I2C
switch pca9548
[    5.301875] i2c i2c-1: Added multiplexed i2c bus 15
[    5.306869] i2c i2c-1: Added multiplexed i2c bus 16
[    5.307741] ata2.00: ATA-7: ST3160812AS, 3.ADH, max UDMA/133
[    5.311866] i2c i2c-1: Added multiplexed i2c bus 17
[    5.317389] ata2.00: 312500000 sectors, multi 0: LBA48 NCQ (depth 32)
[    5.328880] i2c i2c-1: Added multiplexed i2c bus 18
[    5.333886] i2c i2c-1: Added multiplexed i2c bus 19
[    5.338881] i2c i2c-1: Added multiplexed i2c bus 20
[    5.343880] i2c i2c-1: Added multiplexed i2c bus 21
[    5.348874] i2c i2c-1: Added multiplexed i2c bus 22
[    5.353752] pca954x 1-0075: registered 8 multiplexed busses for I2C
switch pca9548
[    5.361338] cdns-i2c ff030000.i2c: 400 kHz mmio ff030000 irq 33
[    5.366050] ata2.00: configured for UDMA/133
[    5.370930] cdns-wdt fd4d0000.watchdog: Xilinx Watchdog Timer with
```

```
timeout 60s
[    5.371767] scsi 1:0:0:0: Direct-Access     ATA      ST3160812AS
H    PQ: 0 ANSI: 5
[    5.378957] cdns-wdt ff150000.watchdog: Xilinx Watchdog Timer with
timeout 10s
[    5.387232] sd 1:0:0:0: [sda] 312500000 512-byte logical blocks: (160 GB/
149 GiB)
[    5.394415] cpufreq: cpufreq_online: CPU0: Running at unlisted freq:
1199880 KHz
[    5.401532] sd 1:0:0:0: [sda] Write Protect is off
[    5.408945] cpufreq: cpufreq_online: CPU0: Unlisted initial frequency
changed to: 1199999 KHz
[    5.422230] sd 1:0:0:0: [sda] Write cache: enabled, read cache: enabled,
doesn't support DPO or FUA
[    5.431292] usb 2-1: new SuperSpeed Gen 1 USB device number 2 using xhci-
hcd
[    5.464062]  sda: sda1
[    5.467101] sd 1:0:0:0: [sda] Attached SCSI disk
[    5.467629] mmc0: SDHCI controller on ff170000.mmc [ff170000.mmc] using
ADMA 64-bit
[    5.482267] input: gpio-keys as /devices/platform/gpio-keys/input/input0
[    5.489346] rtc_zynqmp ffa60000.rtc: setting system clock to
2020-05-27T01:17:28 UTC (1590542248)
[    5.498217] of_cfs_init
[    5.500678] of_cfs_init: OK
[    5.503618] cfg80211: Loading compiled-in X.509 certificates for
regulatory database
[    5.551387] mmc0: new high speed SDHC card at address aaaa
[    5.557013] usb 2-1: New USB device found, idVendor=054c,
idProduct=09c2, bcdDevice= 1.00
[    5.565192] usb 2-1: New USB device strings: Mfr=1, Product=2,
SerialNumber=3
[    5.572322] usb 2-1: Product: Storage Media
[    5.576496] usb 2-1: Manufacturer: Sony
[    5.580325] usb 2-1: SerialNumber: 5C07104BE28C15CF00
[    5.585676] mmcblk0: mmc0:aaaa SL16G 14.8 GiB
[    5.592325] usb-storage 2-1:1.0: USB Mass Storage device detected
[    5.598452]  mmcblk0: p1 p2
[    5.601833] scsi host2: usb-storage 2-1:1.0
[    5.640394] cfg80211: Loaded X.509 cert 'sforshee: 00b28ddf47aef9cea7'
[    5.646922] clk: Not disabling unused clocks
[    5.651189] ALSA device list:
[    5.654141]   #0: DisplayPort monitor
[    5.658099] platform regulatory.0: Direct firmware load for
regulatory.db failed with error -2
[    5.666706] cfg80211: failed to load regulatory.db
[    5.841844] EXT4-fs (mmcblk0p2): recovery complete
[    5.850361] EXT4-fs (mmcblk0p2): mounted filesystem with ordered data
mode. Opts: (null)
[    5.855064] [drm] Cannot find any crtc or sizes
[    5.858490] VFS: Mounted root (ext4 filesystem) on device 179:2.
[    5.869011] devtmpfs: mounted
[    5.872150] Freeing unused kernel memory: 704K
[    5.876626] Run /sbin/init as init process

INIT: version 2.88 booting

Starting udev
[    6.231116] udevd[171]: starting version 3.2.8
[    6.239333] random: udevd: uninitialized urandom read (16 bytes read)
[    6.245844] random: udevd: uninitialized urandom read (16 bytes read)
[    6.252367] random: udevd: uninitialized urandom read (16 bytes read)
[    6.287438] udevd[172]: starting eudev-3.2.8
[    6.658640] EXT4-fs (sda): ext4_check_descriptors: Block bitmap for
group 880 not in group (block 1838176491)!
[    6.668737] EXT4-fs (sda): group descriptors corrupted!
[    6.684560] scsi 2:0:0:0: Direct-Access     Sony     Storage Media
PMAP PQ: 0 ANSI: 6
[    6.728123] FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some
data may be corrupt. Please run fsck.
[    6.835063] cramfs: Unknown parameter 'umask'
[    6.847623] FAT-fs (sda1): Volume was not properly unmounted. Some data
may be corrupt. Please run fsck.
[    6.881627] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)
Wed May 27 01:23:03 UTC 2020

Configuring packages on first boot....

 (This may take several minutes. Please do not power off the machine.)

Running postinst /etc/rpm-postinsts/100-sysvinit-inittab...

update-rc.d: /etc/init.d/run-postinsts exists during rc.d purge (continuing)

 Removing any system start
INIT: Entering runlevel: 5


Configuring network interfaces... [    7.345608] pps pps0: new PPS source
ptp0
[    7.349678] macb ff0e0000.ethernet: gem-ptp-timer ptp clock registered.
udhcpc: started, v1.31.0

udhcpc: sending discover

[    7.411155] sd 2:0:0:0: [sdb] 15199296 512-byte logical blocks: (7.78 GB/
7.25 GiB)
[    7.419833] sd 2:0:0:0: [sdb] Write Protect is off
[    7.425751] sd 2:0:0:0: [sdb] No Caching mode page found
[    7.431064] sd 2:0:0:0: [sdb] Assuming drive cache: write through
[    7.473694]  sdb: sdb1
[    7.485182] sd 2:0:0:0: [sdb] Attached SCSI removable disk
[    7.644305] cramfs: Unknown parameter 'umask'
[    7.653863] FAT-fs (sdb1): Volume was not properly unmounted. Some data
may be corrupt. Please run fsck.
[    8.347604] macb ff0e0000.ethernet eth0: link up (1000/Full)
[    8.353280] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
udhcpc: sending discover

udhcpc: sending select for 10.10.70.2

udhcpc: lease of 10.10.70.2 obtained, lease time 600

done.

Starting haveged: haveged: listening socket at 3

haveged: haveged starting up





Starting Dropbear SSH server: [   11.609617] urandom_read: 5 callbacks
suppressed
```

```
[   11.609623] random: dropbearkey: uninitialized urandom read (32 bytes
read)
[   11.623891] random: dropbearkey: uninitialized urandom read (32 bytes
read)
Generating 2048 bit rsa key, this may take a while...

haveged: haveged: ver: 1.9.5; arch: generic; vend: ; build: (gcc 9.2.0
CTV); collect: 128K





haveged: haveged: cpu: (VC); data: 16K (D); inst: 16K (D); idx: 11/40; sz:
15456/64452





haveged: haveged: tot tests(BA8): A:1/1 B:1/1 continuous tests(B):  last
entropy estimate 8.00051





haveged: haveged: fills: 0, generated: 0





[   12.347988] random: crng init done
Public key portion is:

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCobZGi0OV/ajCzeowFZ6TeEcJERNytVMWW2+F/
cHeqKnAQWrBrU4Wd7VxS4i5er5CVCUs59isTG9WidFUaVuBYJGLsC61K/
lkHBojGuAa4Tsll4CkcpemSC2ERknqvOctRWEGJUJCKTc3lxmsYg9qklG/
dpUltXte5xedFQjt9DX4QRbgcHuslMXNGa9ARzqOz5oYPKTU6ZOAOoWdQcPHkCfnCPnMfURWjzge
FH73MnMkJfxrnr+5c6n/H69mL/btzXUEtN4IZYQodkZrx/XMn33Ac/
d6Dg2JuvUOr4BivpEUpwS0Sw+jPi0vrFPQMYSg52Evm+Weie25t5uFA6xkh root@xilinx-
zcu102-2020_1

Fingerprint: sha1!!
e6:30:61:30:67:11:cf:5a:92:48:64:ca:ed:e5:02:f0:a1:ed:35:4f

dropbear.

Starting internet superserver: inetd.

Starting syslogd/klogd: done

Starting tcf-agent: OK


PetaLinux 2020.1 xilinx-zcu102-2020_1 /dev/ttyPS0



xilinx-zcu102-2020_1 login: rroot
oot

Password:
```

```
root

root@xilinx-zcu102-2020_1:~#
root@xilinx-zcu102-2020_1:~#
```

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

Login to the virtual system when you see the login prompt on the emulator console with the login `root` and password `root`. 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**.

- Boot a specific Linux image:

The `petalinux-boot` tool can also boot a specific Linux image using the image option (`-i` or `--image`):

```
$ petalinux-boot --qemu --image <path-to-Linux-image-file>
```

For example:

```
$ petalinux-boot --qemu --image ./images/linux/zImage
```

- Direct Boot a Linux Image with Specific DTB:

Device Trees (DTB files) are used to describe the hardware architecture and address map to the Linux kernel. The PetaLinux system emulator also uses DTB files to dynamically configure the emulation environment to match your hardware platform.

If no DTB file option is provided, `petalinux-boot` extracts the DTB file from the given `image.elf` for MicroBlaze processors and from `<plnx-proj-root>/images/linux/system.dtb` for Zynq-7000 devices and Zynq UltraScale+ MPSoC. Alternatively, you can use the `--dtb` option as follows:

```
$ petalinux-boot --qemu --image ./images/linux/zImage --dtb ./images/linux/
system.dtb
```

***Note*:** QEMU version has been upgraded to 4.2. The old options are deprecated in the new version but remain functionally operational. Since PetaLinux tools still use the old options, warning messages are displayed. 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 for Zynq® UltraScale+™ MPSoC and Zynq-7000 devices only because they allow you to boot from SD cards.*

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 PetaLinux BSP Installation.
- 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

1. Mount the SD card on your host machine.
2. 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`
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. You should see a boot messages similar to the following message on the serial console:

```
[    7.741083] cfg80211: Loading compiled-in X.509 certificates for
regulatory database
[    7.882290] cfg80211: Loaded X.509 cert 'sforshee: 00b28ddf47aef9cea7'
[    7.888818] clk: Not disabling unused clocks
[    7.893083] ALSA device list:
[    7.896040]   #0: DisplayPort monitor
[    7.899994] platform regulatory.0: Direct firmware load for
regulatory.db failed with error -2
[    7.908610] cfg80211: failed to load regulatory.db
[    7.913597] Freeing unused kernel memory: 704K
[    7.931199] Run /init as init process

INIT: version 2.88 booting

Starting udev
[    8.026549] mmc0: new high speed SDHC card at address aaaa
[    8.032550] mmcblk0: mmc0:aaaa SP32G 29.7 GiB
[    8.038229] udevd[167]: starting version 3.2.8
[    8.039893]  mmcblk0: p1
[    8.042979] random: udevd: uninitialized urandom read (16 bytes read)
[    8.051695] random: udevd: uninitialized urandom read (16 bytes read)
[    8.058197] random: udevd: uninitialized urandom read (16 bytes read)
[    8.067225] usb 2-1: new SuperSpeed Gen 1 USB device number 2 using
xhci-hcd
[    8.069110] udevd[169]: starting eudev-3.2.8
[    8.075194] [drm] Cannot find any crtc or sizes
[    8.091909] usb 2-1: New USB device found, idVendor=0781,
idProduct=5580, bcdDevice= 0.10
[    8.100109] usb 2-1: New USB device strings: Mfr=1, Product=2,
SerialNumber=3
[    8.107248] usb 2-1: Product: Extreme
[    8.110900] usb 2-1: Manufacturer: SanDisk
[    8.114988] usb 2-1: SerialNumber: AA010104131512483744
[    8.121414] usb-storage 2-1:1.0: USB Mass Storage device detected
[    8.128190] scsi host2: usb-storage 2-1:1.0
[    8.152664] zynqmp_r5_remoteproc ff9a0000.zynqmp-rpu: RPU core_conf:
split
[    8.162198] remoteproc remoteproc0: r5@0 is available
[    8.455632] mali: loading out-of-tree module taints kernel.
[    8.960540] cramfs: Unknown parameter 'umask'
[    8.971664] FAT-fs (sda1): Volume was not properly unmounted. Some
data may be corrupt. Please run fsck.
[    8.981200] vfat filesystem being mounted at /run/media/sda1 supports
timestamps until 2107 (0x10391447e)
[    9.139803] scsi 2:0:0:0: Direct-Access     SanDisk  Extreme
0001 PQ: 0 ANSI: 6
[    9.149297] sd 2:0:0:0: [sdb] 62533296 512-byte logical blocks: (32.0
GB/29.8 GiB)
[    9.157860] sd 2:0:0:0: [sdb] Write Protect is off
[    9.163586] sd 2:0:0:0: [sdb] Write cache: disabled, read cache:
enabled, doesn't support DPO or FUA
[    9.198380]  sdb: sdb1
[    9.204317] sd 2:0:0:0: [sdb] Attached SCSI removable disk
Configuring packages on first boot....

 (This may take several minutes. Please do not power off the machine.)

Running postinst /etc/rpm-postinsts/100-sysvinit-inittab...

