PetaLinux Tools Documentation

Reference Guide

UG1144 (v2019.1) May 22, 2019

XILINX®

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
05/22/2019 Version 2019.1	
Chapter 6: Upgrading the Workspace	Added new section for `petalinux-upgrade` command.
Chapter 12: Technical FAQs	Added new section Package Management.
Chapter 10: Advanced Configurations	Updated FPGA Manager Configuration and Usage for Zynq® UltraScale+™ MPSoC and Zynq-7000 devices.

Table of Contents

XILINX

Chapter 1

Overview

Introduction

PetaLinux is an embedded Linux software development kit (SDK) targeting Xilinx® FPGA-based System-on-Chip 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)
- Minimal downloads
- XSCT and toolchains
- PetaLinux CLI tools

***Note*:** Xilinx Software Development Kit (SDK) (XSDK) is the integrated design environment (IDE) for creating embedded applications on Xilinx microprocessors.

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

Minimal Downloads

BitBake checks PREMIRRORS before looking upstream for any source files. PREMIRRORS are appropriate when you have a shared directory that is not defined by the DL_DIR variable. All projects of the tool use these PREMIRRORS and fetch the source code from them.

The PREMIRROR in tool points to: `$PETALINUX/components/yocto/downloads`. The downloads directory has tar balls of source code for Linux kernel, U-Boot, and other minimal utilities. For more information, see Downloading Mirrors.

XSCT and toolchains

For all embedded software applications, 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.

XILINX

Chapter 2

Setting Up Your Environment

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 8 cores)
  - 100 GB free HDD space
  - Supported OS:
    - Red Hat Enterprise Workstation/Server 7.4, 7.5, 7.6 (64-bit)
    - CentOS 7.4, 7.5, 7.6 (64-bit)
    - Ubuntu Linux 16.04.5, 18.04.1 (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. All of the listed Linux Workstation Environments below have the 32-bit libraries needed by the PetaLinux tool. If there are any additional toolchain packages that need 32-bit libraries on the host, install the same before issuing `petalinux-build`. The table shown below describes the required packages, and how to install them on different Linux workstation environments.
- 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 7.4, 7.5, 7.6 (64-bit)	Red Hat Enterprise Workstation/Server 7.4, 7.5, 7.6 (64-bit)	Ubuntu Linux 16.04.5, 18.04.1 (64-bit)
dos2unix	dos2unix-6.0.3-4.el7.x86_64.rpm	dos2unix-6.0.3-4.el7.x86_64.rpm	tofrodos_1.7.13+ds-2.debian.tar.xz

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

Tool / Library	CentOS 7.4, 7.5, 7.6 (64-bit)	Red Hat Enterprise Workstation/Server 7.4, 7.5, 7.6 (64-bit)	Ubuntu Linux 16.04.5, 18.04.1 (64-bit)
ip	iproute-3.10.0-74.el7.x86_64.rpm	iproute-3.10.0-74.el7.x86_64.rpm	iproute2 4.3.0-1ubuntu3
gawk	gawk-4.0.2-4.el7.x86_64.rpm	gawk-4.0.2-4.el7.x86_64.rpm	gawk (1:4.1.3+dfsg-0.1)
gcc	gcc-4.8.5-11.el7.x86_64	gcc-4.8.5-11.el7.x86_64	-
g++ (gcc-c++)	gcc-c++-4.8.5-11.el7.x86_64	gcc-c++-4.8.5-11.el7.x86_64	-
make	make 3.81	make 3.82	make 3.81
netstat	net-tools 2.0	net-tools 2.0	net-tools
ncurses devel	ncurses -devel 5.9-13	ncurses -devel 5.9-13	libncurses5 -dev
tftp server	tftp-server	tftp-server	tftpd
zlib devel (also, install 32-bit of this version)	zlib-devel-1.2.7-17.el7.x86_64.rpm	zlib-devel-1.2.7-17.el7.x86_64.rpm	zlib1g:i386
openssl devel	openssl -devel 1.0	openssl -devel 1.0	libssl -dev
flex	flex 2.5.37	flex 2.5.37	flex
bison	bison-2.7	bison-2.7.4	bison
libselinux	libselinux 2.2.2	libselinux 2.2.2	libselinux1
gnupg	gnupg	gnupg	gnupg
wget	wget	wget	wget
diffstat	diffstat	diffstat	diffstat
chrpath	chrpath	chrpath	chrpath
socat	socat	socat	socat
xterm	xterm	xterm	xterm
autoconf	autoconf	autoconf	autoconf
libtool	libtool	libtool	libtool
tar	tar:1.24	tar:1.24	tar:1.24
unzip	unzip	unzip	unzip
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	-
32-bit glibc	glibc-2.17-157.el7_3.4.i686 glibc-2.17-157.el7_3.4.x86_64	glibc-2.17-157.el7_3.4.i686 glibc-2.17-157.el7_3.4.x86_64	-
glib2-devel	glib2-devel	glib2-devel	-

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

Tool / Library	CentOS 7.4, 7.5, 7.6 (64-bit)	Red Hat Enterprise Workstation/Server 7.4, 7.5, 7.6 (64-bit)	Ubuntu Linux 16.04.5, 18.04.1 (64-bit)
automake	automake	automake	-
screen	screen	screen	screen
pax	pax	pax	pax
gzip	gzip	gzip	gzip
libstdc++	libstdc++-4.8.5-11.el7.x86_64 libstdc++-4.8.5-11.el7.i686	libstdc++-4.8.5-11.el7.x86_64 libstdc++-4.8.5-11.el7.i686	-

Quick Installation of Packages

The following sections describe the quick installation of packages for Ubuntu and Redhat/CentOS.

Ubuntu

sudo apt-get install -y gcc git make net-tools libncurses5-dev tftpd zlib1g-dev libssl-dev flex bison libselinux1 gnupg wget diffstat chrpath socat xterm autoconf libtool tar unzip texinfo zlib1g-dev gcc-multilib build-essential -dev zlib1g:i386 screen pax gzip

Redhat/CentOS

sudo yum install gawk make wget tar bzip2 gzip python unzip perl patch diffutils diffstat git cpp gcc gcc-c++ glibc-devel texinfo chrpath socat perl-Data-Dumper perl-Text-ParseWords perl-Thread-Queue python34-pip xz which SDL-devel xterm autoconf libtool zlib-devel automake glib2-devel zlib ncurses-devel openssl-devel dos2unix flex bison glibc.i686 screen pax glibc-devel.i686 compat-libstdc+-33.i686 libstdc+.i686

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

IMPORTANT! *PetaLinux 2019.1 works only with hardware designs exported from Vivado® Design Suite 2019.1.*

Installation Steps

Prerequisites

- PetaLinux Tools Installation Requirements is completed. See the Installation Requirements for more information.

- PetaLinux release package is downloaded. You can download PetaLinux installer from PetaLinux Downloads.
- Vivado® Design Suite, Xilinx® SDK, and PetaLinux versions are in sync.

Run PetaLinux Tools Installer

Without any options, the PetaLinux tools are installed into the current working directory. Alternatively, you can specify an installation path.

For example: To install PetaLinux tools under `/opt/pkg/petalinux/2019.1`:

```
$ mkdir -p /opt/pkg/petalinux/2019.1
$ ./petalinux-v2019.1-final-installer.run /opt/pkg/petalinux/2019.1
```

***Note*:** Do not change the installer permissions to CHMOD 775 as it can cause BitBake errors.

This installs the PetaLinux Tools into `/opt/pkg/petalinux/2019.1` directory.

IMPORTANT! *Once installed, you cannot move or copy the installed directory. In the above example, you cannot move or copy `/opt/pkg/petalinux` since the full path will be stored into Yocto e-SDK environment file.*

***Note*:** You cannot install the tool as root user. Ensure that `/opt/pkg/petalinux` is writeable. You can change the permissions after installation to make it globally read-execute (0755). It is not mandatory to install tool in `/opt/pkg/petalinux` directory. You can install at any desired 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 the 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 enabled to send tools usage statistics back to Xilinx. You can turn off the WebTalk feature by running the `petalinux-util --webtalk` command:

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

```
$ petalinux-util --webtalk off
```

***Note*:** To know more about shared state in the downloads area, see Build Optimizations.

Troubleshooting

This section describes some common issues you may experience while installing the PetaLinux tools. If the PetaLinux tools installation fails, the file `$PETALINUX/post-install.log` will be generated in your PetaLinux installation directory.

Table 3: **PetaLinux Installation Troubleshooting**

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

`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-v2019.1-final-installer.run: line 52: /proj/petalinux/petalinux-v2019.1_daily_latest/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 see "PetaLinux SDK Installation
     Guide" for its
     impact and solution
     ```

2. Verify that the working environment has been set:

   ```
   $ echo $PETALINUX
   ```

   Output: `/opt/pkg/petalinux`

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

Troubleshooting

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

Table 4: **PetaLinux Working Environment Troubleshooting**

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

Table 4: **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 5: **Design Flow Overview**

Design Flow Step	Tool / Workflow
Hardware Platform Creation (for custom hardware only)	Vivado® design tools
Create PetaLinux Project	`petalinux-create -t project`
Initialize 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 reference 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.