Running postinst /etc/rpm-postinsts/101-libmali-xlnx...
```

```
[    9.333900] cramfs: Unknown parameter 'umask'
[    9.335514] update-alternatives: Linking /usr/lib/libMali.so.9.0
to /usr/lib/x11/libMali.so.9.0
[    9.340086] FAT-fs (sdb1): Volume was not properly unmounted. Some
data may be corrupt. Please run fsck.
[    9.356558] vfat filesystem being mounted at /run/media/sdb1 supports
timestamps until 2107 (0x10391447e)
[    9.385844] update-alternatives: Linking /usr/lib/libMali.so.9.0
to /usr/lib/x11/libMali.so.9.0
[    9.414144] Warn: update-alternatives: libmali-xlnx has multiple
providers with the same priority, please check /usr/lib/opkg/
alternatives/libmali-xlnx for details
[    9.440237] update-alternatives: Linking /usr/lib/libMali.so.9.0
to /usr/lib/x11/libMali.so.9.0
[    9.480366] update-alternatives: Linking /usr/lib/libMali.so.9.0
to /usr/lib/x11/libMali.so.9.0
update-rc.d: /etc/init.d/run-postinsts exists during rc.d purge
(continuing)

 Removing any system st
INIT: Entering runlevel: 5


Configuring network interfaces... [    9.559410] pps pps0: new PPS
source ptp0
[    9.563425] macb ff0e0000.ethernet: gem-ptp-timer ptp clock
registered.
udhcpc: started, v1.31.0

udhcpc: sending discover

[   10.579726] macb ff0e0000.ethernet eth0: link up (1000/Full)
[   10.585412] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
udhcpc: sending discover

udhcpc: sending select for 10.10.70.1

udhcpc: lease of 10.10.70.1 obtained, lease time 600

done.

Starting system message bus: dbus.

Starting haveged: haveged: listening socket at 3

haveged: haveged starting up






Starting Dropbear SSH server: Generating 2048 bit rsa key, this may take
a while...

haveged: haveged: ver: 1.9.5; arch: generic; vend: ; build: (gcc 9.2.0
CTV); collect: 128K




haveged: haveged: cpu: (VC); data: 16K (D); inst: 16K (D); idx: 11/40;
sz: 15456/64452
```

```
haveged: haveged: tot tests(BA8): A:1/1 B:1/1 continuous tests(B):  last
entropy estimate 7.99902



haveged: haveged: fills: 0, generated: 0



[   14.561816] random: crng init done
[   14.565218] random: 7 urandom warning(s) missed due to ratelimiting
Public key portion is:

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCq0ua7bEdUb0PHmF3f8HVbpcLq/
ahhMCFrQZsgK2UpDmkoMAMMsYRwna2Qog1W1FmaBmIhUT8OI3nVApVc21I7q35Yd2PcAmyeVX
ZqCnYUriaae7hGrwpRgMNq48ixiWqcf7OXCnLSQ5GI8eJjeV5JMHwv1puU0mnSkFFbx46UnL1
N7b+qMxz9FTTLQBIWS/PUZTTewZs+hKExv8OGZE1T46YdmrY89zc
+6of9c5NMdhGtD5fy7RRKY28fnUQ3wkN7ensLcJhVkWfrUyDVOoh2Th/
gh2D6RBAkMwxdnLUmdMrP8tbyGOxY0stM+9pO2t7qn6cvT9wHx7ekUaS/zqaR
root@xilinx-zcu102-2020_1

Fingerprint: sha1!!
c7:b9:cd:c4:a8:0a:b9:a8:ec:7b:a6:b1:55:a8:29:79:68:15:00:05

dropbear.

Starting internet superserver: inetd.

Starting syslogd/klogd: done

Starting tcf-agent: OK


PetaLinux 2020.1 xilinx-zcu102-2020_1 /dev/ttyPS0



xilinx-zcu102-2020_1 login: rroot
Password: oot


root
root@xilinx-zcu102-2020_1:~#
```

TIP: *If you wish to stop auto-boot, hit any key when you see the messages similar to the following on the console:* `Hit any key to stop autoboot:`

10. Type user name `root` and password `root` on the serial console to log into the PetaLinux system.

Troubleshooting

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

Table 10: **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.

JTAG boot communicates with XSDB which in turn communicates with 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 PetaLinux BSP Installation) 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:

   ```
   NOTICE:  ATF running on XCZU9EG/silicon v4/RTL5.1 at 0xfffea000
   NOTICE:  BL31: v2.2(release):v1.1-5588-g5918e656e
   NOTICE:  BL31: Built : 22:53:26, May 26 2020


   U-Boot 2020.01 (May 26 2020 - 22:58:43 +0000)

   Model: ZynqMP ZCU102 Rev1.0
   Board: Xilinx ZynqMP
   DRAM:  4 GiB
   PMUFW:    v1.1
   EL Level:    EL2
   Chip ID:    zu9eg
   ```

```
NAND:   0 MiB
MMC:    mmc@ff170000: 0
In:     serial@ff000000
Out:    serial@ff000000
Err:    serial@ff000000
Bootmode: JTAG_MODE
Reset reason:    SRST
Net:
ZYNQ GEM: ff0e0000, mdio bus ff0e0000, phyaddr 12, interface rgmii-id

Warning: ethernet@ff0e0000 using MAC address from DT
eth0: ethernet@ff0e0000
Hit any key to stop autoboot:  2
ZynqMP>
```

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. Type user name: `root` and password: `root` on the serial console to log into the PetaLinux system.
10. 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 rootfs.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.

- 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
  ```

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 11: **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 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) 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:

```
Xilinx Zynq MP First Stage Boot Loader
Release 2020.1   Apr 27 2020  -  08:55:54
NOTICE:  ATF running on XCZU9EG/silicon v4/RTL5.1 at 0xfffea000
NOTICE:  BL31: v2.2(release):xilinx-v2019.2-1739-gbf72e4d49
NOTICE:  BL31: Built : 10:12:17, Apr 27 2020


U-Boot 2020.01 (Apr 27 2020 - 10:12:56 +0000)

Model: ZynqMP ZCU102 Rev1.0
Board: Xilinx ZynqMP
DRAM:  4 GiB
PMUFW:    v1.1
EL Level:    EL2
Chip ID:    zu9eg
NAND:  0 MiB
MMC:   mmc@ff170000: 0
In:    serial@ff000000
Out:   serial@ff000000
Err:   serial@ff000000
Bootmode: JTAG_MODE
Reset reason:    DEBUG
Net:
ZYNQ GEM: ff0e0000, mdio bus ff0e0000, phyaddr 12, interface rgmii-id

Warning: ethernet@ff0e0000 using MAC address from DT
eth0: ethernet@ff0e0000
Hit any key to stop autoboot:  2

 0
ZynqMP>
ZynqMP>
```

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> set serverip <HOST IP ADDRESS>
```

12. Get the pxe boot file using the following command:

```
ZynqMP> pxe get
```

13. Boot the kernel using the following command:

```
ZynqMP> pxe boot
```

Troubleshooting

Table 12: **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> set 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/plnx-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.

petalinux-upgrade Options

Table 13: **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 arm: sources for Zynq devices microblaze_lite: sources for microblaze_lite microblaze_full: sources for microblaze_full	None

Upgrading Between Minor Releases (2020.1 Tool with 2020.2 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-v2020/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-v2020/
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, microblaze_lite, microblaze_full).

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

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

- **To upgrade eSDKs for Zynq and Zynq UltraScale+ MPSoC 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 the Installing the PetaLinux Tool. To install the aarch64 platform, follow these steps.

1. Go to the installed tool.
2. Source `settings.sh` file.
3. Run: `petalinux-upgrade -u http://petalinux.xilinx.com/sswreleases/rel-v2020/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 and Zynq UltraScale+ MPSoC 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.

Chapter 7

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**/**INITRD**/**JFFS2**/**NFS**/**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.

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.

Steps for Boot Images Storage 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 **Subsystem AUTO Hardware Settings → Advanced Bootable Images Storage Settings**.
4. In the Advanced Bootable Images Storage Settings submenu, you have the following options:
    - Boot image settings (BOOT.BIN which includes FSBL, PMU, and ATF for Zynq® UltraScale+™ MPSoC)

      Select boot device as per requirement.

        - To set flash as the boot device, select **primary flash**.
        - To make SD card as the boot device, select **primary sd**.
    - U-Boot env partition settings
    - Kernel image settings (image.ub - Linux kernel, DTB, and RootFS)
    - Image storage media

      Select storage device as per the requirement.

        - To set flash as the boot device, select **primary flash**.
        - To make SD card as the boot device, select **primary sd**.
    - Image name

      The default kernel image is fitimage (`image.ub`).

      You can change the kernel image (Image) using this menuconfig option.
    - jffs2 RootFS image settings
    - DTB settings

Troubleshooting

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

Table 14: **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.

The PetaLinux tools uses the start address for parallel flash or start offset for SPI flash and the size of the above partitions to generate the following U-Boot commands:

- `update_boot` if the boot image, which is a U-Boot image for MicroBlaze™ processors and a `BOOT.BIN` image for Zynq®-7000 devices, is selected to be stored in the primary flash.
- `update_kernel` and `load_kernel` if the kernel image, which is the FIT image `image.ub`, is selected to be stored in the flash.

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 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

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.

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

   If kernel or the root file system size increases and is greater than 128 MB, you need to do the following:

   a. Mention the Bootm length in `<plnx-proj-root>/project-spec/meta-user/recipes-bsp/u-boot/files/platform-top.h`.

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

   b. Undef `CONFIG_SYS_BOOTMAPSZ` in `<plnx-proj-root>/project-spec/meta-user/recipes-bsp/u-boot/files/platform-top.h`.

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. 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 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> set 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. Build the system image. For more information, see Build System Image.
3. 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) + `rootfs.cpio` (for Zynq® UltraScale+™ MPSoC)
- zImage → zImage (kernel) + `rootfs.cpio` (for Zynq-7000 devices)
- linux.bin.ub → simpleImage.mb (kernel) + `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. 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. 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 JTAG, see Boot a PetaLinux Image on Hardware with an SD Card.
5. Make sure you have configured TFTP server in host.
6. Set the server IP address to the host IP address using the following command at U-Boot prompt.

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

   a. Detect Flash Memory.

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

   b. Erase Flash Memory.

      ```
      ZynqMP> sf erase 0 0x5000000
      ```

c. Read images onto Memory and write into Flash.

- Read `BOOT.BIN`.

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

- Write `BOOT.BIN`.

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

Example: `sf write 0x80000 0 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 0x580000 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 0x1980000 0x7d4000`

Note: Check loading addresses for kernel and root file system inside `system.dts`.

- 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 0x7F80000 0x80000

Note: Check the addresses in `<plnx-project-root>/project-spec/meta-user/recipes-bsp/u-boot/u-boot-zynq-scr/boot.cmd.default.initrd`. If they do not match, the process may fail in the U-Boot prompt.

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

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 4 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 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.

1. The SD card is formatted with two partitions using a partition editor such as gparted.
2. 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. Label this partition as BOOT.
3. 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. Label this partition as RootFS.
4. Copy the files as follows:

   FAT partition:`BOOT.BIN`, `boot.scr`, and Image

   EXT partition: `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. Copy `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.

Troubleshooting

Table 15: **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.

Creating Partitioned Images Using 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 1G and EXT4 with 3 GB. You can find the default kickstart file in `<plnx-proj-root>/build/wic/rootfs.wks` after the `petalinux-package --wic` command is executed.

```
$ petalinux-package --wic
```

Package 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
```

Package 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/
```

Package wic Image with Specified Images Path

The following command packs all bootfiles from the `custom-imagespath/` directory.

```
$ petalinux-package --wic --images-dir custom-imagespath/
```

Package Custom Bootfiles into the /boot Directory

- To copy `boot.bin userfile1 userfile2` files from the `<plnx-proj-dir>/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.

Package 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
```

Copy the Image SD Card

The following command copies the image SD card to the EXT4 partition. Alternatively, you can use the etcher tool or Win32Diskimager from Windows to flash this image.

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

Chapter 8

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 '_', see Recipe Name Having ' _ '.

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.

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 '_', see Recipe Name Having ' _ '.

   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.

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 16: **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>/build/conf/local.conf.<plnx-proj>/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.

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 '_', see Recipe Name Having ' _ '.

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 17: **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 the below steps for PetaLinux Auto Login:

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**.
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 into the PetaLinux project. The upstream layers for ZEUS version can be found http://layers.openembedded.org/layerindex/branch/zeus/layers/.

The following steps demonstrate adding 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: 2020.1 PetaLinux is on ZEUS base line. The layers/recipes should be chosen from the ZEUS branch only. Some of the layers/recipes might not be compatible with our architectures. 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/locallayer.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
```

Chapter 9

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 18: **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 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
```

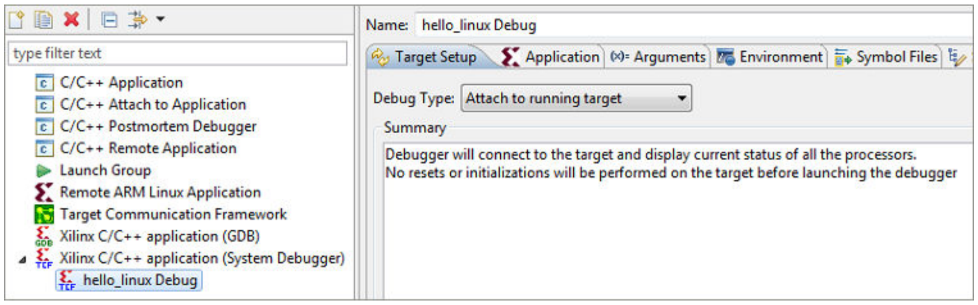
8. Select **console/network→openssh**. Ensure that "openssh-sftp-server" is enabled.
9. Exit the menu.
10. Rebuild the target system image including myapp. For more information, see Build System Image.

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.



type filter text
C/C++ Application
C/C++ Attach to Application
C/C++ Postmortem Debugger
C/C++ Remote Application
Launch Group
Remote ARM Linux Application
Target Communication Framework
Xilinx C/C++ application (GDB)
Xilinx C/C++ application (System Debugger)
hello_linux Debug
Target Setup
Application
Arguments
Environment
Symbol Files

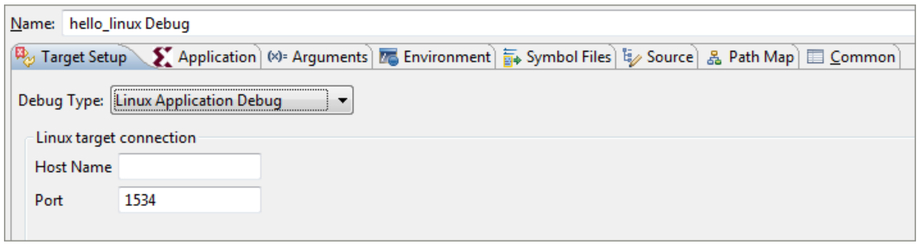
Name: hello_linux Debug

Debug Type: Attach to running target

Summary

Debugger will connect to the target and display current status of all the processors.
No resets or initializations will be performed on the target before launching the debugger

6. In the Debug Configuration window:
   a. Click the **Target Setup** tab.
   b. From the Debug Type drop-down list, select **Linux Application Debug**.



Name: hello_linux Debug

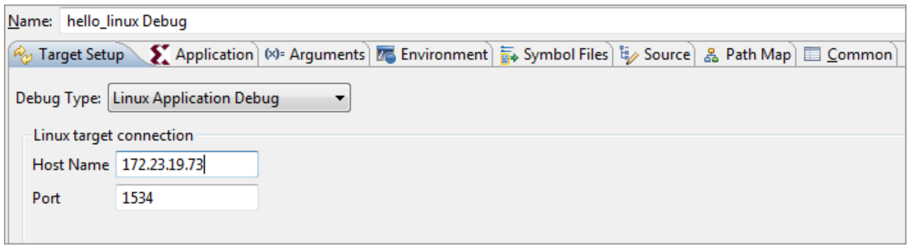
Debug Type: Linux Application Debug

Linux target connection

Host Name

Port 1534

   c. Provide the Linux host name or IP address in the Host Name field.



Name: hello_linux Debug

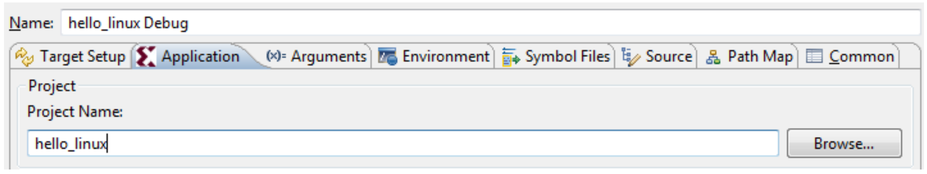
Debug Type: Linux Application Debug

Linux target connection

Host Name 172.23.19.73

Port 1534

   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.



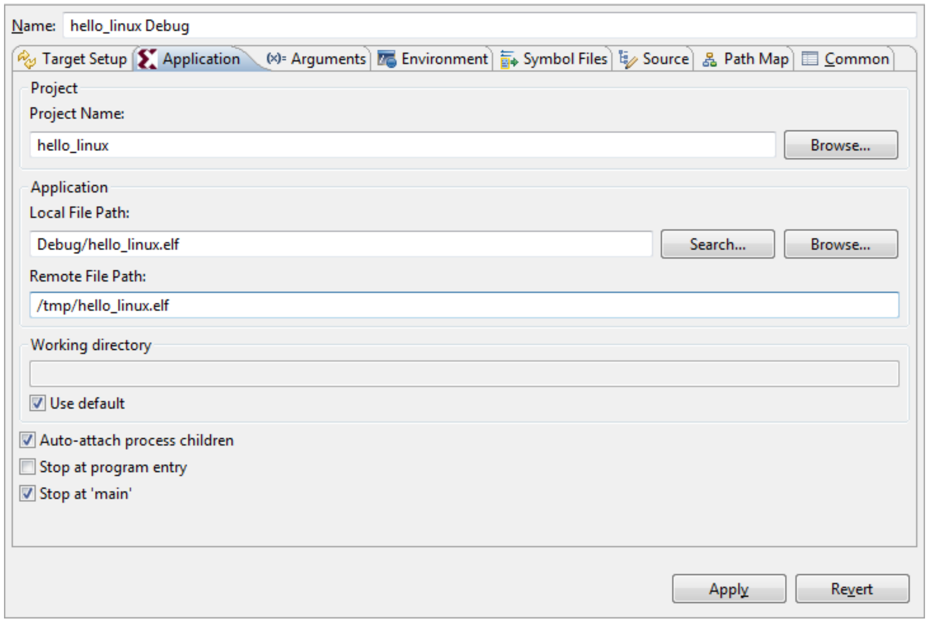
Name: hello_linux Debug

Project

Project Name:

hello_linux

   f. In the Remote File Path field, specify the path where you want to download the application in Linux.



Project

Project Name: hello_linux

Application

Local File Path: Debug/hello_linux.elf

Remote File Path: /tmp/hello_linux.elf

Working directory

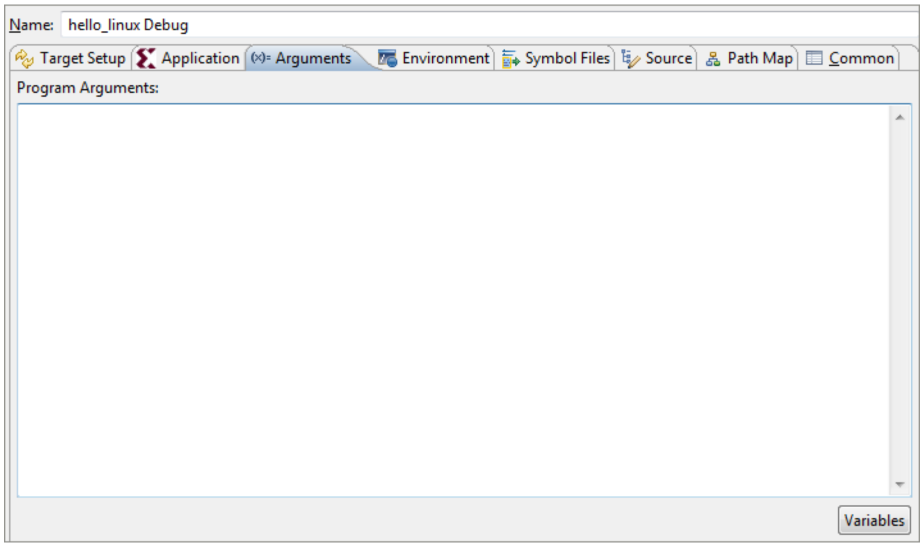
☑ Use default

☑ Auto-attach process children

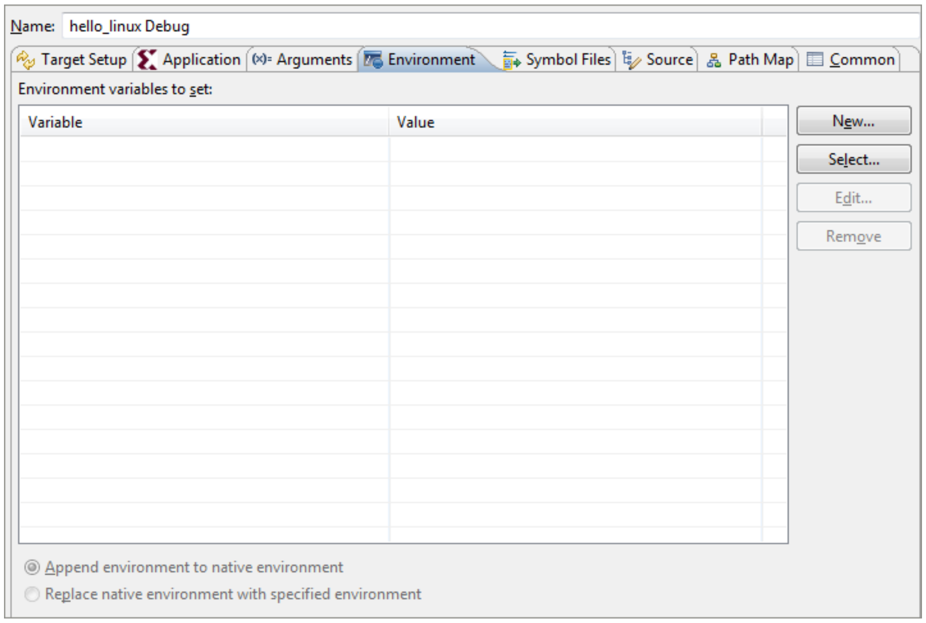
☐ Stop at program entry

☑ Stop at 'main'

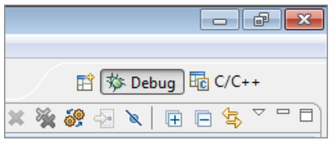
g. If your application is expecting some arguments, specify them in the Arguments tab.

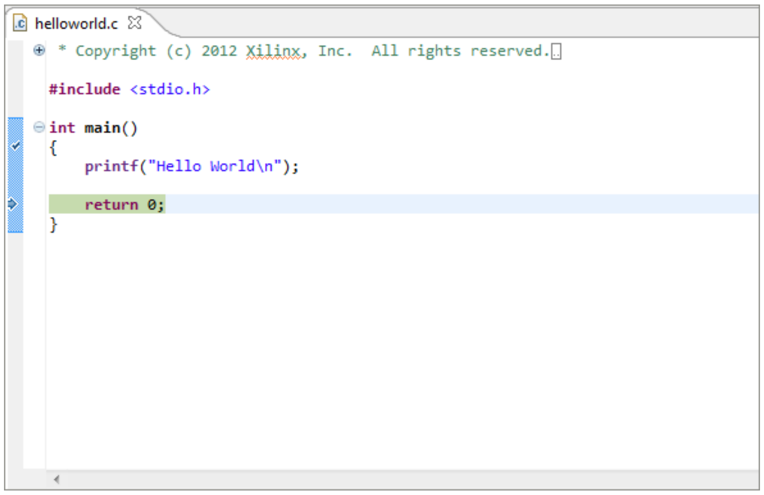


h. If your application is expecting to set some environment variables, specify them in the Environment tab.



i. Click the **Debug** button. A separate console automatically opens for process standard I/O operations.





```
* Copyright (c) 2012 Xilinx, Inc.  All rights reserved.

#include <stdio.h>

int main()
{
    printf("Hello World\n");

    return 0;
}
```

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 Applications with GDB

PetaLinux supports debugging Zynq® UltraScale+™ MPSoC 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. Run `petalinux-config -c rootfs` on the command console:

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

4. Scroll down the user packages Configuration menu to Debugging:

```
Filesystem Packages   --->
PetaLinux Package Groups --->
apps   --->
user packages   --->
PetaLinux RootFS Settings  --->
```

5. Select **user packages**.

[X] myapp-dbg

[ ] myapp-dev

6. Select **myapp-dbg**. Exit the myapp sub-menu.
7. Exit the user packages sub-menu, and select **Filesystem Packages → misc → gdb**.
8. Select **gdb**, and ensure that the GDB server is enabled:

   [ ] gdb

   [ ] gdb-dev

   [X] gdbserver

   [ ] gdb-dbg
9. Exit the menu and select **<Yes>** to save the configuration.
10. Rebuild the target system image. Add the below 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. Run `gdbserver` with the user application on the target system console (set to listening on port 1534):

   ```
   root@plnx_aarch64:~# gdbserver host: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
3. On the workstation, navigate to the compiled user application's directory:

   ```
   $ cd <<TMPDIR>/work/aarch64-xilinx-linux/myapp1/1.0-r0/image/usr/bin/
   myapp
   ```
4. Run GDB client.

   ```
   $ petalinux-util --gdb myapp
   ```

   The GDB console starts:

   ```
   ...
   GNU gdb (crosstool-NG 1.18.0) 7.6.0.20130721-cvs
   ...
   (gdb)
   ```

5. 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 localhost 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
```

6. 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.
```

7. 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");
```