BSP reference designs 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. You will see the output, similar to the below output:

```
INFO: Create project:
INFO: Projects:
INFO:   * xilinx-zcu102-v2019.1
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>`; 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.

If you run `ls` from `/home/user`, you will see the installed 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 6: **PetaLinux BSP Installation Troubleshooting**

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

Configuring Hardware Platform with Vivado Design Suite

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

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 Vivado Tools Working Environment. If you have not, source the appropriate settings scripts as follows:

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

- You are familiar with Vivado Design Suite and Xilinx® SDK tools. For more information, see *Vivado Design Suite User Guide: Getting Started* (UG910).

Configure a Hardware Platform for Linux

You can create your own hardware platform with Vivado®. 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:

1. External memory with at least 64 MB of memory (required)
2. UART for serial console (required)
3. Non-volatile memory (optional), for example, QSPI Flash and SD/MMC
4. 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:

1. 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.*

2. External memory controller with at least 32 MB of memory (required)
3. UART for serial console (required)

4. Non-volatile memory (optional), for example, QSPI Flash and SD/MMC
5. 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:

1. 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)
2. 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 BRAM 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.

Note: Device Support Archive (DSA) is a hardware description format that will be introduced in Vivado® Design Suite 2019.1. DSA is a super set of HDF holding additional configurations that can be changed by XSCT/XSDK.

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, build the hardware bitstream. The PetaLinux project requires a hardware description file (`.hdf/.dsa` file) with information about the processing system. You can get the hardware description file by running Export Hardware from Vivado® Design Suite.

During project initialization (or update), PetaLinux generates a device tree source file, U-Boot configuration header files, and enables Linux kernel drivers based on the hardware description file. These details are explored 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 First Stage Boot Loader (FSBL) built for Cortex™-R5F boot, you will also need to build it with Xilinx® SDK because the FSBL built with PetaLinux tools is for Cortex-A53 boot. For details on how to build the FSBL for Cortex-R5F with Xilinx SDK, see the *Zynq UltraScale+ MPSoC: Software Developers Guide* (UG1137).

Creating a New PetaLinux Project

This section describes how to create a new PetaLinux project. Projects created from templates must be bound 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 UltraScale+™ MPSoC)
  - `zynq` (for Zynq-7000 devices)
  - `microblaze` (for MicroBlaze™ CPUs)

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. The following steps customize these settings to match the hardware project created previously.

If `--template` option is used instead of a BSP, you can use the `petalinux-config` command to choose default board configs that are close to your board design, as shown below:

1. `petalinux-config --get-hw-description=<PATH-TO-HDF/DSA-DIRECTORY>`
2. Set CONFIG_SUBSYSTEM_MACHINE_NAME as required.
   - The possible values are: ac701-full, ac701-lite, kc705-full, kcu105, zc1275-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
   - In `petalinux-config`, select **DTG Settings →(template) MACHINE_NAME**, change the template to any of the above mentioned possible values.

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>`. If `/tmp` is also on NFS, it will throw an error. If you want to change the TMPDIR to a local storage use **petalinux-config → Yocto-settings → TMPDIR**. Do not configure the same location as TMPDIR for two different PetaLinux projects. This may cause build errors. If TMPDIR is at `/tmp/..`, deleting the project will not clean it. You have to explicitly do this step, or use `petalinux-build -x mrproper`.*

XILINX

Chapter 4

Configuring and Building

Version Control

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

Prerequisites

This section assumes that the you have created a new PetaLinux project or have an existing PetaLinux project. See Creating a New PetaLinux Project 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>/project-spec/meta-plnx-generated/`
- `<plnx-proj-root>/components/plnx-workspace/`
- `<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.

IMPORTANT! *Version control is currently a work in progress. It is recommended that you share projects with BSP methodology.*

Note: In concurrent development, TMPDIR in `petalinux-config` should be unique for each user. Use `${PROOT}` as reference to specify the relative path of the TMPDIR before checking in the project into version control.

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 `.hdf/.dsa` 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 a New PetaLinux Project.

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 `.hdf/.dsa` file as follows:

```
$ petalinux-config --get-hw-description=<path-to-directory-containing-
hardware description-file>
```

Note: If both DSA and HDF files are placed in the hardware description directory, the DSA file is given priority over the HDF file.

This launches the top system configuration menu when `petalinux-config --get-hw-description` runs the first time for the PetaLinux project or the tool detects there is a change in the system primary hardware candidates:

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

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 may take a few minutes to complete because the tool will parse 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 config and U-Boot config, the tool will automatically enable 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 Auto Config Settings menu, see the Settings.

The `--oldconfig/--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.

***Note*:** `--oldconfig` option will be obsolete from future releases. Use `--silentconfig` instead.

Build System 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 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 bootloader (if selected), U-Boot, the Linux kernel, and a root filesystem 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 will be within the `<plnx-proj-root>/images` and `/tftpboot` directories.

The console shows the compilation progress. For example:

```
[INFO] building project
[INFO] generating Kconfig for project
[INFO] silentconfig project
[INFO] sourcing bitbake
[INFO] generating plnxtool conf
[INFO] generating meta-plnx-generated layer
[INFO] generating bbappends for project . This may take time !
[INFO] generating u-boot configuration files
[INFO] generating kernel configuration files
[INFO] generating user layers
[INFO] generating kconfig for Rootfs
[INFO] silentconfig rootfs
[INFO] generating petalinux-user-image.bb
INFO: bitbake petalinux-user-image
Parsing recipes: 100% |
##################################################################################
########################################| Time: 0:00:29
Parsing of 2777 .bb files complete (0 cached, 2777 parsed). 3812 targets,
147 skipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
Initialising tasks: 100% |
```

```
##################################################################################
####################################| Time: 0:00:05
Checking sstate mirror object availability: 100% |
##################################################################################
##############| Time: 0:00:10
Sstate summary: Wanted 923 Found 685 Missed 476 Current 0 (74% match, 0%
complete)
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
NOTE: Tasks Summary: Attempted 3316 tasks of which 2254 didn't need to be
rerun and all succeeded.
INFO: Copying Images from deploy to images
INFO: Creating images/linux directory
NOTE: 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®-7000 devices and MicroBlaze™ platforms. The RAM disk image `rootfs.cpio.gz.u-boot` will also be 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.

Warning/Error:

```
<package-name> do_package: Could not copy license file /opt/pkg/petalinux/
components/yocto/source/<arch>/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 `/opt/pkg/petalinux/components/yocto/source/<arch>/layers/core/meta/files/common-licenses/` will have 644 permissions. So they are readable by others but not writable.

Solution:

- Method 1: Modify permissions of the license files coming from the layers manually

```
$ chmod 666  /opt/pkg/petalinux/components/yocto/source/<arch>/layers/
core/meta/files/common-licenses/*
```

  When creating the hard link, the user will 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 will allow the source not 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 will not try to create the link, but the license will not be on the final image either.

Generate uImage

If you want to use uImage instead, use `petalinux-package --image`. For example:

```
$ petalinux-package --image -c kernel --format uImage
```

Note: This option only supports Zynq-7000 devices and MicroBlaze™ processors.

The uImage will be generated to the `images/linux` subdirectory of your PetaLinux project. You will then need to configure your U-Boot to boot with uImage. If you have selected `PetaLinux u-boot config` as your U-Boot config target, you can modify `<plnx-proj-root>/project-spec/meta-user/recipes-bsp/u-boot/files/platform-top.h` of your PetaLinux project to overwrite the `CONFIG_EXTRA_ENV_SETTINGS` macro to define your U-Boot boot command to boot with uImage.

Note: This option will be deprecated in feature releases as `petalinux-build` will generate the uImage.

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 PetaLinux system image. For more information, see Build System Image.

Generate Boot Image

Before executing this step, ensure you have built the hardware bitstream. 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 --format BIN --fsbl images/linux/zynqmp_fsbl.elf --
u-boot
images/linux/u-boot.elf --pmufw images/linux/pmufw.elf --fpga images/linux/
*.bit
--force
INFO: File in BOOT BIN:
"<plnx-proj-root>/images/linux/zynqmp_fsbl.elf"
INFO: File in BOOT BIN:
"/images/linux/pmufw.elf"
INFO: File in BOOT BIN:
"/images/linux/system.bit"
INFO: File in BOOT BIN:
"/images/linux/bl31.elf"
INFO: File in BOOT BIN:
"/images/linux/u-boot.elf"
INFO: Generating zynqmp binary package BOOT.BIN...
```

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 PetaLinux system image. For more information, see Build System Image.

Generate Boot Image

Before executing this step, ensure you have built the hardware bitstream. 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 Processors

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

Prerequisites

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

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 will be copied 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).

Generate Bitstream File for MicroBlaze

Prerequisites

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

Generate Bitstream

Execute the following command to generate the bitstream file for MicroBlaze™ processors.

```
$ 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.

opt-out Default Components

You can opt-out default components if not needed. You can disable FSBL and PMU firmware by un-selecting in **petalinux-config → Linux Components Selection**

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

Deselecting these components will remove these components from the default build flow.

Note: If the FSBL and PMU firmware are not built with PetaLinux, they must be built in Xilinx® SDK.

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. Use the following 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 7: **Local Mirror Servers**

Server	Description
downloads	Source of download files are available in http://petalinux.xilinx.com/sswreleases/rel-v${PETALINUX_VER}/downloads
aarch64	sstate mirrors for Zynq® UltraScale+™ MPSoC
arm	sstate mirrors for Zynq UltraScale+ MPSoC
mb-full	sstate mirrors for MicroBlaze™ processors
mb-lite	sstate mirrors for MicroBlaze designs

CAUTION! *For building Zynq UltraScale+ MPSoC PetaLinux BSPs with Video codec, the access to this sstate through petalinux.xilinx.com or local is a must.*

Source Mirrors

You can set source mirrors through **petalinux-config → Yocto-settings → Add pre-mirror URL**. `file:///<sstate path>/downloads` for all projects. Save the configuration to use the download mirrors and verify the changes in `build/conf/plnxtool.conf`. For example: `file:///proj/petalinux/released/Petalinux-v${PETALINUX_VER}/sstate-rel-v${PETALINUX_VER}/downloads`.

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.
- 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-v${PETALINUX_VER}/aarch64/sstate-cache` by default.

Building Ignoring Dependencies

Default image configuration has initramfs enabled. This leads to multiple dependencies, such as:

- Kernel needs RootFS to be built for initramfs
- Building RootFS builds FSBL, PMU firmware, and ATF as they are part of complete images
- Device tree needs kernel headers
- U-Boot needs device tree as it compiles with the Linux device tree

You can build components individually by handling dependencies explicitly (`petalinux-build -b component`). This option has to be handled very carefully as it builds the specified recipe/tasks ignoring its dependencies. Its usage may lead to multiple intermittent errors if dependencies are not resolved explicitly by the user. To clean the project on random error, use `petalinux-build -x mrproper`.

initramfs Mode

The default mode in the PetaLinux BSPs is the initramfs mode. This mode has multiple dependencies, such as:

- Kernel needs RootFS to be built for initramfs
- Building RootFS builds FSBL, PMU firmware, and ATF
- Device tree needs kernel headers
- U-Boot needs device tree as it compiles with the Linux device tree

Hence, building the device tree builds all the components.

Example 1: Build Device tree Only

The below example shows the steps to generate device-tree from PetaLinux project. The device-tree recipe depends on HDF, native tools (dtc, python-yaml..), and kernel headers.

The setup commands are:

1. Import HDF into work space:

```
petalinux-config --get-hw-description=<PATH-to-HDF/DSA-DIRECTORY>
```

The above command will only copy hardware design from external location into the `petalinux project <proj-root>/project-spec/hw-description/`. The external-hdf is a recipe in Yocto which imports HDF from this location into Yocto work space. All the HDF dependent recipes uses hardware design from Yocto workspace. By default, this dependency is handled internal to recipes. You have to run the following command for every update in hardware design if you are building without dependencies.