8. 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
```

9. 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.

10. 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 19: **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

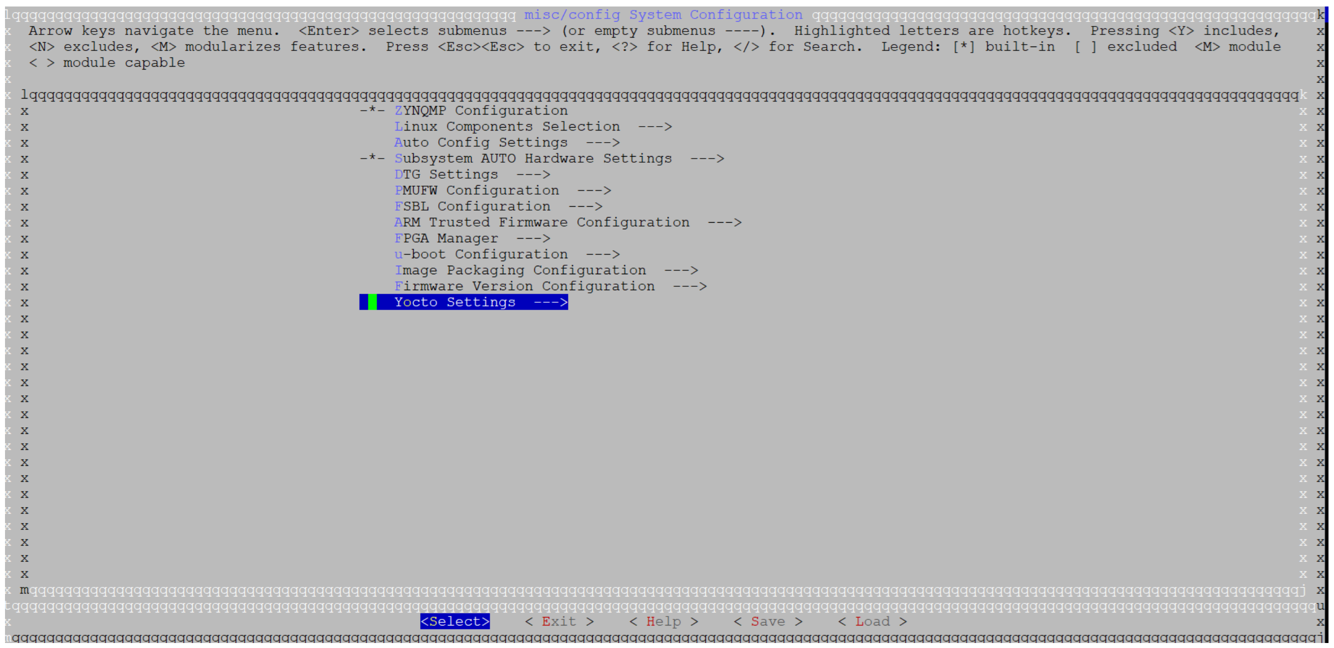
Menuconfig Usage

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**

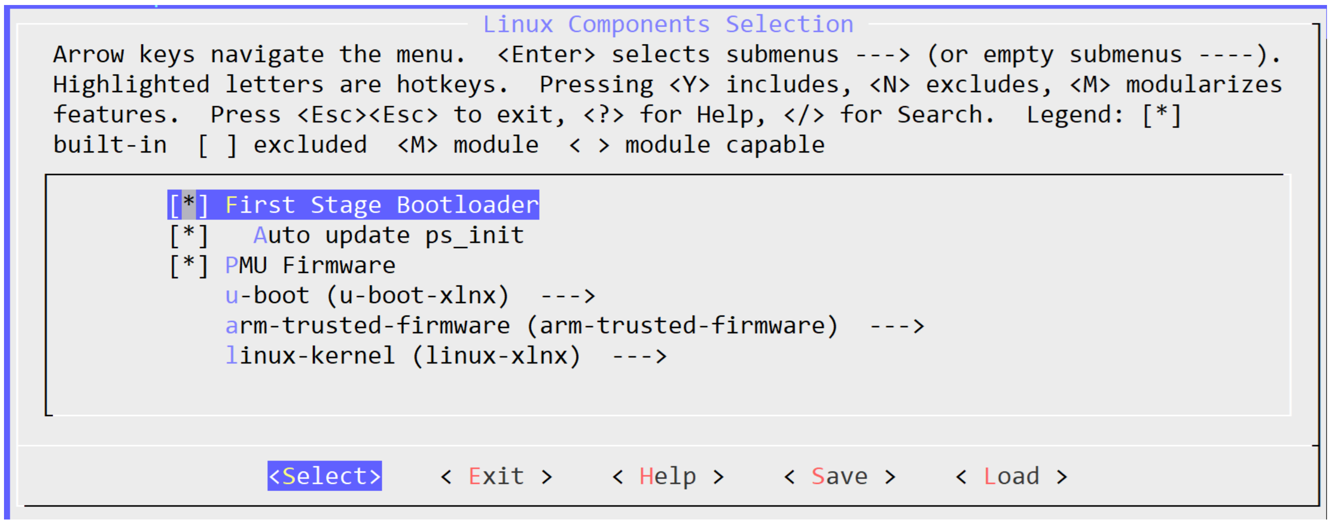


```
misc/config System Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.  Pressing <Y> includes,
  <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module
  < > module capable

                                       -*- ZYNQMP Configuration
                                           Linux Components Selection  --->
                                           Auto Config Settings  --->
                                       -*- Subsystem AUTO Hardware Settings  --->
                                           DTG Settings  --->
                                           PMUFW Configuration  --->
                                           FSBL Configuration  --->
                                           ARM Trusted Firmware Configuration  --->
                                           FPGA Manager  --->
                                           u-boot Configuration  --->
                                           Image Packaging Configuration  --->
                                           Firmware Version Configuration  --->
                                           Yocto Settings  --->

                                              <Select>    < Exit >    < Help >    < Save >    < Load >
```

Linux Components Selection

For Zynq UltraScale+ MPSoC, the Linux system components available in the sub-menu are as follows:

Figure 3: **Linux Components Selection**



```
                   Linux Components Selection
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*]
  built-in  [ ] excluded  <M> module  < > module capable

          [*] First Stage Bootloader
          [*]   Auto update ps_init
          [*] PMU Firmware
              u-boot (u-boot-xlnx)  --->
              arm-trusted-firmware (arm-trusted-firmware)  --->
              linux-kernel (linux-xlnx)  --->

           <Select>    < Exit >    < Help >    < Save >    < Load >
```

- First stage boot loader
- PMU firmware, for Zynq® UltraScale+™ MPSoC only
- U-Boot
- Kernel
- ATF, for Zynq UltraScale+ MPSoC

For ATF, U-Boot, and kernel there are three available options:

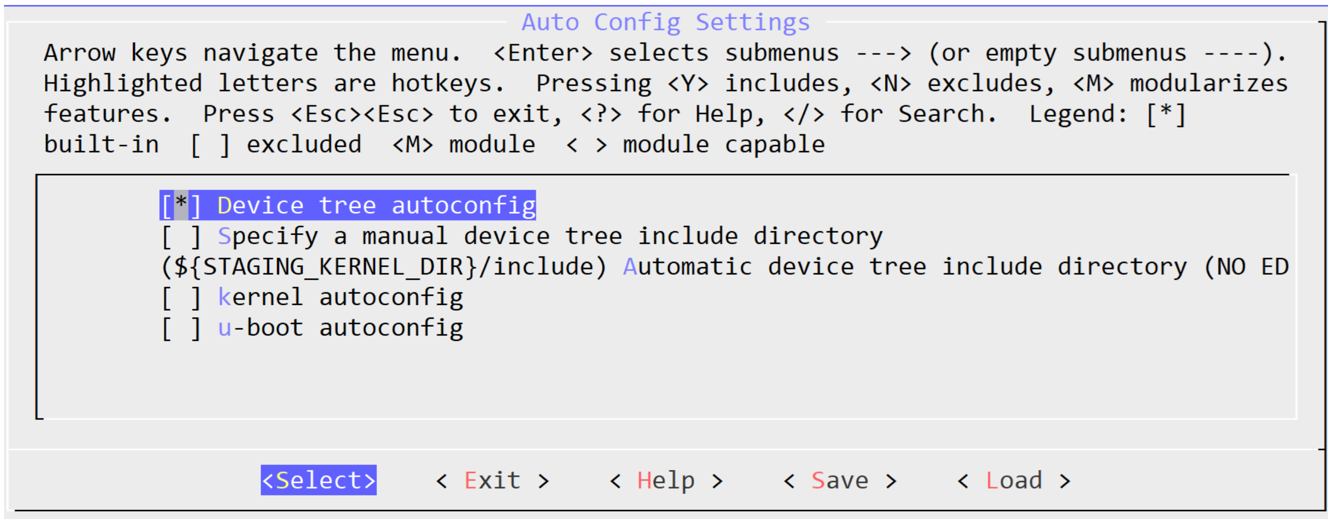
- **Default:** The default component is shipped through PetaLinux tool.
- **External source:** When you have a component downloaded at any specified location, you can feed your component instead of the default one through this configuration option.

  ***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.



```
                          Auto Config Settings
 Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
 Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
 features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*]
 built-in  [ ] excluded  <M> module  < > module capable

        [*] Device tree autoconfig
        [ ] Specify a manual device tree include directory
        (${STAGING_KERNEL_DIR}/include) Automatic device tree include directory (NO ED
        [ ] kernel autoconfig
        [ ] u-boot autoconfig

            <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 4: **Auto Config Settings**

Table 20: **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 and Zynq UltraScale+ MPSoC) • `pl.dtsi` • `system-conf.dtsi` • `system-top.dts` • `<board>.dtsi`
kernel	The following files are in `<plnx-proj-root>/project-spec/configs/linux-xlnx/plnx_kernel.cfg`

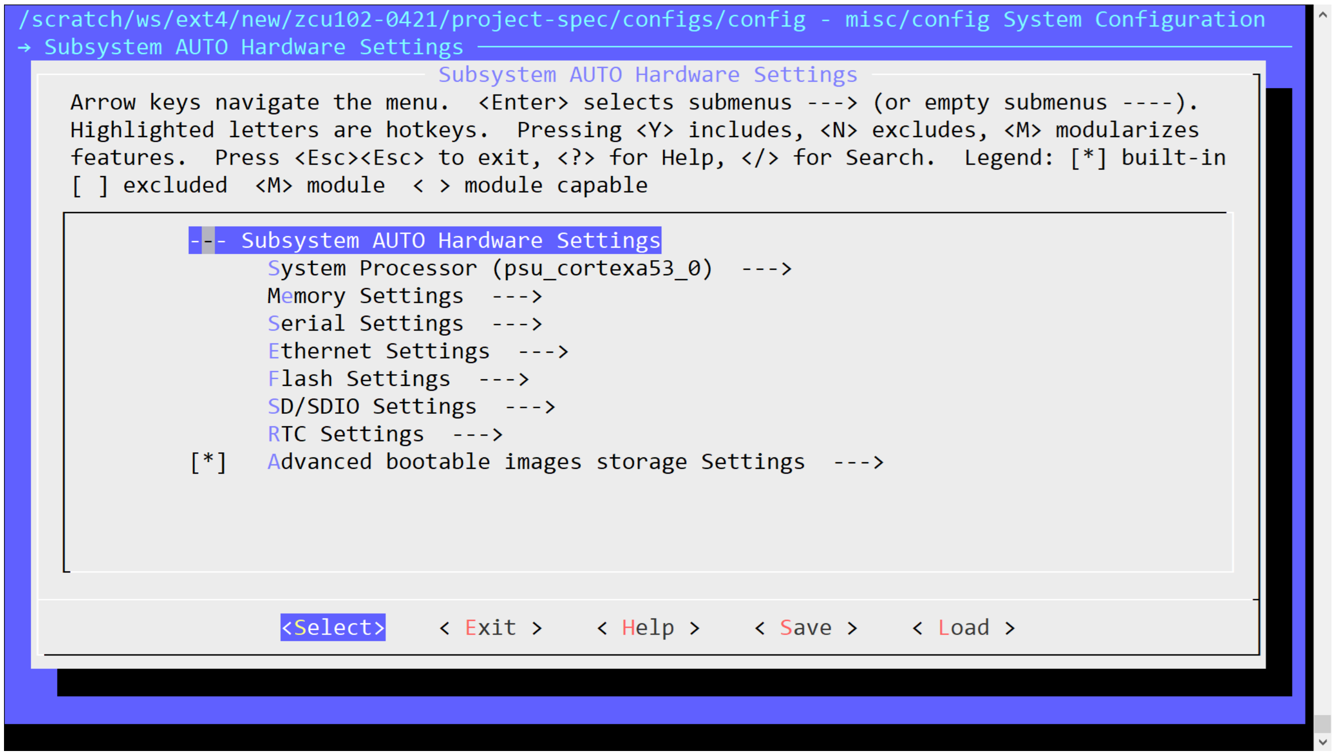
Table 20: **Components and their Configuration Files** *(cont'd)*

Component in the Menu	Files Impacted when the Autoconfig is enabled
U-Boot	The following files are in `<plnx-proj-root>/project-spec/configs/u-boot-xlnx/config.cfg` • `config.mk` (MicroBlaze only) • `platform-auto.h`

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 5: **Subsystem AUTO Hardware Settings**



```
/scratch/ws/ext4/new/zcu102-0421/project-spec/configs/config - misc/config System Configuration
→ Subsystem AUTO Hardware Settings
                         Subsystem AUTO Hardware Settings
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in
  [ ] excluded  <M> module  < > module capable

           --- Subsystem AUTO Hardware Settings
                 System Processor (psu_cortexa53_0)  --->
                 Memory Settings  --->
                 Serial Settings  --->
                 Ethernet Settings  --->
                 Flash Settings  --->
                 SD/SDIO Settings  --->
                 RTC Settings  --->
           [*]   Advanced bootable images storage Settings  --->

                 <Select>    < Exit >    < Help >    < Save >    < Load >
```

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 primary Ethernet

If MAC address is programmed into EEPROM, keep this empty here. Refer to the U-Boot documentation for commands to program EEPROM and to configure for the same.

- 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 which flash is the system's primary flash
- 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

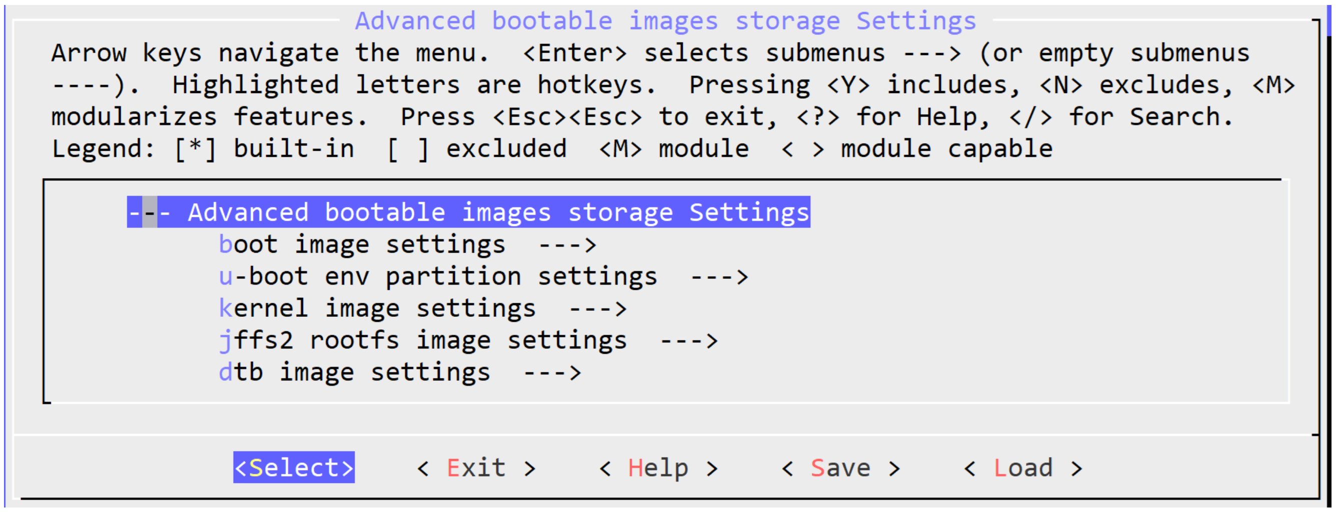
Select an RTC instance that is used as a primary timer for the Linux kernel. If your preferred RTC is not on the list, select **manual** to enable the proper kernel driver for your RTC.