```
petalinux-build -c external-hdf
```

2. Prepare all the prerequisites (native utilities).

   This command has to run only for the first time; re-run is needed only after cleaning

   ```
   petalinux-build -c device-tree -x do_prepare_recipe_sysroot
   ```

   Note: In future release, this feature is deprecated. Using `petalinux-build -c <app/package/component> -x <task>` for building individual task for a component as part of a `petalinux-build` command will be deprecated.

3. Build the device tree ignoring dependency tasks using the following command:

   ```
   petalinux-build -b device-tree
   ```

   This command builds device tree ignoring all dependencies and deploys it in the `images/linux/` directory. If there is any dependency that is not met, it will error out. The above command can be used for incremental builds as well.

Note: The above individual commands need to run with `-b` option. You can get all above functionality in one run: `petalinux-build -c device-tree`. It will remove all dependencies automatically which results in building few more dependent components.

Example 2: Build U-Boot Only

The below example demonstrate building U-Boot ignoring dependencies. `u-boot-xlnx` recipe depends on HDF, device tree, and native tools (mkimage, dtc.)

1. You cannot skip the device tree dependency as it is required. Instead, use the above example to build a device tree.
2. Setup native tools for U-Boot recipe. To do this, use the following command:

   ```
   petalinux-build -c u-boot-xlnx -x do_prepare_recipe_sysroot
   ```

   The above command needs to run only for the first time or after every clean.

   Note: In future release, this feature is deprecated. Using `petalinux-build -c <app/package/component> -x <task>` for building individual task for a component as part of a `petalinux-build` command will be deprecated.

3. Build U-Boot ignoring dependency tasks. To do this, use the following command:

   ```
   petalinux-build -b u-boot-xlnx_2019.1
   ```

The above command builds U-Boot and packages in `images/linux`.

`-b` option needs full name/path of recipe; virtual targets will not work.

Table 8: **Paths of Recipes**

Recipe	Path
kernel, virtual/kernel	linux-xlnx_2019.1
u-boot, virtual/bootloader	u-boot-xlnx-2019.1
device-tree	device-tree

Use the following command to find the path of a recipe:

```
petalinux-build -c "-e virtual/kernel" | grep "^FILE="
```

Replace virtual/kernel with any virtual target or recipe name.

***Note*:** `petalinux-build -b` needs all prerequisites explicitly done by user. `petalinux-build -c` takes care of all dependencies automatically; explicit running of individual commands is not needed.

Commands to be Deprecated in Future Releases

- `petalinux-build -c rootfs`
- `petalinux-build -c <package_group>`
- `petalinux-build -x distclean(for image)`
- `petalinux-build -c component -x <task>`, where task is fetch, unpack, compile, etc.

XILINX

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®-7000 devices, you have generated boot image. For more information, see Generate Boot Image for Zynq UltraScale+ MPSoC.
- For MicroBlaze™ CPUs, you have generated 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 and `--jtag` on a hardware board to boot the image under software emulation (QEMU). 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.

- 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 via JTAG. The `--prebuilt 3` boots the Linux kernel with PMUFW running in the background.

   - 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.

The example of the kernel boot log messages displayed on the console during successful `petalinux-kernel`, is shown below:

```
[    10.709243] Freeing unused kernel memory: 5568K (ffffffc000c20000 -
ffffffc001190000)
[    13.448003] udevd[1666]: starting version 3.2
[    13.458788] random: udevd: uninitialized urandom read (16 bytes read)
[    13.556064] udevd[1667]: starting eudev-3.2
[    14.045406] random: udevd: uninitialized urandom read (16 bytes read)
[    37.446360] random: dd: uninitialized urandom read (512 bytes read)
[    40.406936] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
[    41.460975] macb ff0e0000.ethernet eth0: link up (100/Full)
[    41.474152] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[    44.787172] random: dropbearkey: uninitialized urandom read (32 bytes
read)

PetaLinux 2019.1 xilinx-zcu102-2019_1 /dev/ttyPS0

xilinx-zcu102-2019_1 login: root
Password:
root@xilinx-zcu102-2019_1:~#
root@xilinx-zcu102-2019_1:~#
```

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

TIP: *To exit QEMU, press **Ctrl+A** together and then press **X**.*

Additional Options for Booting on QEMU

- To download 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 will then boot the loaded U-Boot image. Build the system image using `petalinux-build`.
  - For MicroBlaze™ CPUs and Zynq-7000 devices, it will boot `<plnx-proj-root>/images/linux/u-boot.elf` with QEMU.

- To download 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 will then boot the loaded kernel image, with PMU firmware running in the background.

Note: For Zynq UltraScale+ MPSoC kernel boot, you need to 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 <project directory>
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 start up, you will see the normal Linux boot process ending with a login prompt as shown below:

```
[ 10.709243] Freeing unused kernel memory: 5568K (ffffffc000c20000 -
ffffffc001190000)
[ 13.448003] udevd[1666]: starting version 3.2
[ 13.458788] random: udevd: uninitialized urandom read (16 bytes read)
[ 13.556064] udevd[1667]: starting eudev-3.2
[ 14.045406] random: udevd: uninitialized urandom read (16 bytes read)
[ 37.446360] random: dd: uninitialized urandom read (512 bytes read)
[ 40.406936] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
[ 41.460975] macb ff0e0000.ethernet eth0: link up (100/Full)
[ 41.474152] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[ 44.787172] random: dropbearkey: uninitialized urandom read (32 bytes read)
PetaLinux 2019.1 xilinx-zcu102-2019_1 /dev/ttyPS0
xilinx-zcu102-2019_1 login: root
Password:
root@xilinx-zcu102-2019_1:~#
root@xilinx-zcu102-2019_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 2.6. The old options are deprecated in new version but remain functionally operational. PetaLinux tools still use old options, and therefore it gets warning messages. You can ignore them.

Warning message for Zynq UltraScale+ MPSoC:

```
qemu-system-aarch64: -tftp /home/user/xilinx-zcu102-2019.1/images/linux:
The -tftp option is deprecated. Please use '-netdev user,tftp=...' instead.g
```

Boot a PetaLinux Image on Hardware with SD Card

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

This section is for Zynq® UltraScale+™ MPSoC and Zynq-7000 devices only because they allow you to boot from SD card.

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`
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. Watch the serial console, you will see the boot messages similar to the following:

```
[     5.546354] clk: Not disabling unused clocks
[     5.550616] ALSA device list:
[     5.553528]   #0: DisplayPort monitor
[     5.576326] sd 1:0:0:0: [sda] 312581808 512-byte logical blocks: (160
GB/149 GiB)
[     5.583894] sd 1:0:0:0: [sda] Write Protect is off
[     5.588699] sd 1:0:0:0: [sda] Write cache: enabled, read cache:
enabled, doesn't
support DPO or FUA
[     5.630942]  sda:
[     5.633210] sd 1:0:0:0: [sda] Attached SCSI disk
[     5.637897] Freeing unused kernel memory: 512K (ffffffc000c20000 -
ffffffc000ca0000)
INIT: version 2.88 booting
Starting udev
[     5.746538] udevd[1772]: starting version 3.2
[     5.754868] udevd[1773]: starting eudev-3.2
Populating dev cache
Starting internet superserver: inetd.
Running postinst /etc/rpm-postinsts/100-sysvinit-inittab...
Running postinst /etc/rpm-postinsts/libglib-2.0-0...
update-rc.d: /etc/init.d/run-postinsts exists during rc.d purge
(continuing)
INIT: Entering runlevel: 5
Configuring network interfaces... [     6.607236] IPv6:
ADDRCONF(NETDEV_UP): eth0:
link is not ready
udhcpc (v1.24.1) started
Sending discover...
```

```
[    7.628323] macb ff0e0000.ethernet eth0: link up (1000/Full)
[    7.633980] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
Sending discover...
Sending select for 10.10.70.1...
Lease of 10.10.70.1 obtained, lease time 600
/etc/udhcpc.d/50default: Adding DNS 172.19.128.1
/etc/udhcpc.d/50default: Adding DNS 172.19.129.1
Done.
Starting Dropbear SSH server: Generating key, this may take a while...
Public key portion is:
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCxGtijKDWcJgnDxRCGiUPJJIMapFc0tcsCkMGyjJEDs
9LRugWzgaa
8XA+pGy4aTvZqHvGnFTvkMw4gZE/O
+BBgO8mMK9dFei2BvENbljm8M4NotG5LXRCFDaw6bXBCtg4ekCKWNU
6lUQU+PPdpmj9X+JgnTHnHnNB3jP6MrymCuS5wfFbyHfKdrwWXwfLmCycZr7DjRumee7T/
3SrBU3oRJoLcC
Vj2lf5Z7673+rOT1GM3QFzO2HWCCzyz/
3IUcEh9mhKpjzgs4iNEKmxwyi29rl37x7PD7zRsQbaW8uUtheCa
in3M1mjKfPnnygopdVh6IFsAT3FFMK4PYJ1GPL+h root@xilinx-zcu102-zu9-es2-
rev1_0-2019.1
Fingerprint: md5 f2:ce:1d:f2:50:e6:e2:55:5a:96:6f:bc:98:8f:82:99
dropbear.
Starting syslogd/klogd: done
Starting domain watchdog daemon: xenwatchdogd startup
PetaLinux 2019.1 xilinx-zcu102-zu9-es2-rev1_0-2019.1 /dev/ttyPS0
xilinx-zcu102-zu9-es2-rev1_0-2019.1 login: root
Password:
root@xilinx-zcu102-zu9-es2-rev1_0-2019:~#
```

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 9: **PetaLinux Image on Hardware Troubleshooting**

Problem / Error Message	Description and Solution

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

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

Boot a PetaLinux Image on Hardware with JTAG

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

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).
- This is optional and only needed if you wish to make use of prebuilt capability for JTAG boot. You 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. For Zynq-7000 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 3
   ```

   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:

   ```
   INIT: Entering runlevel: 5
   Configuring network interfaces... [    6.607236] IPv6:
   ADDRCONF(NETDEV_UP): eth0:
   link is not ready
   udhcpc (v1.24.1) started
   Sending discover...
   [    7.628323] macb ff0e0000.ethernet eth0: link up (1000/Full)
   [    7.633980] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
   Sending discover...
   Sending select for 10.10.70.1...
   Lease of 10.10.70.1 obtained, lease time 600
   /etc/udhcpc.d/50default: Adding DNS 172.19.128.1
   /etc/udhcpc.d/50default: Adding DNS 172.19.129.1
   Done.
   Starting Dropbear SSH server: Generating key, this may take a while...
   Public key portion is:
   ssh-rsa
   AAAAB3NzaC1yc2EAAAADAQABAAABAQCxGtijKDWcJgnDxRCGiUPJJIMapFc0tcsCkMGyjJEDs
   9LRugWzgaa
   ```