Advanced Bootable Images Storage Settings

The advanced bootable images storage settings sub-menu allows you to specify where the bootable images are located. The settings in this sub-menu are used by PetaLinux to configure U-Boot.

If this sub-menu is disabled, PetaLinux uses the flash partition table specified in the Flash Settings sub-menu to define the location of the bootable images.

Figure 6: **Advanced Bootable Images Storage Settings**



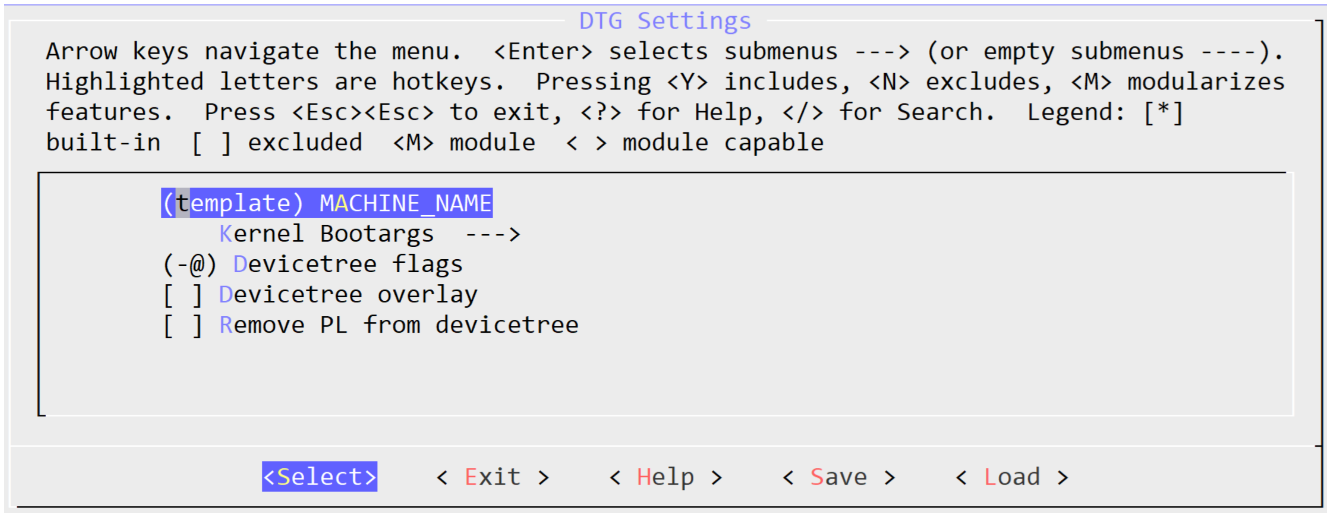
```
                    Advanced bootable images storage Settings
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus
  ----).  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M>
  modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.
  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

        --- Advanced bootable images storage Settings
              boot image settings  --->
              u-boot env partition settings  --->
              kernel image settings  --->
              jffs2 rootfs image settings  --->
              dtb image settings  --->

              <Select>    < Exit >    < Help >    < Save >    < Load >
```

Table 21: **Flash Partition Table**

Bootable Image/U-Boot Environment Partition	Default Partition Name	Description
Boot Image	boot	`BOOT.BIN` for Zynq-7000 devices and Zynq UltraScale+ MPSoC. 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**



```
                              DTG Settings
 Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
 Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
 features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*]
 built-in  [ ] excluded  <M> module  < > module capable

        (template) MACHINE_NAME
            Kernel Bootargs  --->
        (-@) Devicetree flags
        [ ] Devicetree overlay
        [ ] Remove PL from devicetree

         <Select>    < Exit >    < Help >    < Save >    < Load >
```

Machine Name

For custom boards it is template. For Xilinx® evaluation boards the following values are supported:

ac701-full, ac701-lite, kc705-full, kcu105, zcu1275-revb, zcu1285-reva, zc1751-dc1, zc1751-dc2, zc702, zc706, avnet-ultra96-rev1, zcu100-revc, zcu102-rev1.0, zcu104-revc, zcu106-reva, zcu111-reva, zedboard, vcu118- rev2.0, sp701-rev1.0,zcu216-reva, zcu208-reva.

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
```

Note: In Zynq UltraScale+ MPSoC, if you want to see kernel panic prints on console, add `earlycon console=<device>,<baud rate> clk_ignore_unused root=/dev/ram0 rw`. Example: `earlycon console=/dev/ttyPS0,115200 clk_ignore_unused root=/dev/ram0 rw` in `system_user.dtsi`.

For more information, see kernel documentation.

Device Tree Overlay Configuration for Zynq-7000 Devices and Zynq UltraScale+ MPSoC

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`.

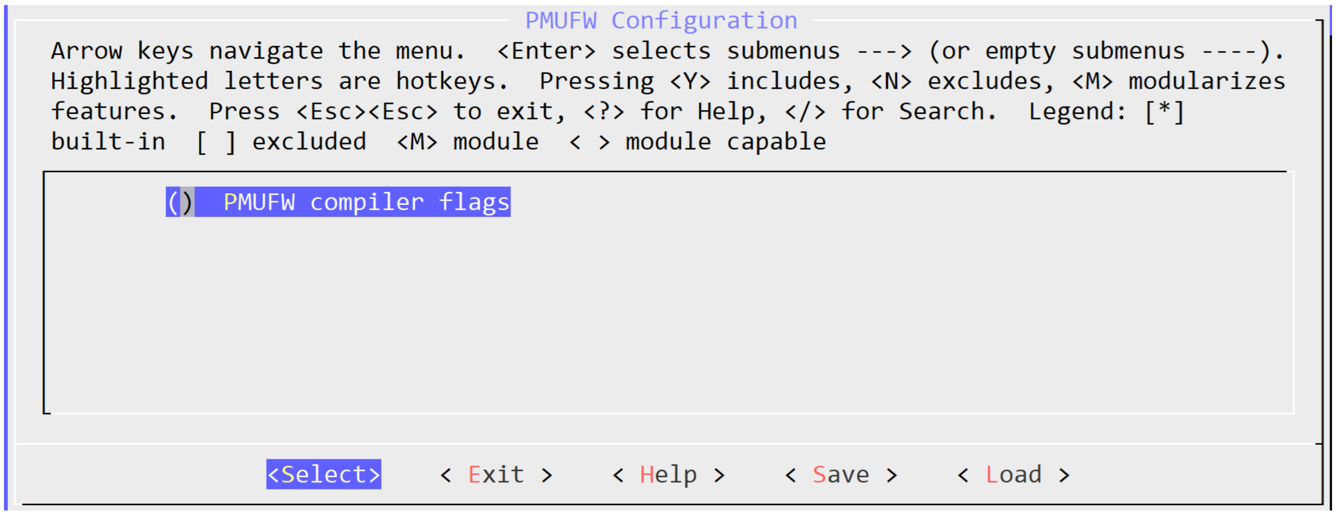
Note: FPGA manager overrides all these options. This come into play only when FPGA manager is not selected.

Note: If you select both device tree overlay and remove PL from device tree, then base DTB has entry for overlay support but there is no PL DTBO 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**

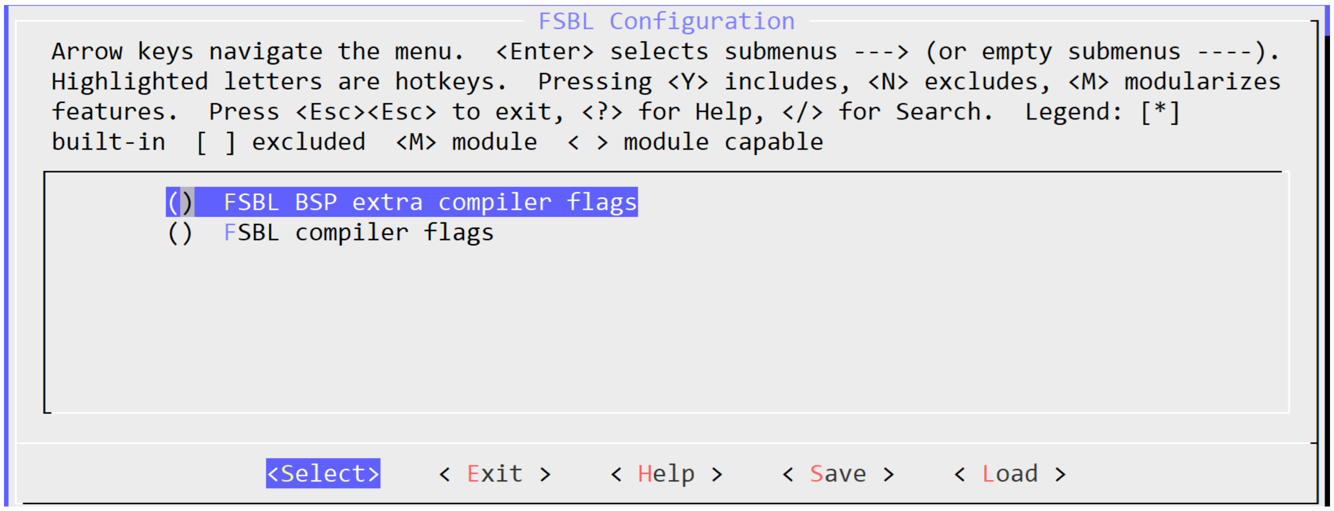


```
                          PMUFW Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*]
  built-in  [ ] excluded  <M> module  < > module capable

         ()  PMUFW compiler flags

              <Select>    < Exit >    < Help >    < Save >    < Load >
```

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**



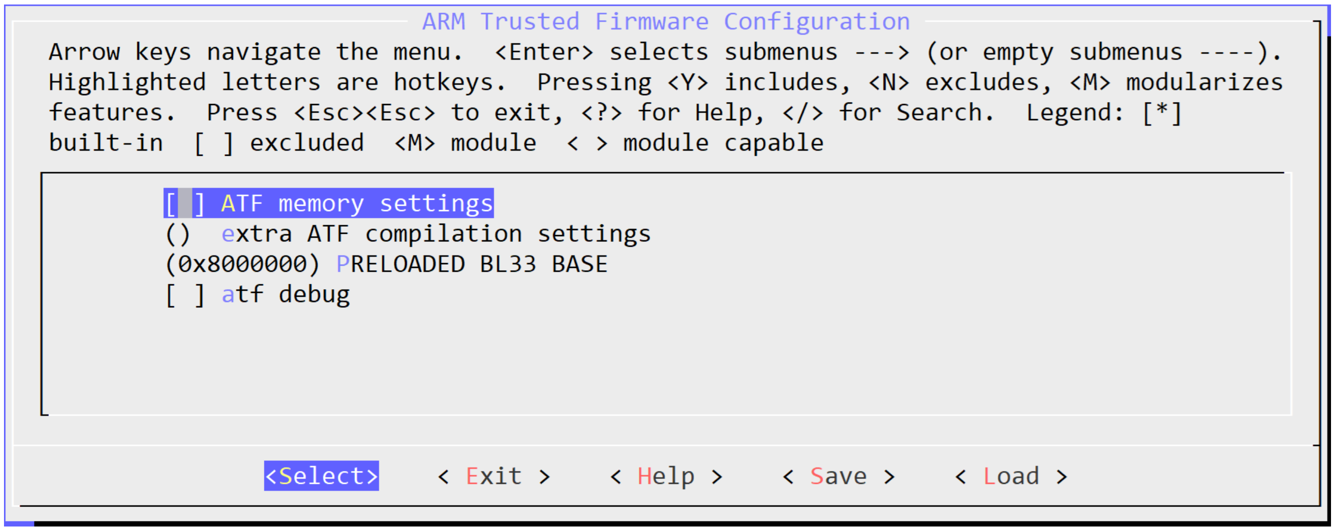
```
                         FSBL Configuration
 Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
 Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
 features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*]
 built-in  [ ] excluded  <M> module  < > module capable

        ()  FSBL BSP extra compiler flags
        ()  FSBL compiler flags

      <Select>    < Exit >    < Help >    < Save >    < Load >
```

Arm Trusted Firmware Configuration for Zynq UltraScale+ MPSoC

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**



```
                    ARM Trusted Firmware Configuration
 Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
 Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
 features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*]
 built-in  [ ] excluded  <M> module  < > module capable

        [ ] ATF memory settings
        ()  extra ATF compilation settings
        (0x8000000) PRELOADED BL33 BASE
        [ ] atf debug

      <Select>    < Exit >    < Help >    < Save >    < Load >
```

FPGA Manager Configuration and Usage for Zynq-7000 Devices and Zynq UltraScale+ MPSoC

FPGA manager provides an interface to the Linux for configuring the programmable logic (PL). It packs bitstreams and dtbos to the `/lib/firmware` directory in the root file system.

After creating a PetaLinux project for Zynq UltraScale+ MPSoC, follow the following steps to build FPGA manager support:

1. Go to `cd <plnx-proj-root>`.
2. In the `petalinux-config` command, select **FPGA Manager → [*] Fpga Manager**.

   Note: PetaLinux FPGA manager configuration when selected:

   1. Generates the `pl.dtsi` nodes as a dt overlay (dtbo).
   2. Packs bitstreams in `.bin` form and dtbos to the `/lib/firmware/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 in **FPGA Manager → Specify hardware directory path**.

   Note: This step is optional. It is required only if multiple bitstreams for same PS and corresponding dtbos, need to be packed into the root file system. It generates and pack bitstream in .bin form and its dtbo in the RootFS at `/lib/firmware/<XSA name>`. Ensure that PS design is same for XSA at hw directory path and `<plnx-proj-root>/project-spec/hw-description/system<.xsa>`.
4. Run `petalinux-build`.

Example loading full bitstream on target:

```
root@xilinx-zcu102-2020_1:~# fpgautil -o /lib/firmware/base/pl.dtbo -b
/lib/firmware/base/design_1_wrapper.bit.bin

Time taken to load DTBO is 239.000000 milli seconds. DTBO loaded through
ZynqMP FPGA manager successfully.
```

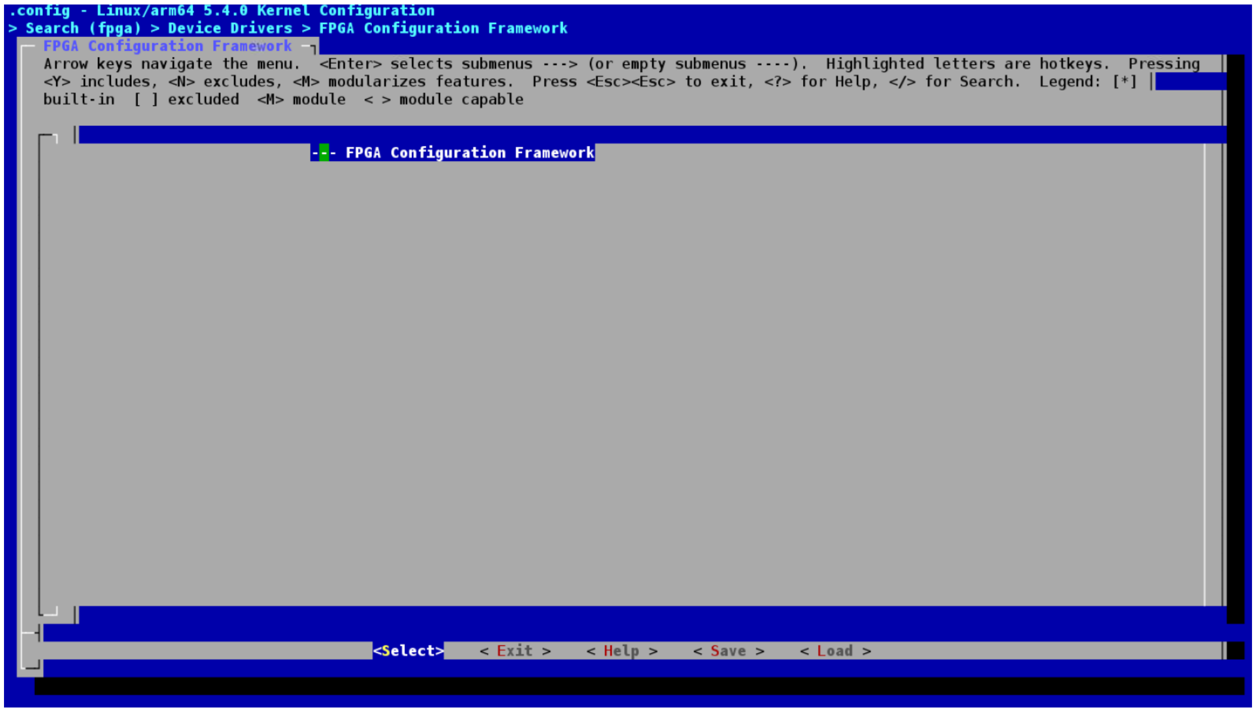
Refer to `petalinux-package` command for generating BOOT.BIN.

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**


```
.config - Linux/arm64 5.4.0 Kernel Configuration
> Search (fpga) > Device Drivers > FPGA Configuration Framework
  FPGA Configuration Framework
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.  Pressing
  <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*]
  built-in  [ ] excluded  <M> module  < > module capable

                          --- FPGA Configuration Framework

                    <Select>    < Exit >    < Help >    < Save >    < Load >
```

Adding Custom dtsi and bit Files to the FPGA Manager for Zynq-7000 Devices and Zynq UltraScale+ MPSoCs

This section provides 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` directory and loads them when the system boots.

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 <project-root-dir>/
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 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. Build the application:

```
$ petalinux-build
```

4. Check the target for dtbo and .bin files:

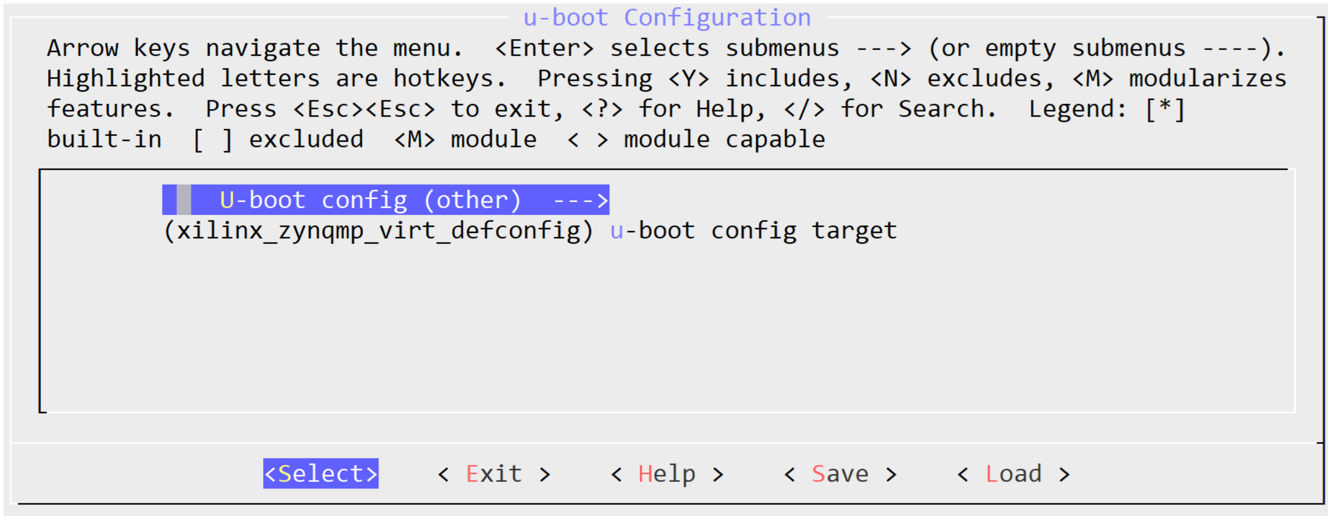
```
$ ls /lib/firmware/can-interface/
                pl.dtbo    system.bit.bin
```

To stop loading the dtbo and .bin files at system boot, add `FPGA_INIT = "0"` to the `<project-root-dir>/project-spec/meta-user/recipes-apps/can-interface/can-interface.bb` file.

U-Boot Configuration

The U-Boot configuration sub-menu allows you to select a U-Boot automatic configuration (autoconfig) by PetaLinux or a U-Boot board configuration target.

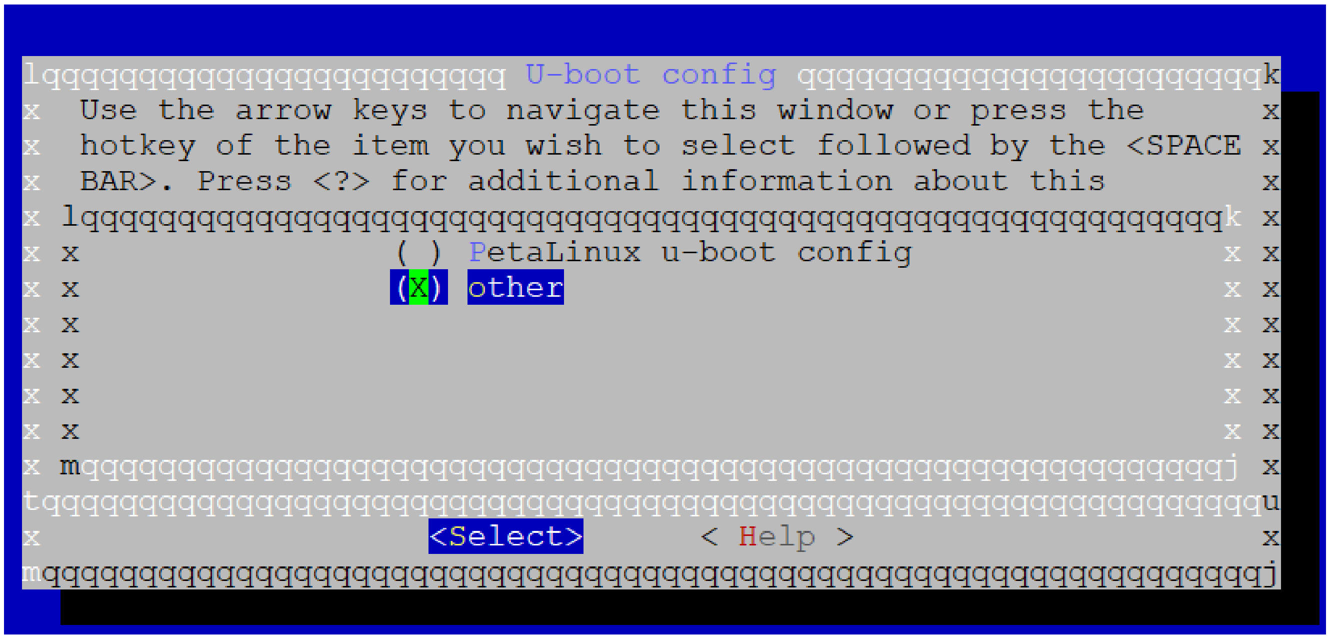
Figure 12: **U-Boot Configuration**



```
                              u-boot Configuration
 Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
 Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
 features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*]
 built-in  [ ] excluded  <M> module  < > module capable

        U-boot config (other)  --->
        (xilinx_zynqmp_virt_defconfig) u-boot config target

              <Select>    < Exit >    < Help >    < Save >    < Load >
```

By default, PetaLinux uses the board configuration "other". If you want configurations from design, select **PetaLinux u-boot config**.

Figure 13: **U-Boot Default Configuration**



```
lqqqqqqqqqqqqqqqqqqqqqqqq U-boot config qqqqqqqqqqqqqqqqqqqqqqqqk
x  Use the arrow keys to navigate this window or press the     x
x  hotkey of the item you wish to select followed by the <SPACE x
x  BAR>. Press <?> for additional information about this        x
x lqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqk x
x x                ( ) PetaLinux u-boot config                x x
x x                (X) other                                  x x
x x                                                           x x
x x                                                           x x
x x                                                           x x
x x                                                           x x
x mqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqj x
tqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqu
x                     <Select>      < Help >                    x
mqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqj
```

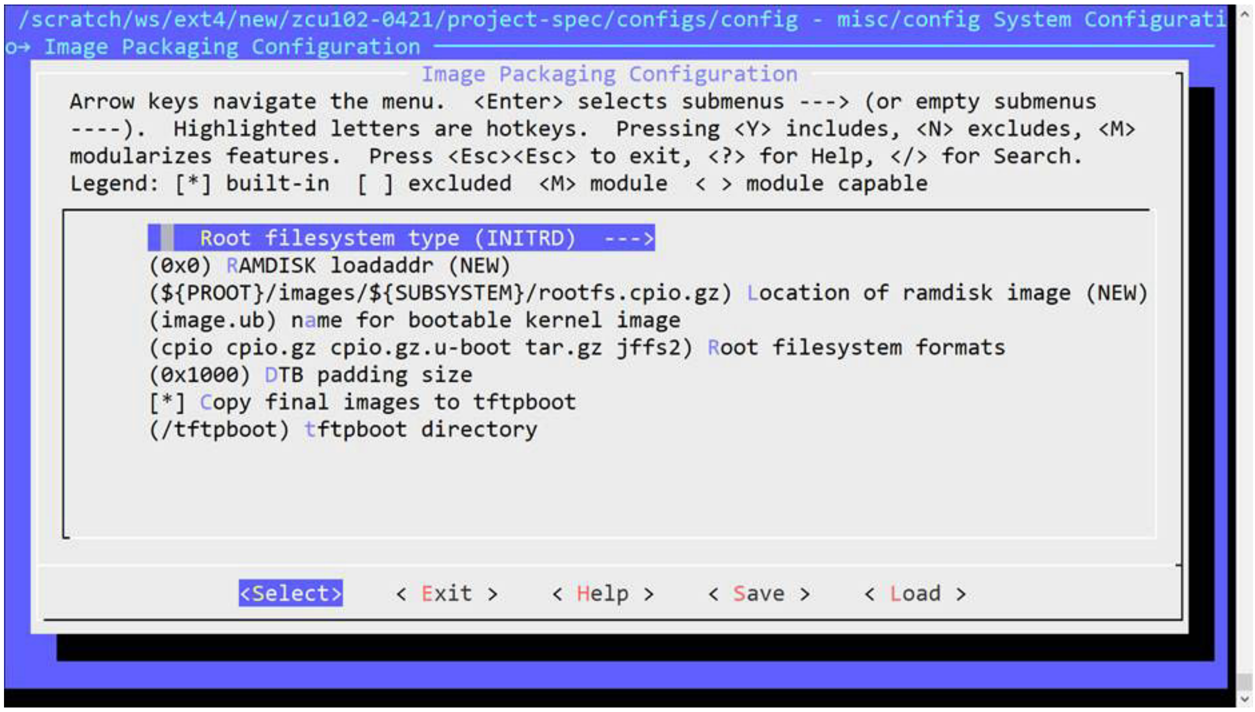
Image Packaging Configuration

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**



```
 /scratch/ws/ext4/new/zcu102-0421/project-spec/configs/config - misc/config System Configurati
o→ Image Packaging Configuration
                          Image Packaging Configuration
    Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus
    ----).  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M>
    modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.
    Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

            Root filesystem type (INITRD)  --->
          (0x0) RAMDISK loadaddr (NEW)
          (${PROOT}/images/${SUBSYSTEM}/rootfs.cpio.gz) Location of ramdisk image (NEW)
          (image.ub) name for bootable kernel image
          (cpio cpio.gz cpio.gz.u-boot tar.gz jffs2) Root filesystem formats
          (0x1000) DTB padding size
          [*] Copy final images to tftpboot
          (/tftpboot) tftpboot directory