```
8XA+pGy4aTvZqHvGnFTvkMw4gZE/O
+BBgO8mMK9dFei2BvENbljm8M4NotG5LXRCFDaw6bXBCtg4ekCKWNU
6lUQU+PPdpmj9X+JgnTHnHnNB3jP6MrymCuS5wfFbyHfKdrwWXwfLmCycZr7DjRumee7T/
3SrBU3oRJoLcC
Vj2lf5Z7673+rOT1GM3QFzO2HWCCzyz/
3IUcEh9mhKpjzgs4iNEKmxwyi29rl37x7PD7zRsQbaW8uUtheCa
in3M1mjKfPnnygopdVh6IFsAT3FFMK4PYJ1GPL+h root@xilinx-zcu102-zu9-es2-
rev1_0-2019.1

Fingerprint: md5 f2:ce:1d:f2:50:e6:e2:55:5a:96:6f:bc:98:8f:82:99
dropbear.
Starting syslogd/klogd: done
Starting domain watchdog daemon: xenwatchdogd startup
PetaLinux 2019.1 xilinx-zcu102-zu9-es2-rev1_0-2019.1 /dev/ttyPS0
xilinx-zcu102-zu9-es2-rev1_0-2019.1 login: root
Password:
root@xilinx-zcu102-zu9-es2-rev1_0-2019:~
```

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 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>
```

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

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

- To download newly built kernel to target board:

```
$ petalinux-boot --jtag --kernel
```

  - For MicroBlaze™ processors, this will boot `<plnx-proj-root>/images/linux/image.elf` on target board.
  - For Zynq® UltraScale+™ MPSoC, this will boot `<plnx-proj-root>/images/linux/Image` on target board.
  - For Zynq-7000 devices, this will boot `<plnx-proj-root>/images/linux/zImage` 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 10: **PetaLinux Image on Hardware with JTAG Troubleshooting**

Problem / Error Message	Description and Solution
`ERROR: This tool requires 'xsdb' and it is missing. Please source Xilinx Tools settings first.`	***Problem Description:*** This error message indicates that PetaLinux tools can not find the xsdb tool that is a part of the Xilinx Vivado or SDK tools. ***Solution:*** You have to setup Vivado Tools Working Environment. For more information, see PetaLinux Working Environment Setup.
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
```

The `--jtag` option tells `petalinux-boot` to boot on hardware via JTAG, and the `--prebuilt 2` option downloads the prebuilt bitstream (and FSBL for 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:

```
U-Boot 2019.01-07106-gec1e403dd6 (Apr 29 2019 - 09:12:44 +0000)

Board: Xilinx ZynqMP

I2C:   ready
DRAM:  4 GiB
EL Level: EL2
Chip ID: xczuunkn
MMC:   Card did not respond to voltage select!
sdhci@ff170000 - probe failed: -95
Card did not respond to voltage select!

zynqmp_qspi_ofdata_to_platdata: CLK 104156250
SF: Detected n25q512a with page size 512 Bytes, erase size 128 KiB,
total 128 MiB
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Bootmode: JTAG_MODE
Net:   ZYNQ GEM: ff0e0000, phyaddr c, interface rgmii-id

Warning: ethernet@ff0e0000 MAC addresses don't match:
Address in SROM is         ff:ff:ff:ff:ff:ff
Address in environment is  00:0a:35:00:22:01
eth0: ethernet@ff0e0000
U-BOOT for xilinx-zcu102-2019_1

BOOTP broadcast 1
DHCP client bound to address 10.0.2.15 (2 ms)
Hit any key to stop autoboot:  0
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 commands:

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

12. Boot the kernel using the following command:

```
ZynqMP> run netboot
```

Troubleshooting

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

Problem / Error Message	Description and Solution
`Error: "serverip" not defined.`	***Problem Description:*** This error message indicates that `U-Boot` environment variable `serverip` is not set. You have to set it to IP Address of the host where the image resides. ***Solution:*** Use the following command to set the `serverip`: `ZynqMP> 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 for submission to WTS for debug 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 will not modify the specified PetaLinux project `<plnx-proj-root>`. It will put the specified hardware project source to `<plnx-proj-root>/hardware/` inside `MY.BSP` archive.

2. BSP packaging with external sources.

The support for search path is obsolete. It is your responsibility to copy the external sources under components/ext_sources. For more information, see Using External Kernel and U-Boot with PetaLinux. The BSP has to be packaged.

Upgrading the Workspace

PetaLinux tool has system software components (embedded SW, ATF, Linux, U-Boot, OpenAMP, and Yocto framework) and host tool components (Vivado® Design Suite, Xilinx® Software Development Kit (SDK), HSI, and more). To upgrade to the latest system software components, you must install the corresponding host tools (Vivado). For example, if you have the 4.18 kernel that ships with the 2019.1 release but you want to upgrade to the 4.19 kernel that will ship with the 2019.2 release, you must install the 2019.2 PetaLinux tool and the 2019.2 Vivado hardware project.

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 individual PetaLinux projects. This allows you to upgrade without having to install the latest version of the Vivado hardware project or XSDK.

***Note*:** `petalinux-upgrade` is a new command introduced in 2019.1.

IMPORTANT! *This upgrade command will work for minor upgrades only. This means that while you are able to upgrade from 2019.1 to 2019.2 using `petalinux-upgrade`, you cannot upgrade from 2019.1 to 2020.1 using this command.*

petalinux-upgrade Options

Table 12: **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. Directory structure should be: • `<file/esdks>` • `<file/downloads>`	None
`-u --url`	URL to target system software components.	User-specified. URL should be: • `<url/esdks>` • `<url/downloads>`	None
`-w, --wget-args`	Passes additional `wget` arguments to the command.	Additional `wget` options	None

Upgrade PetaLinux Tool

Upgrade from Local File

Download the target system software components content from the server URL http://petalinux.xilinx.com/sswreleases/rel-v2019/.

`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 esdk path>`.

Example:

```
petalinux-upgrade -f "/scratch/ws/upgrade-workspace/eSDK"
```

Note: This option is for offline upgrade.

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-v2019/
sdkupdate/"
```

IMPORTANT! *The current release supports minor version upgrades only.*

Upgrade PetaLinux Project

Upgrade an Existing Project with the Upgraded Tool

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

1. Upgrade the tool. To upgrade from local file, see Upgrade from Local File. To upgrade from remote server, see Upgrade from Remote Server.
2. Go to the PetaLinux project you want to upgrade.
3. Enter command: `petalinux-build -x mrproper`.
4. Enter command: `petalinux-build` to upgrade the project with all new system components.

Create a New Project with the Upgraded Tool

Use the following steps to create a new project with the upgraded tool.

CAUTION! *It is recommended that you use the latest Vivado® Design Suite and PetaLinux tool for creating a new project. Use the following option only if you require the latest ssw components but an earlier version of the Vivado hardware project.*

1. Upgrade the tool. To upgrade from local file, see Upgrade from Local File. To upgrade from remote server, see Upgrade from Remote Server.
2. Create a PetaLinux project.
3. Use `petalinux-build` command to build a project with all new system components.

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/SD card** as per the requirement.

   Note: SD boot functionality expects the RootFS 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 - FSBL, PMU firmware, ATF, U-Boot)

     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)

     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**.
   - jffs2 RootFS image settings
   - DTB settings

Troubleshooting

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

Table 13: **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.

Note: PetaLinux tools use 'boot', 'bootenv', 'kernel.' 'boot' stores 'BOOT.BIN'. 'bootenv' stores u-boot env vars. 'kernel' stores 'image.ub'.

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.

FIT image is the default bootable image format. 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 RootFS 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 RootFS packages. If your requirement does not need some of these packages, you can shrink the size of RootFS 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 config items and file system packages lead to increase in the kernel image size and RootFS size respectively.

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

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

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

   b. Undef `CONFIG_SYS_BOOTMAPSZ` in `<plnx_proj>/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 a New PetaLinux Project) 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 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>; saveenv
        ```

      - 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>
        ```

IMPORTANT! *The default RootFS for PetaLinux is INITRAMFS. In INITRD mode, RootFSs is not included in kernel images.*

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 a New PetaLinux Project) 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 SD Card.
   c. Boot a PetaLinux Image on Hardware with JTAG, see Boot a PetaLinux Image on Hardware with JTAG.

IMPORTANT! *The default RootFS for PetaLinux is INITRAMFS.*

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)
- image.elf → 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 a New PetaLinux Project) 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 a New PetaLinux Project) 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 will be automatically updated. You can check `<plnx-proj-root>/components/plnx_workspace/device-tree/device-tr ee/plnx_aarch64-system.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 will be 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 will be downloaded through JTAG and kernel will be 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`) and kernel image (`image.ub`) will be copied to SD card and will be downloaded from SD card. For more information, see Boot a PetaLinux Image on Hardware with SD Card.

Configuring JFFS2 Boot

Journalling 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 a New PetaLinux Project) and imported the hardware platform (see Importing Hardware Configuration).

Steps to Configure JFFS2 Boot

1. Set the RootFS 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 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>; saveenv
```

   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 RootFS inside `system.dts`.

7. Enable QSPI flash boot mode on board.
8. Reset the board (booting will start 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 a New PetaLinux Project) 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:

1. The SD card is formatted with two partitions using a partition editor such as gparted.
2. The first partition should be at least 60 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 will contain 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 will store the system root file system. Label this partition as RootFS.

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 filesystem 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 filesystem type**.
4. Select **SD card** as the RootFS type.
5. Exit menuconfig and save configuration settings.

   Note: The boot arguments will be automatically updated in the `<plnx-proj-root>/components/plnx_workspace/device-tree/device-tree/syst em-conf.dtsi`. These changes will be 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 will be 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 will have device tree and kernel image files.

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

3. Copy `rootfs.tar.gz` file to RootFS partition of 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 SD Card.

Troubleshooting

Table 14: **Configuring SD Card ext Filesystem Boot**

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

Customizing the Rootfs

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/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 Applications

If your prebuilt application 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
```

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. 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 = ""
```

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

   Note: In future releases, using `petalinux-build -c rootfs` to build RootFS is deprecated. Use `petalinux-build` instead.