                 <Select>    < Exit >    < Help >    < Save >    < Load >
```

***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.

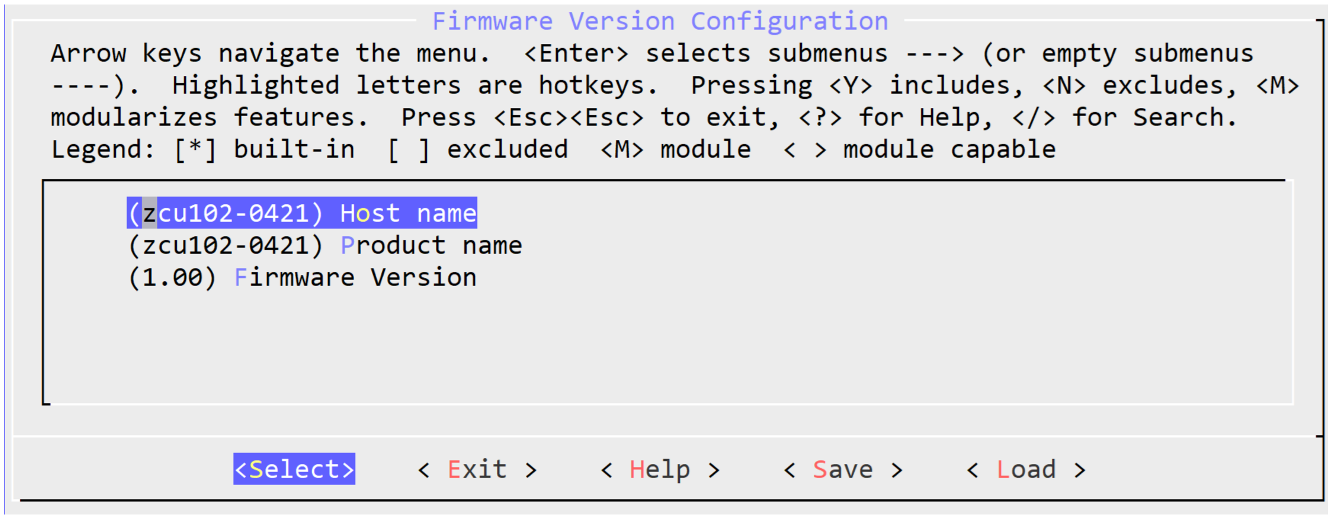
Firmware Version Configuration

The Firmware Version Configuration sub-menu allows you to set the firmware version information:

Table 22: **Firmware Version Options**

Firmware Version Option	File in the Target RootFS
Host name	`/etc/hostname`
Product name	`/etc/petalinux/product`
Firmware Version	`/etc/petalinux/version`

Figure 15: **Firmware Version Configuration**



```
                  Firmware Version Configuration
 Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus
 ----).  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M>
 modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.
 Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

      (zcu102-0421) Host name
      (zcu102-0421) Product name
      (1.00) Firmware Version

           <Select>    < Exit >    < Help >    < Save >    < Load >
```

TIP: *The host name does not get updated. Please see Xilinx Answer 69122 for more details.*

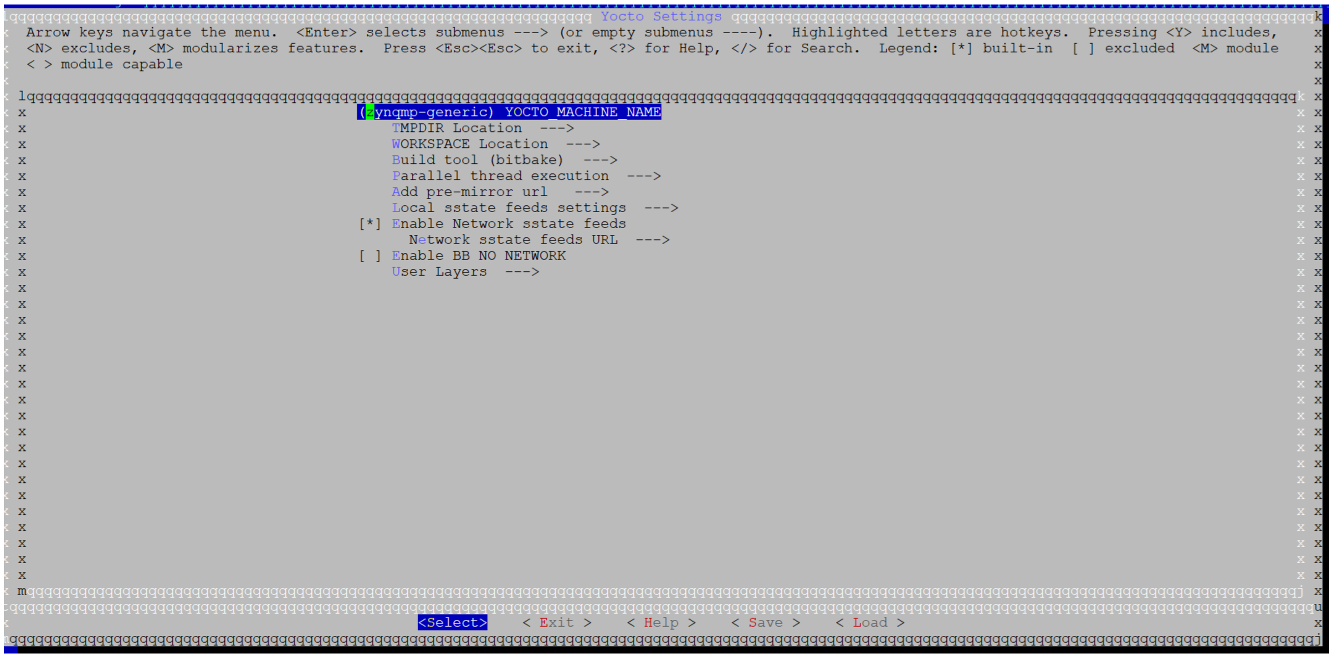
Yocto Settings

Yocto settings allows you to configure various Yocto features available in a project.

Table 23: **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
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
User layers	Adds user layers into projects
BB_NO_NETWORK	When enabled, internet access is disabled on the build machine

Figure 16: **Yocto Settings**



```
                                   Yocto Settings
 Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.  Pressing <Y> includes,
 <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module
 < > module capable

                                        (zynqmp-generic) YOCTO_MACHINE_NAME
                                            TMPDIR Location  --->
                                            WORKSPACE Location  --->
                                            Build tool (bitbake)  --->
                                            Parallel thread execution  --->
                                            Add pre-mirror url   --->
                                            Local sstate feeds settings  --->
                                        [*] Enable Network sstate feeds
                                              Network sstate feeds URL  --->
                                        [ ] Enable BB NO NETWORK
                                            User Layers  --->

                                               <Select>    < Exit >    < Help >    < Save >    < Load >
```

Open Source Bootgen for On-target Use for Zynq Devices 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}
  ```

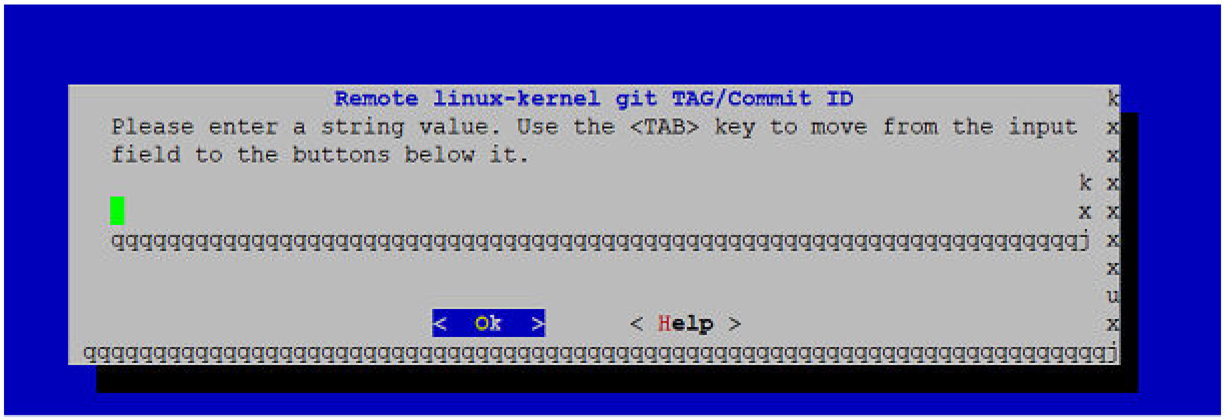
- To point to any tag:

  ```
  tag/mytag
  ```

- To point to any commit id:

  ```
  commit id sha key
  ```

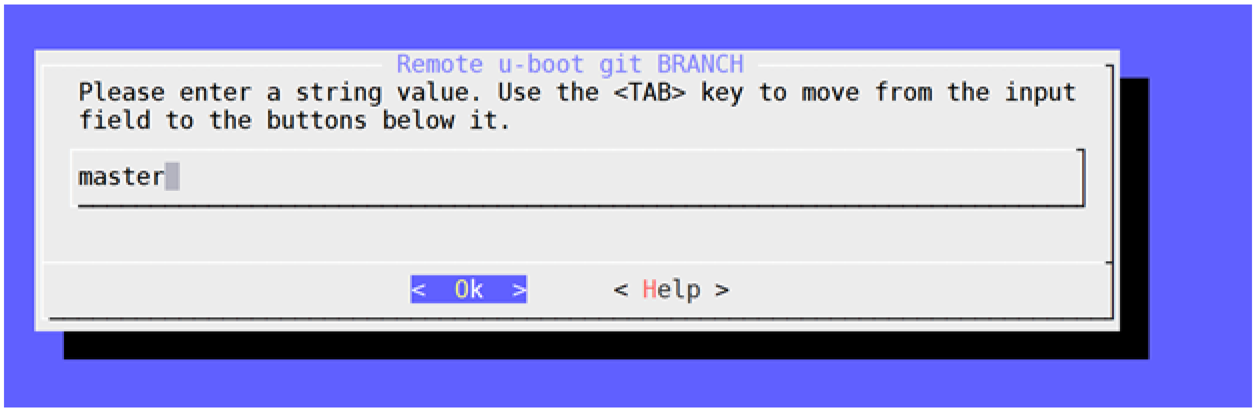
  Once you select git Tag/Commit ID, you can see a prompt to enter a string value as shown in the following figure. Enter any of the above set values.



```
Remote linux-kernel git TAG/Commit ID
Please enter a string value. Use the <TAB> key to move from the input
field to the buttons below it.
```

5. To specify BRANCH to `kernel/u-boot/arm-trusted-firmware`, select **Remote settings (Optional)**.

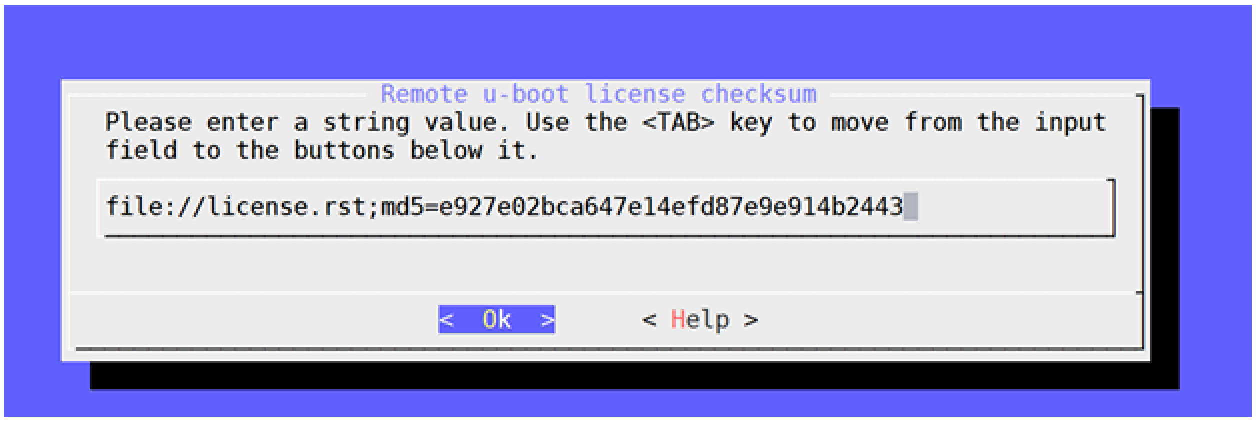
   For example: To specify the master branch, type `master`as shown in the following figure:



```
Remote u-boot git BRANCH
Please enter a string value. Use the <TAB> key to move from the input
field to the buttons below it.

master
```

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:



```
Remote u-boot license checksum
Please enter a string value. Use the <TAB> key to move from the input
field to the buttons below it.

file://license.rst;md5=e927e02bca647e14efd87e9e914b2443
```

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-build -c linux-xlnx -x reset' 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 24: **Configuring Out-of-Tree Build Troubleshooting**

Problem / Error Message	

`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
   ```

***Note*:** `petalinux-config -c <COMPONENT>` command creates the workspace directory `<plnx-proj-root>/components/yocto/workspace/sources/<COMPONENT>` by fetching the source. For example, `petalinux-config -c u-boot` fetches the U-Boot source and create the workspace in `<plnx-proj-root>/components/sources/u-boot-xlnx`. You can use this in your development.

***Note*:** Using `petalinux-config -c <COMPONENT>` the component changes will be stored in workspace directory (`<project-root-dir>/components/yocto/workspace`). To apply workspace changes to the recipe in the meta-user user must run `-x` finish command to return their build location, for example, `petalinux-build -c <COMPONENT> -x finish`.

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 devices, xilinx_zynq_virt_defconfig
- For Zynq UltraScale+ MPSoC, xilinx_zynqmp_virt_defconfig
- For MicroBlaze processors, microblaze-generic_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/fsbl/fsbl_%.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-projroot>/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/`.

For setting U-Boot environment variables, edit CONFIG_EXTRA_ENV_SETTINGS variable in `platform-auto.h`. Note that `platform-auto.h` is regenerated each time `petalinux-config` is run.

CAUTION! *`config.cfg` and `platform-auto.h` files are automatically generated; edit them with caution.*

PetaLinux does not currently automate U-Boot configuration with respect to `CONFIG_` options/settings. You can add these `CONFIG_` options/settings into `platform-top.h` file.

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.

Chapter 11

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 `<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>/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 `<proj_root>/images/linux/sdk.sh`.
   d. Install SDK.

      ```
      petalinux-package --sysroot
      ```

      This command installs SDK at `<proj_root>/images/linux/sdk`.
   e. Use the `images/linux/sdk` directory as sysroot for building the OpenCV applications.

Building and Installing eSDK

Building eSDK

The following command builds the eSDK(extensible SDK) and copies it at `<proj_root>/images/linux/esdk.sh`.

```
petalinux-build --esdk
```

The following is the equivalent BitBake command.