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 Xilinx® SDK), 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 main(int argc, char **argv)
   {
     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`. The file should look like the following.

```
APP = sample
  LIBSOURCES=*.c
  OUTS = *.o
  NAME := sample
  MAJOR = 1.0
  MINOR = 1
  VERSION = $(MAJOR).$(MINOR)

  all: lib$(NAME).so

  lib$(NAME).so.$(VERSION): $(OUTS)
        $(CC) $(LDFLAGS) $(OUTS) -shared -Wl,-soname,lib$(NAME).so.$
(MAJOR) -o lib
$(NAME).so.$(VERSION)

lib$(NAME).so: lib$(NAME).so.$(VERSION)
        rm -f lib$(NAME).so.$(MAJOR) lib$(NAME).so
        ln -s lib$(NAME).so.$(VERSION) lib$(NAME).so.$(MAJOR)
        ln -s lib$(NAME).so.$(MAJOR) lib$(NAME).so

%.o: %.c
        $(CC) $(CFLAGS) -c -fPIC $(LIBSOURCES)

clean:
        rm -rf *.o *.so *.so.*
```

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 `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 will see the following PetaLinux template-generated files:

Table 15: **Adding Custom Applications Files**

Template	Description
`<plnx-proj-root>/project- spec/meta-user/ recipes-co re/images/petalinux-image - full.bbappend`	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/`.

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. Apart from deleting the app directory, you have to remove the line: `IMAGE_INSTALL_append= " myapp"` from `<plnx-proj-root>/project-spec/meta-user/recipes-core/images/petalinux-image-full.bbappend`. Deleting the directory by keeping the mentioned line will throw an error.*

Creating and Adding Custom 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 will see the following PetaLinux template-generated files:

   Table 16: **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/recipes-co re/images/petalinux-image -full.bbappened`	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/`.

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:* `IMAGE_INSTALL_append= "mymodule"` *from* `<plnx-proj-root>/project-spec/meta-user/recipes-core/images/petalinux-image.bbappend`. *Deleting the directory by keeping the mentioned line in* `petalinux-image-full.bbappend` *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>` will rebuild 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
```

Note: In future releases, using `petalinux-build -c rootfs` to build RootFS is deprecated. Use `petalinux-build` instead.

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 will just compile the application, the compiled executable files will be in `${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 will remove 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
```

Note: In future releases, `petalinux-build -c <app/package/component> -x <task>` is deprecated. Individual tasks for a component as part of a `petalinux-build` command is deprecated.

This will install the application into the target RootFS host copy: `<TMPDIR>/work/<MACHINE_NAME>-xilinx-linux/petalinux-user-image/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 user application through its Makefile, the user application will be put in to `/usr/bin` directory.

3. Run your user application on the target system console. For example, to run 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 Modules).

Steps to Build User Modules

Running `petalinux-build` in the project directory "`<plnx-proj-root>`" will rebuild 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
```

Note: In future releases, using `petalinux-build -c rootfs` to build RootFS is deprecated. Use `petalinux-build` instead.

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 will just compile the module, the compiled executable files will be 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
```

  Note: In future releases, `petalinux-build -c <app/package/component> -x <task>` is deprecated. Individual tasks for a component as part of a `petalinux-build` command is deprecated.

  This will install the module into the target RootFS host copy: `<TMPDIR>/work/<MACHINE_NAME>-xilinx-linux/petalinux-user-image/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`.

3. Select **Yocto-settings → Enable debug-tweaks**.
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 will never exit, 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
start ()
{
        echo " Starting myapp"
        start-stop-daemon -S -o --background -x $DAEMON
}
stop ()
{
        echo " Stoping myapp"
        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 THUD version can be found here.

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 command:

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

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

***Note*:** 2019.1 PetaLinux is on THUD base line. The layers/recipes should be chosen from the THUD 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.

IMPORTANT! *If you want change the layer priority, you can update `${proot}/project-spec/meta-mylayer/conf/local.conf` in this file BBFILE_PRIORITY_meta-mylayer = 6 (0 to 10, higher value will have more priority).*

Adding an Existing Recipe into RootFS

Most of the RootFS 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 `/opt/pkg/petalinux/components/yocto/source/aarch64/` (for Zynq® UltraScale+™ MPSoC).

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

1. The location of the recipe is `/opt/pkg/petalinux/components/yocto/source/aarch64/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/recipes-core/images/petalinux-image-full.bbappend`.

   ```
   IMAGE_INSTALL_append = " iperf3"
   ```

IMPORTANT! *Whenever "`_append`" is used, there should be a space after `=` ".*

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 user 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 will get the following warning and your changes for the package will not be 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 that will be used to build the images. Some of the package group recipes are shipped with the PetaLinux tools.

For example:

```
$PETALINUX/components/yocto/source/aarch64/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.

We 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 `IMAGE_INSTALL_append = " packagegroup-petalinux-alsa"` in `<plnx-proj-root>/project-spec/meta-user/recipes-core/petalinux-image.bbappend` to reflect in menuconfig.

Then launch `petalinux-config -c rootfs`, select **user packages → packagegroup-petalinux-alsa**, save and exit. Then run `petalinux-build`.

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.

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 (Linaro GDB 2019.1) 7.12.1.20170130-git
   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-unknown-linux-gnu
   --target=aarch64-linux-gnu".
   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"...
Reading symbols from vmlinux...done.
```

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` will attempt 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 17: **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 will show the full QEMU commands, you should be able to see which port is 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 with the XSDK tool. For more information, see Xilinx Software Development Kit.
- 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.

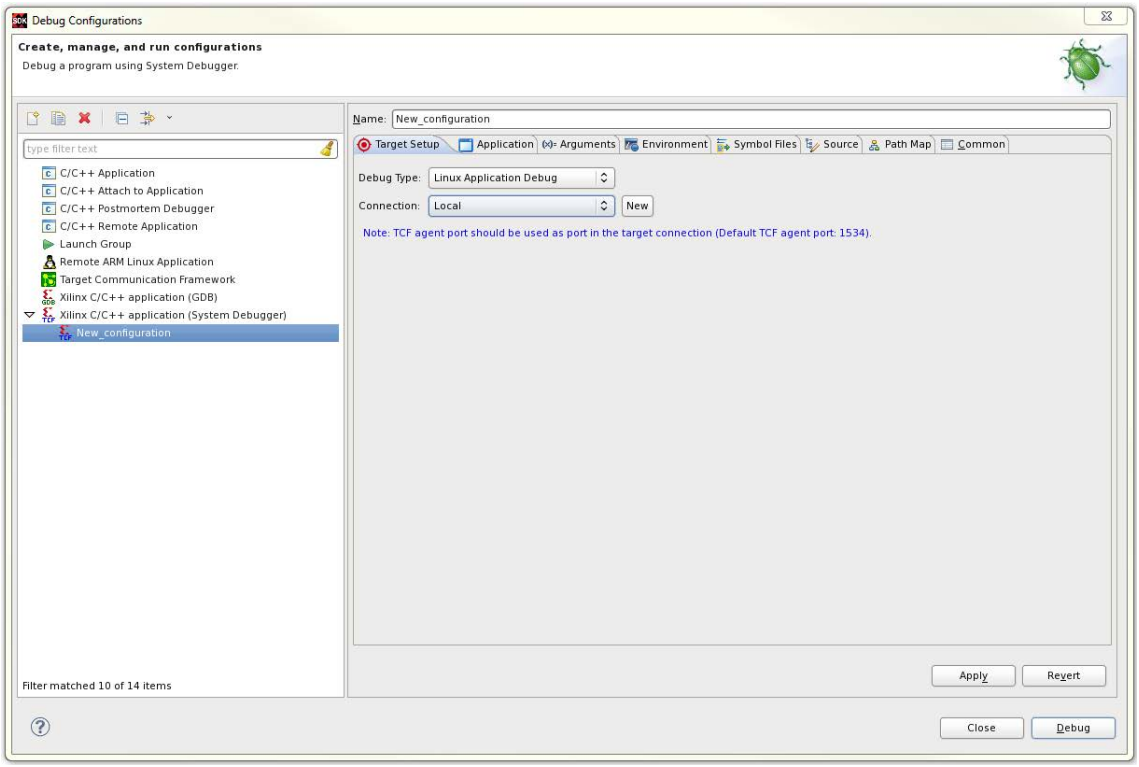
```
[*] dropbear
```

8. Select **console/network → submenu → 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.

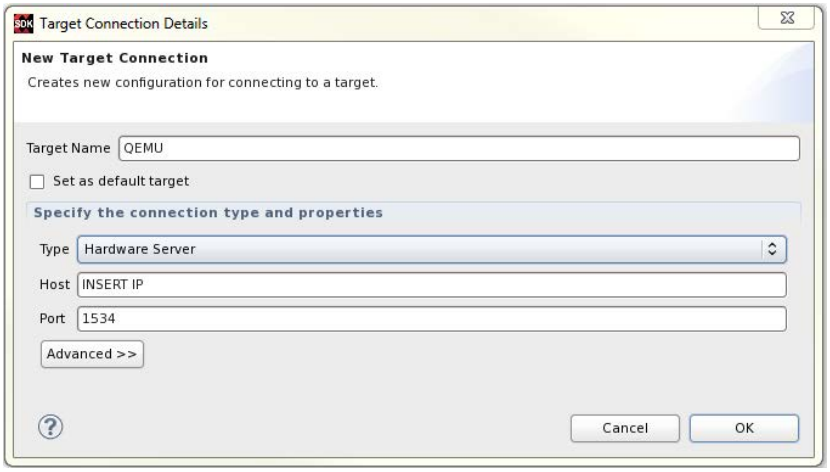
Performing a Debug Session

1. Boot your board (or QEMU) with the new image.
2. The boot log should indicate that tcf-agent has started. The following message should be seen: Starting tcf-agent: OK

3. Launch Xilinx® SDK and create a workspace.
4. Add a Hardware Platform Specification by selecting **File → New → Project**.
5. In the pop-up window, select **Xilinx → Hardware Platform Specification**.
6. Give the Hardware Project a name. For example, ZC702.
7. Locate the `system.hdf` / `system.dsa` for your target hardware. This can be found in `<plnx-proj-root>/project-spec/hw-description`.
8. Open the Debug Launch Configuration window by selecting **Run → Debug Configurations**.
9. Create a new **Xilinx C/C++ application (System Debugger)** and launch configuration:



10. The **Debug Type** should be set to **Linux Application Debug**.
11. Select the New option to enter the Connection details.

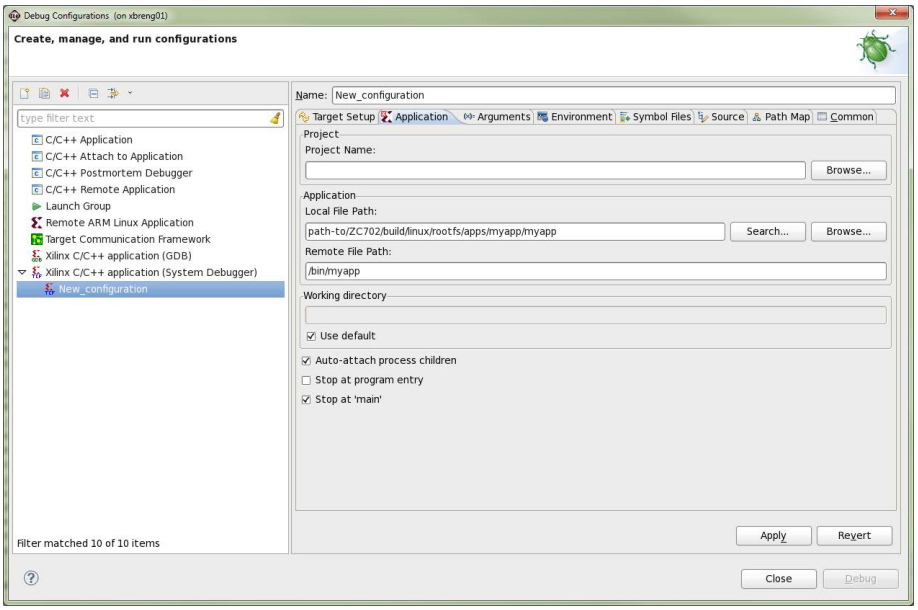


12. Give the Target Connection a name, and specify the Host (IP address for the target).
13. Set the port of tcf-agent and select **OK**.

IMPORTANT! *If debugging on QEMU, see* Appendix D: 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 will need to use localhost as the target IP in the subsequent steps.*

---

14. Switch to the Application Tab.

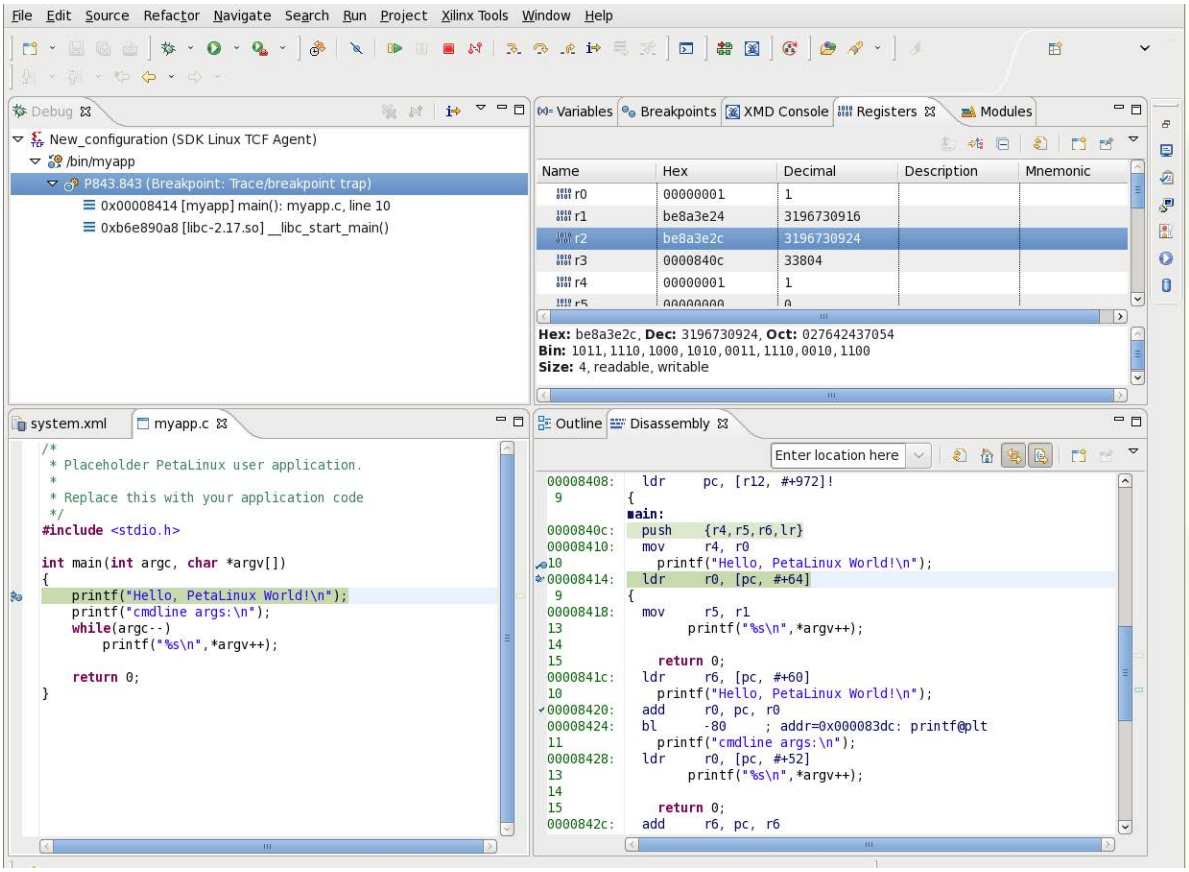


15. Enter the Local File Path to your compiled application in the project directory. For example, `<TMPDIR>/work/aarch64-xilinx-linux/myapp1/1.0-r0/image/usr/bin/`.

    Note: While creating the application, you need to add `RM_WORK_EXCLUDE += "myapp"` in `project-spec/meta-user/conf/petalinuxbsp.conf`, otherwise the images will not be available for debugging.

16. The Remote File Path on the target file system should be the location where the application can be found. For example, `/usr/bin/myapp`.

17. Select **Debug** to Apply the configuration and begin the Debug session. (If asked to switch to Debug Perspective, accept).
18. Standard XSDK debug flow is ready to start:



TIP: *To analyze the code and debug you can use the following short keys:*

- Step Into (F5)
- Step Over (F6)
- Step Return (F7)
- Resume (F8)

Debugging Zynq UltraScale+ MPSoC Applications with GDB

PetaLinux supports debugging Zynq® UltraScale+™ MPSoC user applications with GDB. This section describes the basic debugging procedure.

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/recipes-core/images/peta linux-image.bbappend`:

```
IMAGE_INSTALL_append = " myapp-dev"
IMAGE_INSTALL_append = " 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
   ```
5. The GDB console will start:

   ```
   ...
   GNU gdb (crosstool-NG 1.18.0) 7.6.0.20130721-cvs
   ...
   (gdb)
   ```
6. In the GDB console, connect to the target machine using the command:
   - Use the IP address of the target system, for example: 192.168.0.10. If you are not sure about the IP address, run `ifconfig` on the target console to check.
   - Use the port 1534. If you select a different GDB server port number in the earlier step, use that value instead.

IMPORTANT! *If debugging on QEMU, refer to the QEMU Virtual Networking Modes for information regarding IP and port redirection when testing in non-root (default) or root mode. For example, if testing in non-root mode, you will need to use localhost as the target IP in the subsequent steps.*

```
(gdb) target remote 192.168.0.10:1534
```

The GDB console will attach to the remote target. The GDB server on the target console will display the following confirmation, where the host IP is displayed:

```
Remote Debugging from host 192.168.0.9
```

7. Before starting the execution of the program, create some breakpoints. Using the GDB console you can create breakpoints throughout your code using function names and line numbers. For example, create a breakpoint for the `main` function:

```
(gdb) break main
Breakpoint 1 at 0x10000444: file myapp.c, line 10.
```

8. Run the program by executing the continue command in the GDB console. GDB will begin the execution of the program.

```
(gdb) continue
Continuing.
Breakpoint 1, main (argc=1, argv=0xbffffe64) at myapp.c:10
10  printf("Hello, PetaLinux World!\n");
```

9. To print a list of the code at current program location, use the `list` command.

```
(gdb) list
5 */
6 #include <stdio.h>
7
8 int main(int argc, char *argv[])
9 {
10 printf("Hello, PetaLinux World!\n");
11 printf("cmdline args:\n");
12 while(argc--)
13 printf("%s\n",*argv++);
14
```

10. Try the `step`, `next` and `continue` commands. Breakpoints can be set and removed using the `break` command. More information on the commands can be obtained using the GDB console `help` command.

11. When the program finishes, the GDB server application on the target system will exit. 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 will be 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 18: **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

XILINX

Chapter 10

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

In this release, the Linux system components available in the sub-menu are shown as follows:

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

For ATF, U-Boot, and kernel there are 3 options available:

1. Default

   The default component is shipped through PetaLinux tool.

2. External source

   When you have a component downloaded at any specified location, you can feed your component instead of the default one through this config 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.

3. Remote

   If you want to build a component which was on a custom git repo, this config option has to be used.

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.

Table 19: **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 UltraScale+ MPSoC 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/meta-plnx-generated/recipes-kernel/linux/configs/`
	`plnx_kernel.cfg`
	`bsp.cfg`

U-Boot	The following files are in `<plnx-proj-root>/project-spec/meta-plnx-generated/recipes-bsp/u-boot/configs/`
	`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.

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.

Table 20: **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.

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/ram 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/ram rw`. Example: `earlycon console=/dev/ttyPS0,115200 clk_ignore_unused root=/dev/ram rw` in `system_user.dtsi`.

For more information, see kernel documentation.

ATF Compilation Configuration

The ATF Compilation Configuration appears only for the Zynq UltraScale+ MPSoC platform. This sub-menu allows you to set:

- Extra ATF compilation settings
- Change the base address of bl31 binary
- Change the size of bl31 binary

Power Management Kernel Configuration

The Power Management Kernel Configuration option allows PetaLinux to add power related kernel configs.

Select this to enable/disable power management related kernel configs through `plnx-kernel.cfg`. These configs are later applied over kernel defconfig (xilinx_zynqmp_defconfig). If this is not selected, the kernel configs will not be enabled/disabled by PetaLinux explicitly.

The default configuration from Linux defconfig is retained in your PetaLinux project.