```
bitbake petalinux-image-minimal -c do_populate_sdk_ext
```

Installing eSDK

To install the eSDK, follow these steps:

1. Source the PetaLinux tool.
2. Run: `petalinux-upgrade -f <esdk path> -p <platform>`

Packaging Sources and Licenses

In PetaLinux, you can package all the sources and licenses of the built packages which you build as part `petalinux-build/petalinux-build --sdk` to this, follow these steps.

1. Create a project.
2. Go to the project.
3. To pack all the components of `petalinux-build`, issue the following commands.

   ```
   petalinux-build --archiver
   ```

4. 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-v2020/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 `<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 `<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 25: **Machine Name Change for Templates**

Template	Machine
zynq	zynq-generic
zynqmp	zynqmp-generic
microblaze	microblazeel-v11.0-bs-cmp-mh-div-generic

Table 26: **Machine Name Change for BSPs**

BSP	Machine
zc702	zc702-zynq7
zc706	zc706-zynq7
zcu102 (All variants)	zcu102-zynqmp
zcu106	zcu106-zynqmp
zcu104	zcu104-zynqmp
kc705	microblazeel-v11.0-bs-cmp-mh-div-generic
ac701	microblazeel-v11.0-bs-cmp-mh-div-generic
kcu105	microblazeel-v11.0-bs-cmp-mh-div-generic
zcu111	zcu111-zynqmp
zcu1285	zcu1285-zynqmp
zcu1275	zcu1275-zynqmp
sp701	microblazeel-v11.0-bs-cmp-mh-div-generic
zcu216	zcu216-zynqmp
zcu208	zcu208-zynqmp

Table 26: **Machine Name Change for BSPs** *(cont'd)*

BSP	Machine
zed	zedboard-zynq7
zynqmp-common	zynqmp-generic
zynq-common	zynq-generic

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.

***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>/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. Select devtool as the build tool: **petalinux-config → Yocto settings → Build tool (devtool)**
2. Get the source code from git url specified in meta-layers:

   ```
   petalinux-build -c <recipe-name> -x modify
   ```

   Note: `petalinux-build -c <recipe-name>` or `petalinux-config -c <recipe-name>` will automatically fetch the source to the workspace directory.

The above 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 isn't already one.

3. Make the changes you want to make to the source.
4. 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.
5. Optional: Test your changes on the target.
6. 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
   ```

   `petalinux-build -c <recipe-name> -x finish` creates a patch for the committed changes in recipe sources directory and removes the source from workspace.

   `petalinux-build -c <recipe-name> -x update-recipe` creates a patch for the committed changes in recipe sources directory.
7. Once you have finished working on the recipe, run `devtool reset <recipe-name>` to remove the source directory for the recipe from workspace.

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
```

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-2020.1/build/tmp` cannot be located on NFS.

When TMPDIR is on NFS, BitBake throws an error at the time of parsing. You have to change it from `petalinux-config` and then provide any local storage. To do this, select **Yocto-settings → TMPDIR**.

Do not configure the same TMPDIR for two different PetaLinux projects. This can cause build errors.

Recipe Name Having ' _ '

If the app name is `plnx_myapp`, BitBake throws an error. A version number has to be entered after ' _ '.

For example, `myapp_1` is an accurate app/module name.

To recover, you have to sstateclean the app created and then delete it. Also, delete the line in `conf/user-rootfsconfig`.

```
CONFIG_plnx_myapp
```

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`.

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-project-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 `<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 UltraScale+ MPSoC. Use the following steps to configure and use the package management system:

1. Enable DNF through `petalinux-config -c rootfs`. Enable the following configs to use DNF.

   - Image Features --> [*] package management
   - Set the package feed url from Image Features --> http://petalinux.xilinx.com/sswreleases/rel-v2020/feeds/ultra96-zynqmp) Package feed url.

     Check the available package feeds in http://petalinux.xilinx.com/sswreleases/rel-v2020/feeds.

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-v2020-feeds-ultra96-zynqmp]
   name=OE Remote Repo: sswreleases rel-v2020 feeds ultra96-zynqmp
   baseurl=http://petalinux.xilinx.com/sswreleases/rel-v2020/feeds/ultra96-
   zynqmp
   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.

Linux Boot Hang with Large INITRAMFS Image in Zynq-7000 Devices and Zynq UltraScale+ MPSoC

When `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.

Migration

This section describes the migration details of the current release versus the previous release.

Tool/Project Directory Structure

Following is the tool directory structure:

- For earlier releases, the tool contained the minimal downloads and sstate-cache. From this release onwards, those downloads and sstate-cache are no longer a part of the tool. The project build will use sstate-cache and downloads from the petalinux-xilinx.com.
- The Yocto SDK will be part of each project `<plnx-proj-root>/components/yocto`.
- `<plnx-proj-root>/project-spec/meta-plnx-generated` has been removed. The required Yocto variables are now part of `<plnx-proj-root>/build/conf/plnxtool.conf`.
- From this release onwards, there is an option to specify user-specified eSDK/platform while installing the tool.

```
plnx.run -d <destination-path> -p <platform>
```

DT Overlay Support

- The bitstream filename in `<plnx-proj>/images/linux/` is `system.bit` but if you enable DT Overlay Support, it will be with design name.
- DT Overlay Support has been added for Zynq®-7000 devices.

Linux and U-Boot Default Configurations

For earlier releases of the tool, each PetaLinux BSP had its own Linux and U-Boot default configurations but from this release onwards, each SoC family has a default configuration. They are as follows:

- For Zynq UltraScale+ MPSoC:
  - Zynq UltraScale+ MPSoC linux defconfig: xilinx_zynqmp_defconfig
  - Zynq UltraScale+ MPSoC u-boot defconfig: xilinx_zynqmp_virt_defconfig
- For Zynq-7000 devices:
  - Zynq linux defconfig: xilinx_zynq_defconfig
  - Zynq u-boot defconfig: xilinx_zynq_virt_defconfig
- For MicroBlaze devices:
  - Microblaze linux defconfig: mmu_defconfig
  - Microblaze u-boot defconfig: microblaze-generic_defconfig

***Note*:** For MicroBlaze processors, PetaLinux generates configurations from the design on top of the default configurations for Linux and U-Boot.

Build Changes

Default PetaLinux images are built as INITRD. Use the following images for boot:

- `BOOT.BIN`, Image, `rootfs.cpio.gz.u-boot`, and `boot.scr`
- `BOOT.BIN`, `image.ub`, and `boot.scr`

***Note*:** `system.dtb` is part of `BOOT.BIN`.

In this release, there is an option to build eSDK:

```
petalinux-build --esdk
```

From this release onwards, there is an option to pack sources and licenses as tar:

```
petalinux-build --archiver
petalinux-build --sdk --archiver
```

Menuconfig Changes

In this release, **Image Packaging Configuration → Root Filesytem type → EXT (SD/eMMC/QSPI/SATA/USB)** has changed to **Image Packaging Configuration → Root Filesystem type → EXT4 (SD/eMMC/SATA/USB)**.

- To build open source bootgen and to use on target, select **petalinux-config -c rootfs → Filesystem Packages → bootgen**.
- To switch between bitbake and devtool, select **petalinux-config → Yocto Settings → Build tool**
- To provide any compilation flags for the FSBL application or the BSP, use this menuconfig option: **petalinux-config → FSBL Configuration**.

  Note: This is only applicable for Zynq and Zynq UltraScale+ MPSoC BSPs.
- To provide any compilation flags for the PMU firmware application or BSP, use this menuconfig option: **petalinux-config → PMUFW Configuration**

  Note: This is only applicable for Zynq UltraScale+ MPSoC BSPs.
- To specify devtool workspace when you are building a project in NFS and want to execute `petalinux-config -c kernel/u-boot`, use this option: **petalinux-config → Yocto settings → workspace location**.
- To the enable debug tweaks menu change to root file system configuration, use this option.**$ petalinux-config -c rootfs → Image Features → debug-tweaks**

  Note: This is enabled by default.
- In earlier releases for Zynq UltraScale+ MPSoC and Zynq devices, **petalinux-config → Auto Config Settings → kernel autoconfig/u-boot autoconfig** would be enabled by default. From this release onwards, to support the distro feature by default, **kernel autoconfig** and **u-boot autoconfig** are disabled. Setting that are enabled as part Linux or U-Boot defconfig is built. If you want read the kernel/U-Boot configs from the design (xsa file), enable the above configurations.

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. Here is an example of a PetaLinux project:

```
project-spec
   hw-description
   configs
   meta-user
pre-built
   linux
      implementation
      images
      xen
hardware
   xilinx-zcu104-2020.1
components
   plnx_workspace
      device-tree
config.project
README
```

Table 27: **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`	`platform-auto.h and platform-top.h` is copied to `include/configs/` directory in the U-Boot source code.
`/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-projroot>`.

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`.

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-dor>/project-spec/meta-user/conf/petalinuxbsp.conf`, but it increases the project space.

Table 28: **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.
`<plnx-proj-root>/build/misc/rootfs_config/`	Directory to hold files related to the RootFS build.
`${TMPDIR}/work/plnx_aarch64-xilinx-linux/petalinux-ser-image/1.0-r0/rootfs`	RootFS copy of target. This is the staging directory.

Table 28: **Build Directory in a PetaLinux Project** *(cont'd)*

Build Directory in a PetaLinux Project	Description
`${TMPDIR}/plnx_aarch64`	Stage directory to hold the libs and header files required to build user apps/libs.
`${TMPDIR}/work/plnx_aarch64-xilinx-linux/linux-xlnx`	Directory to hold files related to the kernel build.
`${TMPDIR}/work/plnx_aarch64-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 29: **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 `<proj-plnx-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.

Generating Boot Components

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. 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 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 30: **Wrapper Information**

Offset	Description	Value
0×0	FS-Boot bootable image magic code	0×b8b40008
0×4	BIN image size	User defined
0×100	FS-Boot bootable image target memory address	User defined. PetaLinux tools automatically calculate 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 tools generate the wrapper header to wrap around the U-Boot BIN file.

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. To boot your target with SMP support, follow these steps:

```
<pelnx-tool>/tools/xsct/bin/xsdb
xsdb > connect -url <target-url>
xsdb > fpga -f <plnx-proj>/images/linux/system.bit
xsdb > ta (this will list all avaiable cores)
xsdb > ta <core number>
xsdb > dow -f <plnx-proj>/images/linux/fs-boot.elf
the above two steps for all available cores.
xsdb > dow -f <plnx-proj>/images/linux/uboot.elf
xsdb > dow -f <plnx-proj>/images/linux/image.ub
```

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.

Redirecting Ports in Non-root Mode

If running QEMU in the default non-root mode, and you wish to access the internal (virtual) network from your host machine (For example, to debug with either GDB or TCF Agent), you will need to forward the emulated system ports from inside the QEMU virtual machine to the local machine. The `petalinux-boot --qemu` command utilizes the `--qemu-args` option to perform this redirection. The following table outlines some example redirection arguments. This is standard QEMU functionality, refer to the QEMU documentation for more details.

Table 31: **Redirection Arguments**

QEMU Options Switch	Purpose	Accessing guest from host
`-tftp <path-to-directory>`	Sets up a TFTP server at the specified directory, the server is available on the QEMU internal IP address of 10.0.2.2.	
`-redir tcp:10021:10.0.2.15:21`	Redirects port 10021 on the host to port 21 ftp) in the guest	host> ftp localhost 10021
`-redir tcp:10023:10.0.2.15:23`	Redirects port 10023 on the host to port 23 telnet) in the guest	host> telnet localhost 10023
`-redir tcp:10080:10.0.2.15:80`	Redirects port 10080 on the host to port 80 http) in the guest	Type `http://localhost:10080` in the web browser

Table 31: **Redirection Arguments** *(cont'd)*

QEMU Options Switch	Purpose	Accessing guest from host
`-redir tcp:10022: 10.0.2.15:22`	Redirects port 10022 on the host to port 22 ssh) in the guest	Run `ssh -P 10022 localhost` on the host to open a SSH session to the target

The following example shows the command line used to redirect ports:

```
$ petalinux-boot --qemu --kernel --qemu-args "-redir tcp:1534::1534"
```

This document assumes the use of port 1534 for gdbserver and tcf-agent, but it is possible to redirect to any free port. The internal emulated port can also be different from the port on the local machine:

```
$ petalinux-boot --qemu --kernel --qemu-args "-redir tcp:1444::1534"
```

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
```

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* (UG1169).

Appendix F

Xen Zynq UltraScale+ MPSoC Example

This section details on the Xen Zynq® UltraScale+™ MPSoC example. It describes how to get Linux to boot as dom0 on top of Xen on Zynq UltraScale+ MPSoC.

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 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 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`:

   ```
   tftpb 0xC00000 xen_boot_tftp.scr; source 0xC00000
   ```

   SD BOOT: `xen_boot_sd.scr`, to be loaded at address 0xC00000:

   ```
   load scsi 0:1 0xC00000 xen_boot_sd.scr; source 0xC00000
   ```

Rebuild Xen

After creating a PetaLinux project for Zynq UltraScale+ MPSoC, 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**.
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 xen boot files get updated with that pointed images. Edit the configuration file.

```
$ 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 `<plnx-proj-root>/images/linux`.

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`:

```
tftpb 0xC00000 xen_boot_tftp.scr; source 0xC00000
```

   SD BOOT: `xen_boot_sd.scr`, to be loaded at address `0xC00000`:

```
load scsi 0:1 0xC00000 xen_boot_sd.scr; source 0xC00000
```

   Note: For more information, see http://www.wiki.xilinx.com/XEN+Hypervisor.

Booting Prebuilt OpenAMP

Use the following steps to execute OpenAMP:

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>
```

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.
```

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. *PetaLinux Tools Documentation: PetaLinux Command Line Reference* (UG1157)
5. Xilinx Quick Emulator User Guide (QEMU) (UG1169)
6. *Libmetal and OpenAMP for Zynq Devices User Guide* (UG1186)
7. www.wiki.xilinx.com/Linux
8. PetaLinux Yocto Tips
9. Yocto Project Technical FAQ
10. *Vitis Unified Software Platform Documentation: Embedded Software Development* (UG1400)

Please Read: Important Legal Notices