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.

Image Packaging Configuration

The Image Packaging Configuration sub-menu allows you to set the following image packaging configurations:

- Root file system type
- 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.*

Firmware Version Configuration

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

Table 21: **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`

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

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 will pack bitstreams and dtbos to the `/lib/firmware` directory in RootFS.

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

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

   Note: PetaLinux FPGA manager configuration when selected:

   1. Will generate `pl.dtsi` nodes as a dt overlay (dtbo).
   2. Will pack bitstreams in .bin form and dtbos to the `/lib/firmware/base` directory in RootFS.
   3. `BOOT.BIN` generated using `petalinux-package` command will not have bitstream.
3. Specify extra hw files in FPGA Manager ---> () Specify hw 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 RootFS. It will generate and pack bitstream in .bin form and its dtbo in the RootFS at `/lib/firmware/<DSA/HDF name>`. Ensure that PS design is same for DSA/HDF at hw directory path and `<PROOT>/project-spec/hw-description/system<.hdf/.dsa>`.

4. Run `petalinux-build`.

Example loading full bitstream on target:

```
root@xilinx-zcu102-2019_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-zcu102-2019_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-zcu102-2019_1:~# fpgautil -h`

For more information, see http://www.wiki.xilinx.com/Solution+ZynqMP+PL+Programming.

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`
4. It will 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`).

Configuring Remove PL 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 will not be any PL DTBO generated.

Yocto Settings

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

Table 22: **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 Debug Tweaks	Login into target without password
Enable Network sstate feeds	Enabled NW sstate feeds

Table 22: **Yocto Settings** *(cont'd)*

Parameter	Description
User layers	Adds user layers into projects
BB_NO_NETWORK	When enabled, internet access is disabled on the build machine

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:

     ```
     s${AUTOREV}
     ```

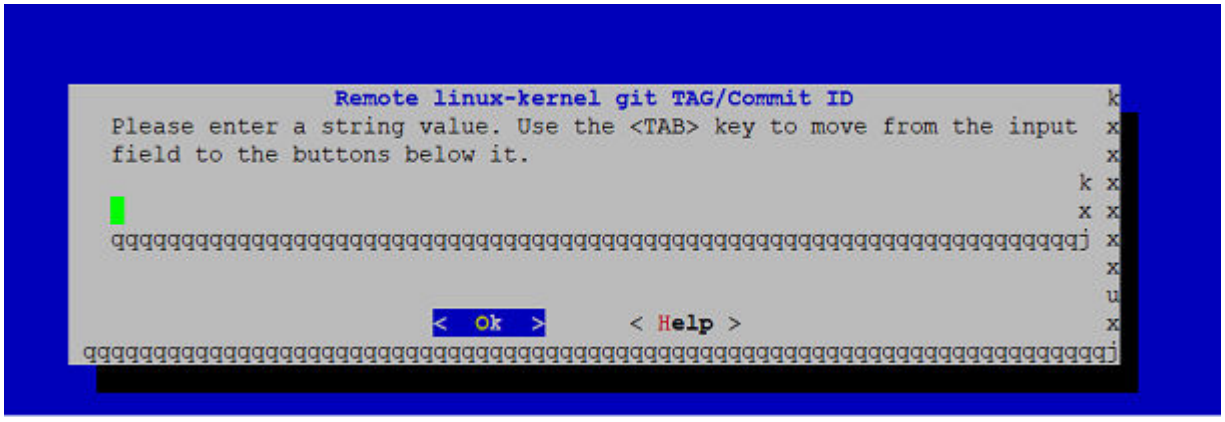
   - To point to any tag:

     ```
     tag/mytag
     ```

   - To point to any commit id:

     ```
     commit id sha key
     ```

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

5. 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:

     `${TOPDIR}/../components/ext_sources/<MY-KERNEL>`
   - For U-Boot, select **External u-boot local source settings --->**. Enter the path:

     `${TOPDIR}/../components/ext_sources/<MY-U-BOOT>`

     ${TOPDIR} is a Yocto variable pointing to `<plnx-proj-root>/build` directory. You can also specify an absolute path of the source. The sources can be placed outside the project as well.

   ***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 repo, 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 23: **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
```

TIP: *Set U-Boot target in `petalinux-config` menuconfig as required, for your custom board. Set `$ petalinux-config Set MACHINE_NAME` as required. Possible values are ac701-full, ac701-lite, kc705-full, kc705-lite, kcu105, zc1254-reva, zc1275-reva, zc1275-revb, zc1751-dc1, zc1751-dc2, zc702, zc706, avnet-ultra96-rev1, zcu100-reva, zcu100-revb, zcu100-revc, zcu102-rev1.0, zcu102-reva, zcu102-revb, zcu104-reva, zcu104-revc, zcu106-reva, zcu111-reva, zedboard, sp701-rev1.0, vcu118-rev2.0, zcu1285-reva*

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_%.bba ppend` for FSBL initializations.

```
YAML_COMPILER_FLAGS_append = " -DXPS_BOARD_ZCU102" #for zcu102
YAML_COMPILER_FLAGS_append = " -DXPS_BOARD_ZCU106" # for zcu106
```

PetaLinux automated the system currently, it does not add these macros.

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/`:

- `multi-arch/`
- `system-user.dtsi`
- `xen.dtsi`
- `zynqmp-qemu-arm.dts`

- `openamp.dtsi`
- `xen-qemu.dtsi`

The generated files will be in the `<plnx-projroot>/components/plnx_workspace/device-tree/device-tree/` directory.

CAUTION! *All the above mentioned dtsi files are generated by the tool. Editing any of these files is not recommended.*

For more details on device tree files, see Appendix B: PetaLinux Project Structure.

CAUTION! *DTSI files listed above `*.dtsi` are automatically generated; you are not supposed to edit these files.*

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.

***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 defines:

- Configuration `_OPTIONS_`:

  You will be able to select the configuration options. They have names beginning with "`CONFIG_`".

- Configuration `_SETTINGS_`:

  These depend on the hardware etc. 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 `<plnxproj_root>/project-spec/ meta-plnx-generated/recipes-bsp/u-boot/configs` and `platform-top.h` located at `<plnxproj_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 will be as follows:*

```
U-Boot-PetaLinux> mtest Unknown command 'mtest' - try 'help'
```

For more information on U-Boot, see https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842223/U-boot.

Yocto Features

SDK Generation (Target Sysroot Generation)

The OpenEmbedded build system uses BitBake to generate the Software Development Kit (SDK) installer script standard SDKs. PetaLinux builds and installs SDK. The installed SDK can be used as sysroot for the application development.

This does not refer to Xilinx® SDK.

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-user-image -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-user-image.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.

Accessing BitBake in a Project

BitBake is available only in the bash shell.

Steps to Get the BitBake Utility for Zynq UltraScale+ MPSoC

1. Run `petalinux-config` or `petalinux-config --oldconfig` 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 /opt/pkg/petalinux/components/yocto/source/aarch64/env ironment-
setup
 -aarch64-xilinx-linux
```

4. Source the environment setup script:

```
source /opt/pkg/petalinux/components/yocto/source/aarch64/layers/core/
 oe-init-build-env
```

   After the above step, you will be redirected to the build directory. Stay in the build directory to run BitBake.

5. Export XSCT:

```
export PATH=/opt/pkg/petalinux/tools/hsm/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 will be 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.

Shared sstate-cache

Yocto e-SDK contains minimal shared sstate cache. Xilinx® hosts the full `petalinux-image` sstate-cache at http://petalinux.xilinx.com/sswreleases/rel-v2019.1/.

During `petalinux-build`, BitBake will search for sstate-cache in the PetaLinux tool, that is the minimal set. If the sstate-cache is not found in this location, BitBake then searches for the same in http://petalinux.xilinx.com/sswreleases/rel-v2019.1/. Yet, if the sstate-cache is not found, BitBake will build from scratch. sstate is signature locked.

For a `.bbappend` file which you create for any RootFS component, you must add `SIGGEN_UNLOCKED_RECIPES += "<component>"` in `<plnx proj root>/project-spec/meta_user/conf/petalinuxbsp.conf`.

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, this URL is added to the Yocto SOURCE Mirror in `petalinux-config`.

If any component is rebuilt from the scratch, BitBake first searches for its source in pre-mirrors, that is, in downloads of the tool, and then searches in petalinux.xilinx.com downloads mirror URL. Later, it searches in SRC_URI of recipes for downloading the source of that component.

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`.

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 into 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 24: **Machine Name Change for Templates**

Template	Machine
zynq	plnx-zynq7
zynqmp	plnx-zynqmp
microblaze	plnx-microblazeel

Table 25: **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	plnx-microblazeel
ac701	plnx-microblazeel
kcu105	plnx-microblazeel
zcu111	zcu111-zynqmp
zcu1285	zcu1285-reva
zc1275	zc1275-revb
sp701	sp701-rev1.0
vcu118	vcu118-rev2.0

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 SoC's 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. The mali440 override is added for EG and EV devices. VCU override is added based on the VCU IP in hardware design.
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" - RFSoC devices

The default value is "eg". PetaLinux automatically assigns "ev" and "dr" based on the presence of IP in the HDF.

***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"
```

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 patching and 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-2019.1/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 `<plnx-proj-root>/project-spec/meta-user/recipes-core/images/petalinux-image.bbappend`.

```
IMAGE_INSTALL_append = " plnx_myapp"
```

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
"/opt/pkg/petalinux/components/yocto/source/aarch64/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, 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:

- You will see the error "`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, we need a UTF-8 when Python starts, otherwise 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`, when the kernel and U-Boot BitBake try to open a new terminal inside, sometimes it fails. The following are the possible error message.

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. Uncommment the OE_TERMINAL line in `<plnx-proj-root>/project-spec/meta-user/conf/petalinuxbsp.conf`. You can set any terminal which suits you. For more details, see Chapter 11: Yocto Features. You have to change the OE_TERMINAL as it is not able to get through default. Uncomment the OE_TERMINAL in `<plnx-proj-root>/project-spec/meta-user/conf/petalinuxbsp.conf` and set it to xterm or screen. For this, you are required to have the corresponding utility installed in your PC.

External Source Configurations

The cfg or scc files will not be applied with external source in the Yocto flow (upstream behavior). PetaLinux needs to handle external source with configurations applied; it has the faculty to handle only cfgs. Therefore, it is always recommended to use cfgs instead of sccs.

Xen and openamp are handled through distro features. Adding distro features will 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"
```

All the scc files have to be taken care by replacing with their respective cfg files, while 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 RootFS type to SD card.

   ```
   $ petalinux-config
   ```

   Select **Image Packaging Configuration → Root filesystem type → SD card**

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 DNF package management system for Zynq-7000 devices. Use the following steps to configure and use the package management system:

1. Enable DNF through `petalinux-config -c rootfs`. Enable **package management** under Image Features and **dnf** under file system packages/base.
2. Build the project.

```
#petalinux-build
```

3. Boot Linux in SD or in JTAG boot mode.
4. Setup a `.repo` file on target on Xilinx platforms by creating `oe-remote-repo-sswreleases-rel-v2019.1-feeds-ultra96-zynqmp.repo` in `/etc/yum.repos.d/` directory.

```
[oe-remote-repo-sswreleases-rel-v2019.1-feeds-ultra96-zynqmp]
name=OE Remote Repo: sswreleases rel-v2019.1 feeds ultra96-zynqmp
baseurl=http://petalinux.xilinx.com/sswreleases/rel-v2019.1/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 Chapter 12: Technical FAQs. Storage based RootFilesystem is
recommended for large images."
```

If your INITRAMFS image size is large, use storage based boot.

XILINX

Appendix A

Migration

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

Tool Directory Structure

Following is the tool directory structure:

- `<path-to-installed-petalinux>/tools/linux-i386/petalinux` has been moved to `<path-to-installed-petalinux>/tools/xsct/petalinux`.
- Since the PetaLinux tool is not using the available toolchain, `<path-to-installed-petalinux>/tools/linux-i386`, it has been removed.
- `<path-to-installed-petalinux>/tools/xsct/SDK/2018.3` has been moved to `<path-to-installed-petalinux>/tools/xsct`.

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.

Build Changes

`petalinux-build` will generate uImage for Zynq®-7000 devices and MicroBlaze™ processors. uImage support for Zynq® UltraScale+™ MPSoC has been removed.

XILINX

Appendix B

PetaLinux Project Structure

This section provides a brief introduction to the file and directory structure of a PetaLinux project. A PetaLinux project supports development of a single Linux system development at a time. A built Linux system is composed of the following components:

- Device tree
- First stage boot loader (optional)
- U-Boot
- Linux kernel
- RootFS 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
   meta-plnx-generated
   hw-description
   configs
   meta-user
pre-built
   linux
      implementation
      images
hardware
   xilinx-zcu104-2019.1
components
   plnx_workspace
      device-tree
config.project
README
README.hw
```

Table 26: **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.
`/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-plnx-generated/recipes-bsp/u-boot/configs`	U-Boot PetaLinux configuration files. The following files are auto generated by `petalinux-config`:
	`config.mk` for MicroBlaze processors only
	`platform-auto.h`
	`config.cfg`
	`platform-top.h` will not be modified by any PetaLinux tools. When U-Boot builds, these files are copied into U-Boot build directory `build/linux/u-boot/src/<U_BOOT_SRC>/` as follows:
	config is the U-Boot kconfig file.
	`config.mk` is copied to `board/xilinx/microblaze-generic/` for MicroBlaze.

`/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.

Table 26: **PetaLinux Project Description** *(cont'd)*

File / Directory in a PetaLinux Project	Description
`/components/`	Directory for embedded software workspace and place to hold external sources while packing BSP. You can also manually copy components into this directory. Here is the rule to place a external component: `<plnx-proj-root>/components/ext_source/<COMPONENT>`
`/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 will be 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`.

Here is an example:

```
<plnx-proj-root>
 -build
 -bitbake.lock
 -build.log
 -config.log
 -cache/
 -conf/
 -downloads/
 -misc/
 -config/
 -plnx-generated/
 -rootfs_config/
 -sstate-cache/
 -tmp/
 -components
 -plnx_workspace/
 -config.project
 -hardware
 -images
 -linux/
 -pre-built
 -linux/
 -project-spec
 -attributes
 -configs/
 -config
 -rootfs_config
 -hw-description/
 -meta-plnx-generated/
 -meta-user/
```

***Note*:** `<plnx-proj-root>/build/` are automatically generated. Do not manually edit files in this directory. Contents in this directory will get updated when you run `petalinux-config` or `petalinux-build`. `<plnx-proj-root>/images/` are also automatically generated. Files in this directory will get 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 remove `INHERIT += "rm_work"` from `build/conf/local.conf`, but it increases the project space.

Table 27: **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
`${TMPDIR}/work/plnx_aarch64-xilinx-linux/petalinux-ser-image/1.0-r0/rootfs`	RootFS copy of target. This is the staging directory.
`${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

Table 28: **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

Project Layers

The PetaLinux project has two following layers under `<proj-plnx-root>/project-spec`

1. meta-plnx-generated

This layer holds all bbappends and configuration fragment (cfg) for all components. All files in this layer are generated by the tool based on HDF/DSA and user configuration. The files in this layer should not be updated manually, as it is regenerated for `petalinux-config` and `petalinux-build` commands.

2. 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.

Appendix C

Generating Boot Components

First Stage Boot Loader

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. 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 will be 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 will be 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 the user wishes not to have PetaLinux build the PMU firmware, then you will need 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. 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 will be 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%5CPMU+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 will need 8 KB block RAM at least.

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 29: **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

FS-Boot ignores other fields in the wrapper header. PetaLinux tools generate the wrapper header to wrap around the U-Boot BIN file.

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 30: **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
`-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

Appendix E

Xilinx IP Models Supported by QEMU

The QEMU emulator shipped in PetaLinux tools supports the following Xilinx® IP models:

- Zynq® UltraScale+™ MPSoC Arm® Cortex™-A53 MPCore
- Zynq UltraScale+ MPSoC Cortex-R5F
- Zynq-7000 Arm Cortex-A9 CPU
- MicroBlaze™ CPU (little-endian AXI)
- Xilinx Zynq-7000/Zynq UltraScale+ MPSoC DDR Memory Controller
- Xilinx Zynq UltraScale+ MPSoC DMA Controller
- Xilinx Zynq UltraScale+ MPSoC SD/SDIO Peripheral Controller
- Xilinx Zynq UltraScale+ MPSoC Gigabit Ethernet Controller
- Xilinx Zynq UltraScale+ MPSoC NAND Controller
- Xilinx Zynq UltraScale+ MPSoC UART Controller
- Xilinx Zynq UltraScale+ MPSoC QSPI Controller
- Xilinx Zynq UltraScale+ MPSoC I2C Controller
- Xilinx Zynq UltraScale+ MPSoC USB Controller (Host support only)
- Xilinx Zynq-7000 Triple Timer Counter
- Xilinx Zynq-7000 DMA Controller
- Xilinx Zynq-7000 SD/SDIO Peripheral Controller
- Xilinx Zynq-7000 Gigabit Ethernet Controller
- Xilinx Zynq-7000 USB Controller (Host support only)
- Xilinx Zynq-7000 UART Controller
- Xilinx Zynq-7000 SPI Controller
- Xilinx Zynq-7000 QSPI Controller
- Xilinx Zynq-7000 I2C Controller
- Xilinx AXI Timer and Interrupt controller peripherals

- Xilinx AXI External Memory Controller connected to parallel flash
- Xilinx AXI DMA Controller
- Xilinx AXI Ethernet
- Xilinx AXI Ethernet Lite
- Xilinx AXI UART 16650 and Lite

***Note*:** By default, QEMU will disable 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, see *Xilinx Quick Emulator User Guide: QEMU* (UG1169).

XILINX

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 ZCU102 reference BSP.
- There are Xen related prebuilts in the `pre-built/linux/images` directory, which are `xen.dtb`, `xen.ub`, `xen-Image` and `xen-rootfs.cpio.gz.u-boot`.

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/images/xen.dtb <tftpboot>/
$ cp pre-built/linux/images/xen.ub <tftpboot>/
$ cp pre-built/linux/images/xen-Image <tftpboot>/
$ cp pre-built/linux/images/xen-rootfs.cpio.gz.u-boot <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 SD Card and for JTAG boot, see Boot a PetaLinux Image on Hardware with JTAG.

3. Setup TFTP server IP from U-Boot:

```
ZynqMP> setenv serverip <TFTP SERVERIP>
```

4. Load Xen images and from U-Boot:

```
ZynqMP> tftpboot 1000000 xen.dtb
ZynqMP> tftpboot 80000 xen-Image
ZynqMP> tftpboot 1030000 xen.ub
ZynqMP> tftpboot 2000000 xen-rootfs.cpio.gz.u-boot
ZynqMP> bootm 1030000 2000000 1000000
```

TIP: *For re-built images that differ in RootFS image size, the above addresses have to be adjusted so that there is no overlap when these images are copied to the RAM.*

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`: `$ petalinux-build`.
7. The build artifacts will be 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 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/xen.ub <tftpboot>/
$ cp  images/linux/Image <tftpboot>/
$ cp  images/linux/rootfs.cpio.gz.u-boot <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 SD Card and for JTAG boot, see Boot a PetaLinux Image on Hardware with JTAG.

3. Setup TFTP server IP from U-Boot:

```
ZynqMP> setenv serverip <TFTP SERVERIP>
```

4. Load Xen images from U-Boot:

```
ZynqMP> tftpboot 1000000 xen.dtb
ZynqMP> tftpboot 80000 xen-Image
ZynqMP> tftpboot 1030000 xen.ub
ZynqMP> tftpboot 2000000 xen-rootfs.cpio.gz.u-boot
ZynqMP> bootm 1030000 2000000 1000000
```

   Note: The load addresses of Xen are just representatives, for exact steps and load addresses see http://www.wiki.xilinx.com/XEN+Hypervisor.

Execute OpenAMP

Use the following steps to execute OpenAMP:

1. Boot kernel:

```
$ 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>
```

2. To load any firmware and run any test application:

```
$ echo <echo_test_firmware> > /sys/class/remoteproc/remoteproc0/firmware
```

   For example, to load image_echo_test:

```
$ echo image_echo_test >  /sys/class/remoteproc/remoteproc0/firmware
$ echo start > /sys/class/remoteproc/remoteproc0/state
$ modprobe rpmsg_user_dev_driver
$ echo_test
```

3. To unload the application:

```
modprobe -r rpmsg_user_dev_driver echo stop > /sys/class/remoteproc/
remoteproc0/state
```

For more information on OpenAMP, see *Libmetal and OpenAMP for Zynq Devices User Guide* (UG1186).

XILINX

Appendix G

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

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

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.

Please Read: Important Legal Notices

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright