

PetaLinux Tools Documentation

Reference Guide

UG1144 (v2023.2) October 18, 2023

AMD Adaptive Computing is creating an environment where employees, customers, and partners feel welcome and included. To that end, we're removing non-inclusive language from our products and related collateral. We've launched an internal initiative to remove language that could exclude people or reinforce historical biases, including terms embedded in our software and IPs. You may still find examples of non-inclusive language in our older products as we work to make these changes and align with evolving industry standards. Follow this [link](#) for more information.



Table of Contents

Chapter 1: Overview.....	7
Introduction.....	7
Navigating Content by Design Process.....	9
Chapter 2: Setting Up Your Environment.....	10
Installation Steps.....	10
PetaLinux Working Environment Setup.....	14
Design Flow Overview.....	15
Chapter 3: Creating a Project.....	17
Creating a Project Using PetaLinux BSP.....	17
Configuring a Hardware Platform with Vivado Design Suite.....	19
Exporting a Hardware Platform to PetaLinux Project.....	20
Creating an Empty Project from a Template.....	21
Chapter 4: Configuring and Building.....	23
Version Control.....	23
Importing Hardware Configuration.....	24
Building a System Image.....	27
Build Optimizations.....	30
Chapter 5: Packaging and Booting.....	32
Packaging Boot Image.....	32
Packaging Prebuilt Images.....	39
BSP Packaging.....	40
Booting PetaLinux Prebuilt Images.....	41
Booting PetaLinux Image on QEMU.....	43
Booting PetaLinux Image on Hardware with an SD Card.....	46
Booting PetaLinux Image on Hardware with JTAG.....	49
Booting PetaLinux Image on Hardware with TFTP.....	52
Booting PetaLinux Image on Hardware with QSPI or OSPI.....	54
Boot from HBM/Higher DDR/LP DDR.....	58

Chapter 6: Upgrading the Workspace.....	61
Upgrading Between Minor Releases (2023.1 Tool with 2023.X Tool)	61
Upgrading the Installed Tool with More Platforms.....	63
Upgrading the Installed Tool with your Customized Platform.....	64
Chapter 7: Customizing the Project.....	65
Configuring Firmware Version.....	65
Configuring Root File System Type.....	66
Configuring U-Boot Boot Script (boot.scr).....	66
Configuring Primary Flash Partition.....	69
Configuring INITRD Boot.....	70
Configuring INITRAMFS Boot.....	71
Configuring TFTP/PXE Boot.....	72
Configuring NFS Boot.....	73
Configuring JFFS2 Boot.....	74
Configuring UBIFS Boot.....	76
Configuring SD Card ext File System Boot.....	80
Managing Image Size.....	82
Chapter 8: Customizing the Root File System.....	84
Including Prebuilt Libraries.....	84
Including Prebuilt Applications.....	84
Creating and Adding Custom Libraries.....	85
Creating Apps in PetaLinux Project.....	86
Creating and Adding Custom Applications.....	86
Creating and Adding Custom Kernel Modules.....	88
Building User Applications.....	89
Testing User Applications.....	91
Building User Modules.....	92
PetaLinux Auto Login.....	93
Application Auto Run at Startup.....	93
Adding Layers.....	94
Adding an Existing Recipe into the Root File System.....	95
Adding a Package Group.....	96
Appending Root File System Packages.....	97
Chapter 9: Debugging.....	98
Debugging the Linux Kernel in QEMU.....	98

Debugging Applications with TCF Agent.....	100
Debugging Zynq UltraScale+ MPSoC and Versal Adaptive SoC Applications with GDB.	114
Debugging Individual PetaLinux Components.....	119
Chapter 10: Advanced Configurations.....	120
Menuconfig Usage.....	120
PetaLinux Menuconfig System.....	120
Open Source Bootgen for On-target Use for Zynq Devices, Versal Adaptive SoC, and Zynq UltraScale+ MPSoC.....	155
Configuring Out-of-tree Build.....	155
Configuring Project Components.....	159
Chapter 11: Yocto Features.....	165
SDK Generation (Target Sysroot Generation).....	165
Accessing BitBake/Devtool in a Project.....	167
Shared State Cache.....	168
Downloading Mirrors.....	169
Machine Support.....	170
SoC Variant Support.....	171
Image Features.....	172
Filtering RootFS Packages Based on License.....	173
Creating and Adding Patches For Software Components within a PetaLinux Project...	173
Adding Extra Users to the PetaLinux System.....	174
Chapter 12: Technical FAQs.....	176
Troubleshooting	176
Appendix A: Migration.....	185
PetaLinux Supported Dynamic Configuration.....	185
Login Changes.....	196
Systemv to Systemd.....	197
Yocto Override Syntax Changes.....	198
Yocto Recipe Name Changes.....	198
U-Boot Image Changes.....	198
MCS File Support.....	199
BIF File Changes.....	199
Switch_root in petalinux-config.....	200
Added Distroboot Support for MicroBlaze Processors.....	201
Use Yocto Environment Variables in petalinux-config Option.....	202

Removed Webtalk From PetaLinux.....	202
U-Boot Configuration Changes.....	202
Changes to petalinux-boot Command.....	203
FPGA Manager Changes.....	204
Host GCC Version Upgrade.....	205
Image Selector.....	206
Board/Board Variant Changes.....	206
Removed Auto Login.....	207
Removed platform-auto.h	207
Changes in Machine Conf.....	207
Removed config.project	208
Usage of uenv.txt.....	208
Renamed ARM Trusted Firmware Configuration.....	208
Appendix B: PetaLinux to Create PL Applications and Install on Target.....	210
Appendix C: PetaLinux Project Structure.....	211
Project Layers.....	214
Appendix D: Generating Boot Components.....	215
Platform Loader and Manager Firmware (PLM).....	215
Processing System Management Firmware (PSM).....	216
Image Selector.....	216
First Stage Boot Loader for Zynq UltraScale+ and Zynq 7000 Devices.....	217
Trusted Firmware-A (TF-A).....	218
PMU Firmware.....	219
FS-Boot for MicroBlaze Platform Only.....	219
Appendix E: QEMU Virtual Networking Modes.....	221
Specifying the QEMU Virtual Subnet.....	221
Appendix F: AMD IP Models Supported by QEMU.....	222
Appendix G: Xen Zynq UltraScale+ MPSoC and Versal Adaptive SoC Example.....	223
Prerequisites.....	223
Boot Prebuilt Linux as dom0.....	223
Rebuild Xen.....	224



Boot Built Linux as dom0.....	225
Appendix H: Booting Prebuilt OpenAMP.....	226
Appendix I: Partitioning and Formatting an SD Card.....	227
Appendix J: Auto-mounting an SD Card.....	229
Appendix K: PetaLinux Commands.....	231
petalinux-create.....	231
petalinux-config.....	235
petalinux-build.....	238
petalinux-boot.....	242
petalinux-package.....	282
petalinux-util.....	294
petalinux-upgrade.....	297
petalinux-devtool.....	297
Appendix L: Additional Resources and Legal Notices.....	303
Finding Additional Documentation.....	303
Support Resources.....	304
Revision History.....	304
References.....	304
Please Read: Important Legal Notices.....	305

Overview

Introduction

PetaLinux is an embedded Linux Software Development Kit (SDK) targeting FPGA-based system-on-a-chip (SoC) designs or FPGA designs.

You are expected to possess fundamental proficiency in Linux, including the ability to execute Linux commands. You should also know the OS and host system functionalities, such as the OS version, Linux distribution, security privileges, and [elementary concepts of Yocto](#).

The PetaLinux tool contains the following:

- Yocto Extensible SDK ([eSDK](#))
- XSCT (Software Command-Line Tool) and toolchains
- PetaLinux Command Line Interface (CLI) tools

Note: AMD Vitis™ unified software platform is the integrated design environment (IDE) for creating embedded applications on AMD microprocessors. For more details, refer to *Vitis Unified Software Platform Documentation: Embedded Software Development* ([UG1400](#)).

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

New Features in 2023.2

Table 1: New Features

Sr No.	New Features	Section
1	NFS root support for MB.	Configuring NFS Boot
2	Automated boot flow when user select any available memory in design.	Boot from HBM/Higher DDR/LP DDR
3	Segmented boot flow template support in PetaLinux tools.	Versal (Segmented Configuration Flow)
4	Removed OS support for Ubuntu 20.04 LTS and 20.04.1 LTS versions. Added OS support for ubuntu 20.04.6 LTS, 22.04.2 LTS version. Added OS support for OPEN SUSE Leap 15.4 and Add OS support for AlmaLinux 9.1 version.	Installation Requirements

Table 1: New Features (cont'd)

Sr No.	New Features	Section
5	Added Config option to specify Inherit conf file used to generate the yocto machine conf file using gen-machine-conf.	Yocto Settings
6	Added Config option to specify additional MACHINEOVERIDES to generate the machine conf file.	Yocto Settings
7	Password recovery mechanism in PetaLinux without rebuilding.	Password Recovery
8	Migrated to use the New FPGA manager template classes from PetaLinux.	Dynamic Configuration Support in PetaLinux
9	New BSPs added in 2023.2.	KD240 starter kit flat BSP. VEK280 PRE-PROD BSP with new eth phy.
10	Dropped BSPs.	Removed zcu1275 and zcu1285 BSPs.
11	Default boot out of HBM for vhk158 prod BSP.	

Yocto Extensible SDK

The following table details the three extensible SDKs installed.

Table 2: Extensible SDKs

Path	Architecture
PETALINUX/components/yocto/source/aarch64	AMD Zynq™ UltraScale+™ MPSoC and AMD Versal™ adaptive SoC.
PETALINUX/components/yocto/source/arm	Zynq 7000 devices.
PETALINUX/components/yocto/source/microblaze	MicroBlaze™ platform.

Note: MicroBlaze lite design is not supported from 2022.1 onwards.

Table 3: PetaLinux Uses the Following Properties to Identify between MicroBlaze Lite Design or MicroBlaze Full Design:

MicroBlaze Lite	MicroBlaze Full
Verify the below properties in <plnx-proj-root>/build/misc/plnx_syshw_data file: XILINX_MICROBLAZE0_USE_DIV: 0 int XILINX_MICROBLAZE0_USE_HW_MUL: 0 int	Verify the below properties in <plnx-proj-root>/build/misc/plnx_syshw_data file: XILINX_MICROBLAZE0_USE_DIV: 1 int XILINX_MICROBLAZE0_USE_HW_MUL: 2 int

Note: Earlier, the eSDKs were extracted in the specified path, but now they are in self-extractable TAR files. From the 2021 release onwards, your eSDK scripts have the same names. They are extracted into <plnx-proj-root>/components/yocto when you run the petalinux-config or the petalinux-build command in the PetaLinux project. The project extracts the corresponding eSDK; for example, if you create a Zynq UltraScale+ MPSoC project, only the aarch64 eSDK is extracted into the <plnx-proj-root>/components/yocto project.

XSCT and toolchains

PetaLinux tool uses the XSCT underneath for all the embedded software applications configuration tool.

PetaLinux Command Line Interface (CLI) tools

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

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

Navigating Content by Design Process

AMD Adaptive Computing documentation is organized around a set of standard design processes to help you find relevant content for your current development task. You can access the AMD Versal™ adaptive SoC design processes on the [Design Hubs](#) page. You can also use the [Design Flow Assistant](#) to better understand the design flows and find content that is specific to your intended design needs.

- **Embedded Software Development:** Creating the software platform from the hardware platform and developing the application code using the embedded CPU. Also covers XRT and Graph APIs. Topics in this document that apply to this design process include:
 - [Chapter 3: Creating a Project](#)
 - [Chapter 4: Configuring and Building](#)
 - [Chapter 5: Packaging and Booting](#)
 - [Chapter 7: Customizing the Project](#)
 - [Chapter 8: Customizing the Root File System](#)

Setting Up Your Environment

Installation Steps

Installation Requirements

The PetaLinux tools installation requirements are:

- Minimum workstation requirements:
 - 8 GB RAM (recommended minimum for AMD tools)
 - 2 GHz CPU clock or equivalent (minimum of eight cores)
 - 100 GB free HDD space
 - Supported OS:
 - Completely removed RHEL and CENTOS to align with upstream Yocto.
 - Ubuntu Desktop/Server 18.04.1 LTS, 18.04.2 LTS, 18.04.3 LTS, 18.04.4 LTS, 18.04.5 LTS, 18.04.06 LTS, 20.04.2 LTS, 20.04.3 LTS, 20.04.4 LTS, 20.04.5 LTS(64-bit), 20.04.6 LTS, 22.04 LTS, 22.04.1 LTS and 22.04.2 LTS
 - OpenSuse Leap 15.3 and 15.4
 - AlmaLinux 8.7 and 9.1
- You need access to install the required packages mentioned in the release notes. The PetaLinux tools need to be installed as non-root users.
- PetaLinux requires standard development tools and libraries installed on your Linux host workstation. Install the libraries and tools listed in the [release notes](#) on the host Linux.
- PetaLinux tools require your host system /bin/sh to be 'bash.' If you use Ubuntu distribution and your /bin/sh is 'dash,' consult your system administrator to change your default system shell /bin/sh with the `sudo dpkg-reconfigure dash` command.

Note: For package versions, refer to the [PetaLinux 2023.2 Release Notes](#) and Master Answer Record: [000035006](#).



CAUTION! Consult your system administrator if unsure about the correct host system package management procedures.



IMPORTANT! PetaLinux 2023.2 works only with hardware designs exported from AMD Vivado™ Design Suite 2023.2.

Prerequisites

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

Installing the PetaLinux Tool

The PetaLinux tool is installed into the current working directory without any options tool is installed into the current working directory.

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

Alternatively, you can specify an installation path.

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

Table 4: PetaLinux Installer Options

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

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

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

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

This installs the PetaLinux tool into the `/home/<user>/petalinux/<petalinux-version>` directory. By default, it installs all three eSDKs compressed script files. To install a specific eSDK as part of the PetaLinux tool, see [Installing a Preferred eSDK as part of the PetaLinux Tool](#).



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

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

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

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

Installing a Preferred eSDK as part of the PetaLinux Tool

As described in [Installing the PetaLinux Tool](#), the PetaLinux tool has three eSDKs: aarch64, arm, and MicroBlaze.

While installing the tool, you can specify your preferred eSDK, for example, if you are working on a Zynq platform, you can opt to install the Arm eSDK into the PetaLinux tool.

By default, all platform eSDKs are installed into the tool install directory. To install the desired eSDK, follow these examples:

- To install eSDKs for all AMD-supported architectures like Zynq, Zynq UltraScale+ MPSoC, Versal, MicroBlaze:

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

- To install only the Zynq eSDK for arm architecture:

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

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

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

- To install MicroBlaze device eSDKs for MicroBlaze architecture:

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

Troubleshooting

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

Table 5: PetaLinux Installation Troubleshooting

Problem / Error Message	Description and Solution
WARNING: You have less than 1 GB free space on the installation drive	<p>Problem Description: This warning message indicates that the installation drive is almost full. After the installation, it's possible that you might not have sufficient available space to carry out the development of the hardware or software project.</p> <p>Solution: Clean up the installation drive to clear some more free space. Alternatively, move the PetaLinux installation to another hard disk drive.</p>
WARNING: No tftp server found	<p>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.</p> <p>Solution: Enable the TFTP service on your workstation. Contact your system administrator if you are unsure how to enable this service.</p>
ERROR: GCC is not installed - unable to continue. Please install and retry	<p>Problem Description: This error message indicates that you do not have gcc installed on the host workstation.</p> <p>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.</p>
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	<p>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."</p> <p>Solution: Install the packages of the missing tools. For more information, see Installation Requirements.</p>

Table 5: PetaLinux Installation Troubleshooting (cont'd)

Problem / Error Message	Description and Solution
<code>./petalinux-v<petalinux-version>-final-installer.run: line 52: petalinux_installation_log: Permission denied</code>	Problem Description: This error message indicates that the PetaLinux installed directory does not have writable permissions. Solution: Set 755 permissions to the install directory.
PetaLinux: Project fails to build on RHEL or CentOS 7.8 when encryption and FIPS are enabled	https://www.xilinx.com/support/answers/76518.html

PetaLinux Working Environment Setup

After the installation, the remaining setup is completed 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. PetaLinux can work with any interactive shell, and examples are as follows:

- For Bash as a user login shell:

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

- For C shell as a user login shell:

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

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

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

2. Verify that the working environment has been set:

```
echo $PETALINUX
```

Example output: /opt/pkg/petalinux

Environment variable \$PETALINUX should point to the installed PetaLinux path. The output can differ from this example based on the PetaLinux installation path.

Troubleshooting

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

Table 6: PetaLinux Working Environment Troubleshooting

Problem / Error Message	Description and Solution
WARNING: /bin/sh is not bash	<p>Problem Description: This warning message indicates that your default shell is linked to dash.</p> <p>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 <code>sudo dpkg-reconfigure dash</code> command.</p>
Failed to open PetaLinux lib	<p>Problem Description: This error message indicates that a PetaLinux library failed to load. The possible reasons are:</p> <ul style="list-style-type: none"> The PetaLinux <code>settings.sh</code> 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. <p>Solution:</p> <ol style="list-style-type: none"> Source the <code>settings.sh</code> script from the top-level PetaLinux directory. For more information, see PetaLinux Working Environment Setup. 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. <pre data-bbox="616 1332 1480 1450">cd \$PETALINUX/tools/xsct/lib/lnx64.o chcon -R -t textrel_shlib_t lib</pre>

Design Flow Overview

In general, the PetaLinux tool follows a sequential workflow model. The following table 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 7: Design Flow Overview

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

Creating a Project

Creating a Project Using PetaLinux BSP

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

BSPs are not included in the PetaLinux tools installer and need to be downloaded separately. PetaLinux BSP packages are available on the [Xilinx.com Download Center](#). A README is available for each BSP.

Note: Download only the BSPs you need.



CAUTION! BSPs can cause conflict with custom board settings. If you are using a custom board, create a project using template flow and import the custom board design with `petalinux-config --get-hw-description`.

Prerequisites

This section assumes that the following prerequisites are 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](#).

Creating 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
```

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

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

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

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

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



CAUTION! Do not create the symbolic link to an NFS file system from a local file system. You cannot use NFS to locate the 'tmp' directory in the build; it fails.

Note: PetaLinux requires a minimum of 50 GB and a maximum of 100 GB space on TMPDIR to build the project successfully when you create the project on NFS.

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

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

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

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



CAUTION! Do not create PetaLinux projects in the tool install area or use the tool install area as a tmp build area.

To enable XRT if not enabled in BSP, click [Create PetaLinux Project with XRT](#).

Troubleshooting

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

Table 8: PetaLinux BSP Installation Troubleshooting

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

Configuring a Hardware Platform with Vivado Design Suite

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

Configuring a Hardware Platform for Linux

You can create your hardware platform with AMD Vivado™ tools. Regardless of how the hardware platform is created and configured, a small number of hardware IP and software platform configuration settings are required to make the hardware platform Linux ready. These are described as following:

Zynq UltraScale+ MPSoC and Versal Adaptive SoC

The following is a list of hardware requirements for an AMD Zynq™ UltraScale+™ MPSoC and an AMD Versal™ adaptive SoC hardware project to boot Linux:

- External memory of at least 2 GB (required).
- UART for serial console (required).
- Non-volatile memory, for example, QSPI flash and SD/MMC. This memory is optional, without which only JTAG boot can work.
- Ethernet (optional, essential for network access).



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

Zynq 7000 Devices

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

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



IMPORTANT! If multiple TTCs are enabled, the Zynq 7000 Linux kernel uses the first TTC block from the device tree. Make sure others do not use the TTC.

- External memory controller with at least 512 MB of memory (required).
- UART for serial console (required).
- Non-volatile memory, for example, QSPI flash and SD/MMC. This memory is optional, but only the JTAG boot can work.
- Ethernet (optional, essential for network access).



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

MicroBlaze processors (AXI)

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

- IP core checklist:
 - External memory controller with at least 512 MB of memory (required)
 - Dual channel timer with interrupt connected (required)
 - UART with interrupt connected for serial console (required)
 - Non-volatile memory such as Linear Flash or SPI Flash (required)
 - Ethernet with interrupt connected (optional, but required for network access)
- MicroBlaze processor configuration:
 - MicroBlaze processor initial boot loader fs-boot needs a minimum of 4 KB of block RAM for parallel flash and 8 KB for SPI flash when the system boots from non-volatile memory.

Note: PetaLinux only supports 32-bit MicroBlaze processors.

Note: Do not disable any instruction set-related options enabled by the template unless you understand the implications of such a change.

Exporting a Hardware Platform to PetaLinux Project

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

Prerequisites

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

Exporting a Hardware Platform

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

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

Creating an Empty Project from a Template

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

Prerequisites

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

Creating an Empty Project

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

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

The parameters are as follows:

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

Note: The MicroBlaze template option cannot be used with Zynq 7000 devices or Zynq UltraScale+ or Versal designs in the Programmable Logic (PL).

- `--name <PROJECT_NAME>` - The name of your project.

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

For more information, see [Importing Hardware Configuration](#).



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

Configuring and Building

Version Control

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

Prerequisites

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

Version Control

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

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

By default, these files are added into <plnx-proj-root>/.gitignore while creating the project.

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

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

Importing Hardware Configuration

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

Prerequisites

This section assumes that the following prerequisites are satisfied:

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

Importing a 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 using the following steps:

- Using path of the directory containing the .xsa file:

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

Or

- Using the XSA file path as follows:

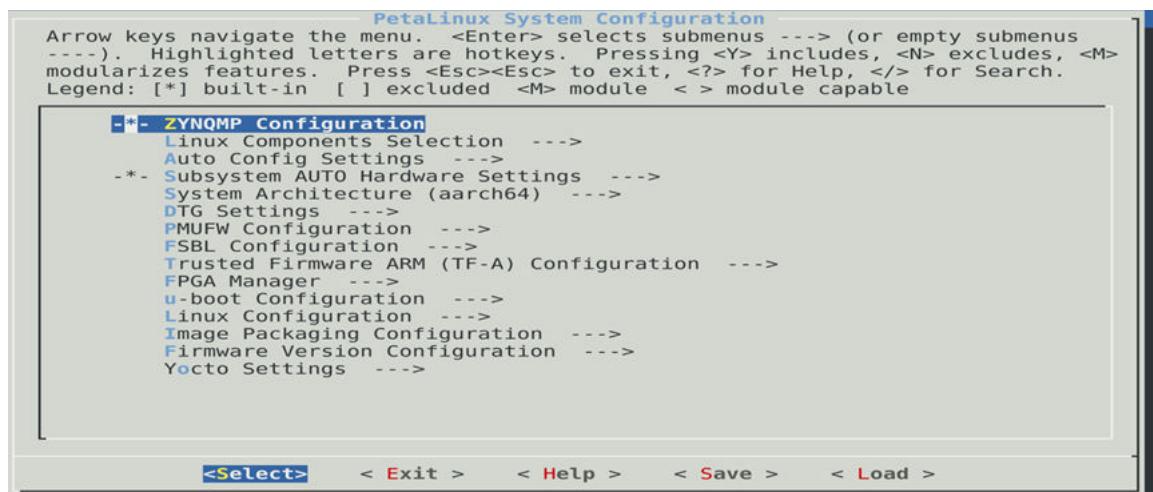
When you have multiple XSA files in a specified folder petalinux-config gives you an error. To avoid these kind of errors you can provide complete XSA file path as follows:

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

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

3. This launches the top system configuration menu. When the `petalinux-config --get-hw-description` command runs for the PetaLinux project, the tool detects the changes in the imported hardware design and launches the menu.

Figure 1: System Configuration Menu



Ensure **DTG Settings → (template) MACHINE_NAME** is selected and use the option as follows:

- Keep the BSP setting as AUTO when using an AMD evaluation board with the default BSP/hardware project. The DTG machine configuration is automatically determined based on your specific design.
- For the custom board, keep the default value.
- For customized hardware design implemented on an AMD evaluation board, use the values in the following table.

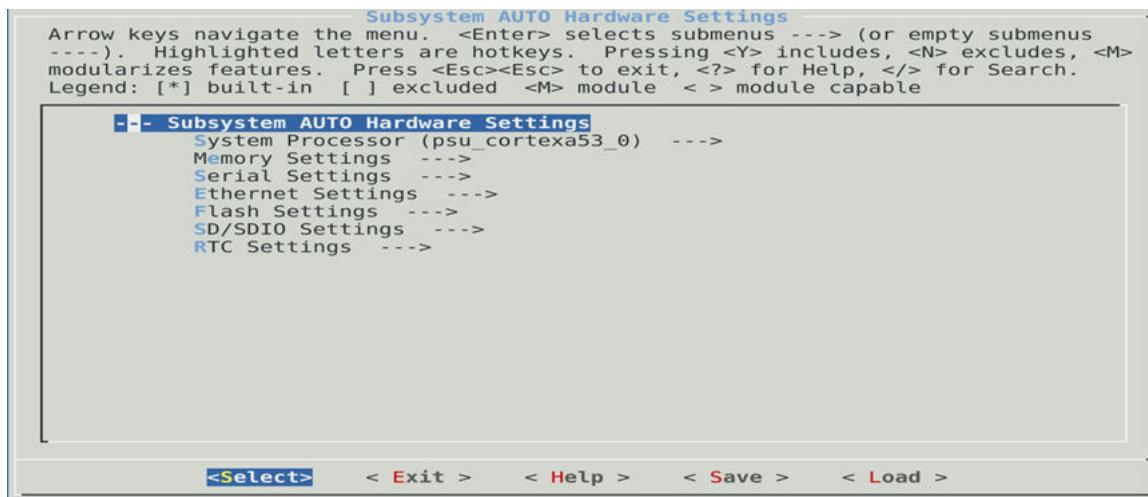
Table 9: BSP and Machine Names

BSP	Machine
ZCU102	zcu102-rev1.0
ZCU104	zcu104-revc
ZCU106	zcu106-reva
ZCU111	zcu111-reva
ZCU1275	zcu1275-revb
ZCU1285	zcu1285-reva
ZCU216	zcu216-reva
ZCU208	zcu208-reva

Table 9: BSP and Machine Names (cont'd)

BSP	Machine
ZCU670	zcu670-revb
ZCU208-SDFEC	zcu208-reva
ZCU100	zcu100-revc
ZC702	zc702
ZC706	zc706
ZEDBOARD	zedboard
AC701	ac701-full
KC705	kc705-full
KCU105	kcu105
VCU118	vcu118-rev2.0
SP701	sp701-rev1.0
VCK190	versal-vck190-reva-x-ebm-01-reva
VPK120	versal-vpk120-reva
VMK180	versal-vmk180-reva-x-ebm-01-reva
VPK180	versal-vpk180-reva

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

Figure 2: Subsystem AUTO Hardware Settings

The **Subsystem AUTO Hardware Settings** → menu allows customizing system-wide hardware settings. You have a choice to select the processor, memory bank, serial console, or flash.

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

Note: For more details on the Auto Config Settings menu, see the [Auto Config Settings](#).

Silentconfig

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

You can change the configurations without opening system level configuration menu. You can directly edit `<plnx-proj-root>/project-spec/configs/config` file and issue `petalinux-config --silentconfig`.

Building a System Image

Prerequisites

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

Building a PetaLinux System Image

1. Change into the directory of your PetaLinux project.

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

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

```
petalinux-build
```

This step generates a device tree DTB file, a first stage boot loader (for Zynq devices, Zynq UltraScale+ MPSoC, and MicroBlaze), PLM (for Versal adaptive SoC), PSM (for Versal adaptive SoC) and TF-A (for Zynq UltraScale+ MPSoC and Versal adaptive SoC), U-Boot, the Linux kernel, a root file system image, and the U-Boot boot script (`boot.scr`). Finally, it generates the necessary boot images.

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



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

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

The console shows the compilation progress. For example:

```
petalinux-build
[INFO] Sourcing buildtools
[INFO] Building project
[INFO] Silentconfig project
[INFO] Silentconfig rootfs
[INFO] Generating workspace directory
INFO: bitbake petalinux-image-minimal
NOTE: Started PRServer with DBfile: xilinx-vck190-2023.2/build/cache/prserv.sqlite3, Address: 127.0.0.1:44081, PID: 24954
Loading cache: 100%
|
| ETA: ---:--:--
Loaded 0 entries from dependency cache.
Parsing recipes: 100% |
#####
Time: 0:00:29
Parsing of 4414 .bb files complete (0 cached, 4414 parsed). 6351 targets,
348 skipped, 1 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
NOTE: Fetching uninative binary shim file:///xilinx-vck190-2023.2/
components/yocto/downloads/uninative/
3dd82c3fbdb59e87bf091c3eef555a05fae528eeda3083828f76cd4deaceca8b/x86_64-
nativesdk-
libc-3.9.tar.xz;sha256sum=3dd82c3fbdb59e87bf091c3eef555a05fae528eeda3083828f
76cd4deaceca8b (will check PREMIRRORS first)
Initialising tasks: 100% |
#####
Time: 0:00:05
Checking sstate mirror object availability: 100% |
#####
Time: 0:00:43
Sstate summary: Wanted 1752 Local 57 Mirrors 1204 Missed 491 Current 0 (71%
match, 0% complete)
NOTE: Executing Tasks
WARNING: zocl-202320.2.16.0-r0 do_package_qa: QA Issue: File /lib/modules/
6.1.30-xilinx-v2023.2/extrazocl.ko in package kernel-module-zocl-6.1.30-
xilinx-v2023.2 contains reference to TMPDIR [buildpaths]
NOTE: Tasks Summary: Attempted 4631 tasks of which 3751 didn't need to be
rerun and all succeeded.

Summary: There was 1 WARNING message.
INFO: Failed to copy built images to tftp dir: /tftpboot
```

Default Boot Images

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

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



IMPORTANT! By default, besides the kernel, RootFS, and U-Boot, the PetaLinux project is configured to generate and build the other boot components for AMD Zynq™ FSBL, for Zynq UltraScale+ MPSoC FSBL and PMU firmware and for Versal PLM and PSM firmware . For more details on the auto generated boot components, see [Appendix D: Generating Boot Components](#).

Troubleshooting

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

Warnings/Errors

Table 10: Warnings/Errors Description

Warning or Error Message	Description	Solution
Skipping recipe linux-xlnx as it does not produce a package with the same name	It appears if the provided recipe name does not match with the packages provided by it, for example, if <code>linux-xlnx</code> provides <code>kernel-image</code> , <code>kernel-base</code> , <code>kernel-dev</code> , and <code>kernel-modules</code> packages and these does not match with the name <code>linux-xlnx</code> which was in workspace directory.	You can ignore this warning message.

Table 10: Warnings/Errors Description (cont'd)

Warning or Error Message	Description	Solution
<pre><package-name> do_package: Could not copy license file <plnx- proj-root>/components/ yocto/layers/core/meta/ files/common-licenses/ to /opt/pkg/petalinux/ build/tmp/work/<machine- name>-xilinx-linux/ image/usr/share/licenses/ <package-name>/COPYING.MIT [Errno 1] Operation not permitted:</pre>	<p>When the tool is installed, all license files in <code><plnx-proj-root>/components/yocto/layers/core/meta/files/common-licenses/</code> have 644 permissions. Therefore, they are readable by others but not writable.</p>	<ul style="list-style-type: none"> Method 1: Manually modify permissions of the license files coming from the layers <code>chmod 666 <plnx-proj-root>/components/yocto/layers/core/meta/files/common-licenses/*</code> When creating the hard link, you have write permissions to the source of the link. Method 2: Disable hard linking protection on the kernel <code>sysctl fs.protected_hardlinks=0</code> The kernel does not allow the source to be writable by the current user when creating the hard link. Method 3: Set the following Yocto variables in <code><plnx-proj-root>/meta-user/conf/petalinuxbsp.conf</code> <code>LICENSE_CREATE_PACKAGE_forcevariable = "0"</code> <code>SIGGEN_LOCKEDSIGS_TASKSIG_CHECK = "none"</code> The build system does not try to create the link and the license is not on the final image. <p>Note: It is recommended to use the troubleshooting solutions specified in method 1 and method 3 because method 2 requires all the users to have sudo access.</p>

Build Optimizations

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

Deselecting Default Components

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

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

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

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

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

Note: If the FSBL, PMU firmware, PLM firmware, and PSM firmware are not built with PetaLinux, they must be built in the AMD Vitis™ software platform. The final load image must include all required components regardless of which tool built them.

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 <https://petalinux.xilinx.com/sswreleases/rel-v2023/>. Follow these steps to work with local, NFS, or the internal web server copy of sstate in PetaLinux. You can download the sstate from the download area along with PetaLinux.

Table 11: Local Mirror Servers

Server	Description
downloads	Source of download files are available.
aarch64	sstate mirrors for Zynq UltraScale+ MPSoC/RFSoC , and Versal devices.
arm	sstate mirrors for Zynq 7000 SoCs.
microblaze	sstate mirrors for MicroBlaze™ processors.

Source Mirrors

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

Example: `file:///home/user/daily-downloads-2023`

Reduce Build Time

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

Sstate Feeds

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

- sstate feeds on NFS: Go to **petalinux-config** → **Yocto Settings** → **Local sstate feeds settings** and enter the full path of the sstate directory. By enabling this option, you can point to your own shared state which is available at a NFS/local mount point.
For example, to enable, use `/opt/petalinux/sstate-cache_2020/aarch64`.
- sstate feeds on webserver: Go to **petalinux-config** → **Yocto Settings** → **Enable Network sstate feeds** → **Network sstate feeds URL** and enter the URL for sstate feeds.
For more information, see [How to reduce build time using SSTATE CACHE](#).

Packaging and Booting

Packaging Boot Image

This section describes how to generate the Boot Image for PetaLinux Build images.

Generate Boot Image for Versal Adaptive SoC

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

Prerequisites

This section assumes that you have built PetaLinux system image. For more information, see [Building a System Image](#).

Generate Boot Image

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

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

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

Note: Specifying `--u-boot` adds all the required images to boot up to U-Boot into `BOOT.BIN`

```
petalinux-package --boot --u-boot
[INFO] Sourcing buildtools
INFO: File in BOOT BIN: "xilinx-vck190-2023.2/project-spec/hw-description/vpl_gen_fixed.pdi"
INFO: File in BOOT BIN: "xilinx-vck190-2023.2/images/linux/plm.elf"
INFO: File in BOOT BIN: "xilinx-vck190-2023.2/images/linux/psmfw.elf"
INFO: File in BOOT BIN: "xilinx-vck190-2023.2/images/linux/system-default.dtb"
INFO: File in BOOT BIN: "xilinx-vck190-2023.2/images/linux/bl31.elf"
INFO: File in BOOT BIN: "xilinx-vck190-2023.2/images/linux/u-boot.elf"
INFO: Generating versal binary package BOOT.BIN...

***** Bootgen v2023.2
```

```
**** Build date : Apr 7 2023-10:18:04
** Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
** Copyright 2022-2023 Advanced Micro Devices, Inc. All Rights Reserved.

[INFO]      : Bootimage generated successfully

INFO: Generating QEMU boot images...
INFO: File in qemu_boot.img: xilinx-vck190-2023.2/images/linux/BOOT.BIN
INFO: File in qemu_boot.img: xilinx-vck190-2023.2/images/linux/
ramdisk.cpio.gz.u-boot
INFO: File in qemu_boot.img: xilinx-vck190-2023.2/images/linux/boot.scr
INFO: Binary is ready.
WARNING: Unable to access the TFTPBOOT folder /tftpboot!!!
WARNING: Skip file copy to TFTPBOOT folder!!!
```

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

To change the DTB load address, use this command:

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

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

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

For detailed usage, see the `--help` option and [petalinux-package --boot](#).

Generate MCS Image

An MCS image for Versal usually contains a PDI file (imported from hardware design), PLM, PSM firmware, Arm® trusted firmware, U-Boot, DTB, and Kernel Fit image (optional).

Execute the following command to generate the MCS image to boot up to U-Boot using build images:

```
petalinux-package --boot --u-boot --format MCS
```

Note: Specifying `--u-boot` adds all the required images to boot up to U-Boot into `boot.mcs` file.

Execute the following command to generate the MCS image to boot up to U-Boot using prebuilt images:

```
cd <plnx-proj-root>
$ petalinux-package --boot --u-boot pre-built/linux/images/u-boot.elf --dtb
pre-built/linux/images/system.dtb --plm pre-built/linux/images/plm.elf --
psmfw pre-built/linux/images/psmfw.elf --atf pre-built/linux/images/
bl31.elf --format MCS
```

This generates `boot.mcs` in `<plnx-proj-root>/images/linux` directory. The default DTB load address is `0x1000`. For more information on Bootgen arguments, see *Bootgen User Guide* ([UG1283](#)).

Execute the following command to generate the MCS image to boot up to Linux using build images:

```
petalinux-package --boot --u-boot --kernel --offset 0xF40000 --format MCS
```

Execute the following command to generate the MCS image to boot up to Linux using prebuilt images:

```
cd <plnx-proj-root>
$ petalinux-package --boot --u-boot pre-built/linux/images/u-boot.elf --dtb pre-built/linux/images/system.dtb --plm pre-built/linux/images/plm.elf --psmfw pre-built/linux/images/psmfw.elf --atf pre-built/linux/images/b131.elf --kernel pre-built/linux/images/image.ub --offset 0xF40000 --boot-script pre-built/linux/images/boot.scr --format MCS
```

This generates `boot.mcs` in `images/linux` directory containing images to boot up to Linux using Fit image (with tiny root file system if `switch_root` is enabled) loaded at `0xF40000` Flash offset.

For detailed usage, see the `--help` option and [petalinux-package --boot](#).

Generate Boot Image for Zynq UltraScale+ MPSoC

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

Prerequisites

This section assumes that you have built the PetaLinux system image. For more information, see [Building a System Image](#).

Generate Boot Image

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

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

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

Note: Specifying `--u-boot` adds all the required images to boot up to U-Boot into the `BOOT.BIN`.

```
petalinux-package --boot --u-boot pre-built/linux/images/u-boot.elf --dtb
pre-built/linux/images/system.dtb --fsbl pre-built/linux/images/
zynq_fsbl.elf --format MCS
[INFO] Sourcing buildtools
INFO: File in BOOT BIN: "xilinx-zc706-2023.2/pre-built/linux/images/
zynq_fsbl.elf"
INFO: File in BOOT BIN: "xilinx-zc706-2023.2/pre-built/linux/images/u-
boot.elf"
INFO: File in BOOT BIN: "xilinx-zc706-2023.2/pre-built/linux/images/
system.dtb"
INFO: Generating zynq binary package boot.mcs...

*****
* Bootgen v2023.1
* Build date : Apr 7 2023-10:18:04
** Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
** Copyright 2022-2023 Advanced Micro Devices, Inc. All Rights Reserved.

[INFO] : Bootimage generated successfully

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

For detailed usage, see the `--help` option or [petalinux-package --boot](#).

Generate MCS Image

An MCS image for Zynq UltraScale+ MPSoC usually contains a First Stage Boot Loader Image (FSBL), FPGA bitstream, PMU firmware, TF-A, U-Boot, DTB, and Kernel Fit image (optional).

Execute the following command to generate the MCS image to boot U-Boot using build images:

```
petalinux-package --boot --u-boot --format MCS
```

Note: Specifying `--u-boot` adds all the required images to boot up to U-Boot into `boot.mcs` file.

Execute the following command to generate the MCS image to boot up to U-Boot using prebuilt images:

```
petalinux-package --boot --u-boot pre-built/linux/images/u-boot.elf --dtb
pre-built/linux/images/system.dtb --pmufw pre-built/linux/images/pmufw.elf
--fsbl pre-built/linux/images/zynqmp_fsbl.elf --atf pre-built/linux/images/
bl31.elf --format MCS
```

This generates `boot.mcs` in `<plnx-proj-root>/images/linux` directory. For more information on Bootgen arguments see [Bootgen User Guide \(UG1283\)](#).

Execute the following command to generate the MCS image to boot up to Linux using build images:

```
petalinux-package --boot --u-boot --kernel --offset 0xF40000 --format MCS
```

Execute the following command to generate the MCS image to boot up to Linux using prebuilt images:

```
petalinux-package --boot --u-boot pre-built/linux/images/u-boot.elf --dtb pre-built/linux/images/system.dtb --pmufw pre-built/linux/images/pmu fw.elf --fsbl pre-built/linux/images/zynqmp_fsbl.elf --atf pre-built/linux/images/bl31.elf --kernel pre-built/linux/images/image.ub --offset 0xF40000 --boot-script pre-built/linux/images/boot.scr --format MCS
```

This generates `boot.mcs` in the `images/linux` directory containing images to boot up to Linux using Fit image (with tiny root file system if `switch_root` enabled) loaded at `0xF40000` Flash offset.

For detailed usage, see the `--help` option and [petalinux-package --boot](#).

Generate Boot Image for Zynq 7000 Devices

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

Prerequisites

This section assumes that you have built the PetaLinux system image. For more information, see [Building a System Image](#).

Generate Boot Image

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

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

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

Note: Specifying `--u-boot` adds all the required images to boot up to U-Boot into the `BOOT.BIN`.

```
petalinux-package --boot --u-boot --kernel --offset 0x1080000 --format MCS
[INFO] Sourcing buildtools
INFO: File in BOOT BIN: "xilinx-zc706-2023.2/images/linux/zynq_fsbl.elf"
INFO: File in BOOT BIN: "xilinx-zc706-2023.2/images/linux/u-boot.elf"
INFO: File in BOOT BIN: "xilinx-zc706-2023.2/images/linux/system.dtb"
INFO: File in BOOT BIN: "xilinx-zc706-2023.2/images/linux/boot.scr"
INFO: File in BOOT BIN: "xilinx-zc706-2023.2/images/linux/image.ub"
INFO: Generating zynq binary package boot.mcs...
```

```
***** Bootgen v2023.1
***** Build date : Apr 7 2023-10:18:04
** Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
** Copyright 2022-2023 Advanced Micro Devices, Inc. All Rights Reserved.

[WARNING]: Partition zynq_fsbl.elf.0 range is overlapped with partition
boot.scr.0 memory range
[WARNING]: Partition zynq_fsbl.elf.0 range is overlapped with partition
image.ub.0 memory range
[WARNING]: Partition system.dtb.0 range is overlapped with partition
image.ub.0 memory range
[WARNING]: Partition boot.scr.0 range is overlapped with partition
image.ub.0 memory range

[INFO]      : Bootimage generated successfully

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

For detailed usage, see the `--help` option or [petalinux-package --boot](#).

Generate MCS Image

An MCS image for Zynq usually contains a First stage boot loader image (FSBL), FPGA bitstream, Arm® trusted firmware, U-Boot, DTB, and Kernel Fit image (optional).

Execute the following command to generate the MCS image to boot up to U-Boot using build images:

```
petalinux-package --boot --u-boot --format MCS
```

Note: Specifying `--u-boot` adds all the required images to boot up to U-Boot into `boot.mcs` file.

Execute the following command to generate the MCS image to boot up to U-Boot using prebuilt images:

```
cd <plnx-proj-root>
petalinux-package --boot --u-boot pre-built/linux/images/u-boot.elf --dtb
pre-built/linux/images/system.dtb --fsbl pre-built/linux/images/
zynq_fsbl.elf --format MCS
```

This generates `boot.mcs` in `<plnx-proj-root>/images/linux` directory. For more information on Bootgen arguments, see [Bootgen User Guide \(UG1283\)](#).

Execute the following command to generate the MCS image to boot up to Linux using build images:

```
petalinux-package --boot --u-boot --kernel --offset 0x1080000 --format MCS
```

Execute the following command to generate the MCS image to boot up to Linux using prebuilt images:

```
cd <plnx-proj-dir>
petalinux-package --boot --u-boot pre-built/linux/images/u-boot.elf --dtb
pre-built/linux/images/system.dtb --fsbl pre-built/linux/images/
zyng_fsbl.elf --kernel pre-built/linux/images/image.ub --offset 0x1080000 --
boot-script pre-built/linux/images/boot.scr --format MCS
```

This generates `boot.mcs` in `images/linux` directory containing images to boot up to Linux using Fit image (with tiny root file system if `switch_root` enabled) loaded at `0x1080000` Flash offset.

For detailed usage, see the `--help` option and [petalinux-package --boot](#).

Generate Boot Image for MicroBlaze Processor

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

Prerequisites

Note: To generate MCS file, package the MMI file and bitstream as a part of Vivado or Vitis exported XSA.

This section assumes that you have built the PetaLinux system image. For more information, see [Building a System Image](#).

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

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

Generate Boot Image

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

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

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

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

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

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

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

For detailed usage, see the `--help` option or [petalinux-package --boot](#).

Note: PetaLinux only supports 32-bit MicroBlaze processors.

Modify Bitstream File for MicroBlaze Processor

Prerequisites

This section assumes that you have built the PetaLinux system image and FSBL. For more information, see [Building a System Image](#).

Modify Bitstream

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

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

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

Note: PetaLinux only supports 32-bit MicroBlaze processors.

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 are satisfied:

- For Zynq UltraScale+ MPSoC, Zynq 7000 devices, and Versal devices, you have generated the boot image. For more information, see [Generate Boot Image for Zynq UltraScale+ MPSoC](#).
- For MicroBlaze processors, you have generated the system image. For more information, see [Building a System Image](#).

Packaging Prebuilt Image

1. Change into the root directory of your project.

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

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

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

For detailed usage, see the `--help` option or the [petalinux-package --prebuilt](#).

BSP Packaging

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

BSP Packaging

Steps on how to package a project are as follows:

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

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

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

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

Additional BSP Packaging Options

1. BSP packaging with hardware source.

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

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

2. Exclude workspace changes.

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

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

Alternatively, you can clean the project before executing the `petalinux-package --bsp` command as follows:

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

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

3. BSP packaging with external sources.

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

Booting PetaLinux Prebuilt Images

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

Prerequisites

This section assumes that you have packaged prebuilt images. For more information, see [Packaging Prebuilt Images](#).

Boot Levels for Prebuilt Option

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

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

The command for the QEMU boot is as follows:

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

Note: The QEMU boot does not support BOOT_LEVEL 1.

- Level 1: Download the prebuilt FPGA bitstream.
 - It boots FSBL and PMU firmware for AMD 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 TF-A before booting U-Boot.
 - For AMD Versal™ devices: It loads the required cdo files from .pdi file, PLM, PSM firmware, and TF-A before booting U-Boot.
- Level 3: Download and boot the prebuilt Linux
 - 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 prebuilt FPGA bitstream, prebuilt FSBL, prebuilt PMU firmware, prebuilt TF-A, prebuilt U-Boot and prebuilt kernel on target.
 - For Versal devices: It loads the required cdo files from .pdi file, PLM, PSM firmware, TF-A, and U-Boot before booting Kernel.

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

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

Booting PetaLinux Image on QEMU

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

For details on AMD IP Models supported by QEMU, see [Appendix F: AMD IP Models Supported by QEMU](#).

Prerequisites

This section assumes that the following prerequisites are satisfied:

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



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

Booting a PetaLinux Image on QEMU

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

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

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

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

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

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

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

To know more about different boot levels for prebuilt option, see [Booting PetaLinux Prebuilt Images](#).



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

Additional Options for Booting on QEMU

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

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

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

- To download the newly built kernel with QEMU:

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

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

Note: For Versal adaptive SoC, QEMU boots up to the tiny root file system if `switch_root` is enabled and gives you the `rootfs` prompt with `ERROR: There's no '/dev' on rootfs..` message. This is because PetaLinux uses the SD boot mode in QEMU and the SD image has only FAT with tiny rootfs by default.

After Linux booted successfully, ensure to first login with username `petalinux` and change the password. Try commands such as `ls`, `ifconfig`, `cat /proc/cpuinfo` and `cat /proc/meminfo`. They behave the same as on real hardware. To exit the emulator when you are finished,

1. press **Ctrl A**
2. release

3. press X

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

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

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

- For Zynq UltraScale+ MPSoC and Versal adaptive SoC, use `image`:

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

- For Zynq 7000 devices, use `zImage`:

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

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

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

- To download a customized root file system image with `--rootfs` option:

```
petalinux-boot --qemu --kernel --rootfs <specify custom cpio.gz.u-boot path>
```

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

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

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

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

This is not supported by Versal adaptive SoCs.

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

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

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

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

- To change the boot mode to QSPI for the board which does not have SD :

- The default petalinux-boot command with QEMU work in SD boot mode. If your board does not have the SD and support QSPI or OSPI, follow the steps to change the boot mode to QSPI or OSPI

- **Note:** In the following command, boot mode is specified as 8, which works with vck190 board. To check the boot mode number of the respective board, refer to the board user guide of that board mentioned in the BSP README file.

```
petalinux-package --boot --uboot
dd if=/dev/zero bs=1G count=1 > <QEMU_QSPI>.bin
dd if=prebuilt/linux/images/BOOT.BIN of=<QEMU_QSPI>.bin bs=1 seek=0
conv=notrunc
petalinux-boot --qemu --prebuilt 3 --qemu-args=" -drive
file=<QEMU_QSPI>.bin,if=mtd,format=raw,index=4 -boot mode=8"
booti [<addr> [<initrd>[:<size>]] [<fdt>]]
```

Note: QEMU version has been upgraded to 7.1. The old options are deprecated in the new version but remain functionally operational. Warning messages are displayed because PetaLinux tools still use the old options. You can ignore them.

Booting PetaLinux Image on Hardware with an SD Card

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



IMPORTANT! This section is only for devices that allow booting from SD cards. This includes AMD Versal™ adaptive SoCs, AMD Zynq™ UltraScale+™ MPSoCs, and Zynq 7000 devices.

Prerequisites

This section assumes that the following prerequisites are 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 [Creating a Project Using PetaLinux BSP](#).
- A serial communication program such as minicom/kermit/gtkterm has been installed; the baud rate of the serial communication program has been set to 115200 bps.

Booting a PetaLinux Image on Hardware with SD Card

You can boot a PetaLinux image on hardware using an SD card by copying the required images manually or by flashing a WIC image into the mounted SD card.

Flashing and Booting the PetaLinux Images Manually

1. Partition the SD card by following [Appendix I: Partitioning and Formatting an SD Card](#).

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

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

3. Mount the ext4 partition and extract the rootfs.tar.gz folder into the ext4 partition of the SD card with sudo permissions.

```
-> mount /dev/mmcblk1p1 /mnt  
-> cd /mnt  
-> tar -xzvf /PATH/TO/rootfs.tar.gz
```

4. Connect the serial port on the board to your workstation.
5. 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.
6. Power off the board.
7. Set the boot mode of the board to SD boot. Refer to the board documentation for details.
8. Plug the SD card into the board.
9. Power on the board.
10. A boot message displays on the serial console.

Flashing and Booting the PetaLinux Image Using WIC Image

1. To generate the WIC image in PetaLinux project see [petalinux-package --wic Command Examples](#).
2. Go to the <plnx-proj-root>/images/linux or <plnx-proj-root>/pre-built/linux/images/ directory. For example, cd images/linux/.
3. If the WIC image is compressed with XZ format, extract the petalinux-sdimage.wic.xz file using xz command. For example, xz -d petalinux-sdimage.wic.xz.

Note: If xz package is not installed in your build machine, use the prebuilt xz binary from \$PETALINUX/components/yocto/buildtools/sysroots/x86_64-petalinux-linux/usr/bin/xz.

4. Flash the extracted petalinux-sdimage.wic image into the SD card.
 - **Flash the WIC image in Linux:** To flash the WIC image to SD card in Linux machines, connect the SD card to the host system and use the dd command:

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

Note: You need sudo access to do this.

- **Flash the WIC image in Windows:** To flash the WIC image to the SD card in Windows, you can use any of the following:
 - BalenaEtcher tool
 - Win32DiskImager
5. Insert the SD card into the board and boot the system.
 6. Connect the serial port on the board to your workstation.
 7. 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.
 8. Power off the board.
 9. Set the boot mode of the board to SD boot. Refer to the board documentation for details.
 10. Plug the SD card into the board.
 11. Power on the board. The boot logs display on the serial console:



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

Troubleshooting

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

Table 12: 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 <code>bootcmd</code> environment variable is not set properly. Solution: To see the default boot device, print <code>bootcmd</code> environment variable using the following command in U-Boot console. <code>U-Boot-PetaLinux> print bootcmd</code> If it is not run using <code>sdboot</code> flow, there are a few options as follows: <ul style="list-style-type: none"> • Without rebuild PetaLinux, set <code>bootcmd</code> to boot from your desired media, use <code>setenv</code> command. For SD card boot, set the environment variable as follows. <code>U-Boot-PetaLinux> setenv bootcmd 'run sdboot' ; saveenv</code> • Run <code>petalinux-config</code> to set to load kernel image from SD card. Rebuild PetaLinux and regenerate <code>BOOT.BIN</code> with the rebuilt U-Boot, and use the new <code>BOOT.BIN</code> to boot the board. See Generate Boot Image for Zynq UltraScale+ MPSoC on how to generate <code>BOOT.BIN</code>.
Error string observed while booting <code>[systemd[1]: File System Check on Root Device was skipped because of a failed condition check (ConditionPathIsReadWrite=!)</code> .	Error message indicates that the file system devices has read and write permission. No functional impact, you can ignore this.



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

Booting PetaLinux Image on Hardware with JTAG

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

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

Prerequisites

This section assumes that the following prerequisites are satisfied:

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

Booting a PetaLinux Image on Hardware with JTAG

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

```
petalinux-boot --jtag --prebuilt 3 --hw_server_url <hostname:3121>
```

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

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

Note: To know more about different boot levels for prebuilt option, see [Booting PetaLinux Prebuilt Images](#).

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

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

Additional Options for Booting with JTAG

- To download a bitstream to target board:

```
petalinux-boot --jtag --fpga --bitstream <BITSTREAM> --hw_server_url  
<hostname:3121>
```

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

```
petalinux-boot --jtag --u-boot --hw_server_url <hostname:3121>
```

- To download newly built kernel to target board:

```
petalinux-boot --jtag --kernel --hw_server_url <hostname:3121>
```

- For MicroBlaze™ processors, this boots `system.bit`, `u-boot.elf`, `linux.bin.ub`, `system.dtb`, and `rootfs.cpio.gz.u-boot` on target board.

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

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

- For AMD Zynq™ UltraScale+™ MPSoC, this boots `pmufw.elf`, `zynqmp_fsbl.elf`, `u-boot.elf`, `Image`, `system.dtb`, and `ramdisk.cpio.gz.u-boot` on target board.
- For Zynq 7000 devices, this boots `zynq_fsbl.elf`, `u-boot.elf`, `uImage`, `system.dtb`, and `rootfs.cpio.gz.u-boot` on target board.
- For AMD Versal™ adaptive SoC, this boots `BOOT.BIN`, `Image`, `ramdisk.cpio.gz.u-boot`, and `boot.scr` on target board.

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

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

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

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

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

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

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

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

- For Zynq UltraScale+ MPSoC and Versal adaptive SoC, use **Image**:

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

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

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

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

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

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

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

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

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

- To download a customized pmufw image with kernel:

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

Logging Tcl/XSDB for JTAG Boot

Use the following command to take log of XSDB commands used during JTAG boot. It dumps Tcl script (which 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 can experience while booting a PetaLinux image on hardware with JTAG.

Table 13: PetaLinux Image on Hardware with JTAG Troubleshooting

Problem / Error Message	Description and Solution
Cannot see any console output when trying to boot U-Boot or kernel on hardware but boots correctly on QEMU.	<p>Problem Description: This problem is usually caused by one or more of the following:</p> <ul style="list-style-type: none"> The serial communication terminal application is set with the wrong baud rate. Mismatch between hardware and software platforms. <p>Solution:</p> <ul style="list-style-type: none"> Ensure your terminal application baud rate is correct and matches your hardware configuration. Ensure the PetaLinux project is built with the right hardware platform. <ul style="list-style-type: none"> 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 Building a System Image).
Error string observed while booting [systemd[1]: File System Check on Root Device was skipped because of a failed condition check (ConditionPathIsReadWrite=!).]	Error message indicates that the file system devices has read and write permission. No functional impact, you can ignore this.

Booting 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 are satisfied:

- Host environment with TFTP server is setup and PetaLinux Image is built for TFTP boot. For more information, see [Configuring TFTP/PXE 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 are installed.

Booting 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 AMD Zynq™ 7000 devices and Zynq UltraScale+ MPSoC and AMD Versal™ device boards, ensure that the mode switches are set to JTAG mode. Refer to the board documentation for details.
6. Power on the board.
7. Open a console on your workstation and start with the preferred serial communication program (for example, kermit, minicom) with the baud rate set to 115200 on that console.
8. Run the `petalinux-boot` command as follows on your workstation

```
petalinux-boot --jtag --prebuilt 2 --hw-server-url <hostname:3121>
```

9. When autoboot starts, hit any key to stop it. An example of a workstation console output for successful U-Boot download is:

```
Hit any key to stop autoboot: 4
```

10. Check whether the TFTP server IP address is set to the IP Address of the host where the image resides. This can be done using the following command:

```
ZynqMP> print serverip
```

11. Set the server IP address to the host IP address using the following command:

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

12. Set the board IP address using the following command:

```
ZynqMP> setenv ipaddr <BOARD IP ADDRESS>
```

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

```
ZynqMP> pxe get
```

14. Boot the kernel using the following command:

```
ZynqMP> pxe boot
```

Troubleshooting

Table 14: PetaLinux Image on Hardware with TFTP

Problem / Error Message	Description and Solution
Error: "serverip" not defined.	<p>Problem Description: This error message indicates that U-Boot environment variable <code>serverip</code> is not set. You have to set it to the IP Address of the host where the image resides.</p> <p>Solution: Use the following command to set the <code>serverip</code>:</p> <pre>ZynqMP> setenv serverip <HOST IP ADDRESS>; saveenv</pre>
Error string observed while booting [systemd[1]: File System Check on Root Device was skipped because of a failed condition check (ConditionPathIsReadWrite=!).]	Error message indicates that the file system devices has read and write permission. No functional impact, you can ignore this.

Booting PetaLinux Image on Hardware with QSPI or OSPI

This section describes how to boot a PetaLinux image on hardware with QSPI/OSPI.

Prerequisites

This section assumes that the following prerequisites are satisfied:

- You have installed PetaLinux Tools on the Linux workstation. If you have not installed, see [Installation Steps](#).
- You have installed PetaLinux BSP on the Linux workstation. If you have not installed, see [Creating a Project Using PetaLinux BSP](#).
- You have successfully created BOOT.BIN with the newly built PetaLinux project. If you have not created, see [Packaging Boot Image](#).
- A serial communication program such as minicom, kermit, or gtkterm has been installed. The baud rate of the serial communication program has been set to 115200 bps.

Booting a PetaLinux Image on Hardware with QSPI or OSPI

You can boot a PetaLinux image on hardware using QSPI/OSPI by flashing/copying the Fit image (image.ub) or Individual images(Kernel image + Rootfs) to Flash memory.

Steps to Flash and Boot the PetaLinux Images using Fit image

1. Set Primary Flash as boot device and set the Boot script configuration to load the images into QSPI/OSPI flash. For more information, see [Configuring Primary Flash Partition](#) and [Configuring U-Boot Boot Script \(boot.scr\)](#).
2. Build the system image. For more information, see [Building a System Image](#).
3. Boot a PetaLinux Image on hardware with JTAG up to U-Boot, see [Booting PetaLinux Image on Hardware with JTAG](#).
4. Make sure you have configured TFTP server in host.
5. To check or update the offsets for kernel, boot.scr, and root file system, see [Configuring U-Boot Boot Script \(boot.scr\)](#). If they do not match, the process can fail at the U-Boot prompt.
6. Set the server IP address to the host IP address using the following command at U-Boot prompt.

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

- a. Detect Flash Memory.

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

- b. Erase Flash Memory.

```
ZynqMP> sf erase 0 <Flash size in hex>
```

- c. Read images onto Memory and write into Flash.

- Read BOOT.BIN.

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

- Write BOOT.BIN.

```
ZynqMP> sf write 0x80000 0x0 $filesize
```

Example: sf write 0x80000 0x0 0x121e7d0

- Read image.ub.

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

- Write image.ub.

```
ZynqMP>sf write 0x80000 <Fit Image Flash Offset Address> $filesize
```

Example: sf write 0x80000 0xF40000 0xF00000

- Read boot.scr

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

- Write boot.scr

```
ZynqMP> sf write 0x80000 <boot.scr Flash Offset Address> $filesize
```

Example: sf write 0x80000 0x03e80000 0x80000

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

Steps to Flash and Boot the PetaLinux Images using Individual images

1. Set Primary Flash as boot device and Adjust/set the Boot script configuration to load the images into QSPI/OSPI flash. For more information, see [Configuring Primary Flash Partition](#) and [Configuring U-Boot Boot Script \(boot.scr\)](#).
2. Build the system image. For more information, see [Building a System Image](#).
3. Boot a PetaLinux Image on hardware with JTAG up to U-Boot, see [Booting PetaLinux Image on Hardware with JTAG](#).
4. Make sure you have configured TFTP server in host.

Note: To check or update the offsets for kernel, boot.scr, and root file system see [Configuring U-Boot Boot Script \(boot.scr\)](#). If they do not match, the process can fail at the U-Boot prompt.

5. Set the server IP address to the host IP address using the following command at U-Boot prompt.

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

- a. Detect Flash Memory.

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

- b. Erase Flash Memory.

```
ZynqMP> sf erase 0 <Flash size in hex>
```

- c. Read images onto Memory and write into Flash.

- Read BOOT.BIN.

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

- Write BOOT.BIN.

```
ZynqMP> sf write 0x80000 0x0 $filesize
```

Example: sf write 0x80000 0x0 0x121e7d0

- Read Image.

```
ZynqMP> tftpboot 0x80000 Image
```

- Write Image.

```
ZynqMP> sf write 0x80000 <Kernel Image Flash Offset Address>
$filesize
```

Example: sf write 0x80000 0xF00000 0x1474200

- Read rootfs.cpio.gz.u-boot.

```
ZynqMP> tftpboot 0x80000 rootfs.cpio.gz.u-boot
```

- Write rootfs.cpio.gz.u-boot.

```
ZynqMP>sf write 0x80000 <Rootfs Flash Offset Address> $filesize
```

Example: sf write 0x80000 0x4000000 0x4e1933

- Read boot.scr

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

- Write boot.scr

```
ZynqMP> sf write 0x80000 <boot.scr Flash Offset Address> $filesize
```

Example: sf write 0x80000 0x03e80000 0x80000

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

Troubleshooting

This section describes some common issues you can experience while booting a PetaLinux image on hardware with QSPI/OSPI Flash.

Table 15: 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 the Kernel offset in the boot.scr and the Kernel image offset in the Flash do not match. Solution: To see the default boot offsets of SPI, click Configuring U-Boot Boot Script (boot.scr)
Wrong Image Format for bootm command ERROR: can't get kernel image! Booting using Fit image failed	Problem Description: This error message appears when you load Image and ramdisk.cpio.gz.uboot for QSPI/OSPI/NAND boot mode. You can ignore this error message.

Table 15: PetaLinux Image on Hardware Troubleshooting (cont'd)

Problem / Error Message	Description and Solution
Error string observed while booting [systemd[1]: File System Check on Root Device was skipped because of a failed condition check (ConditionPathIsReadWrite=!/).]	Error message indicates that the file system devices has read and write permission. No functional impact, you can ignore this.



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

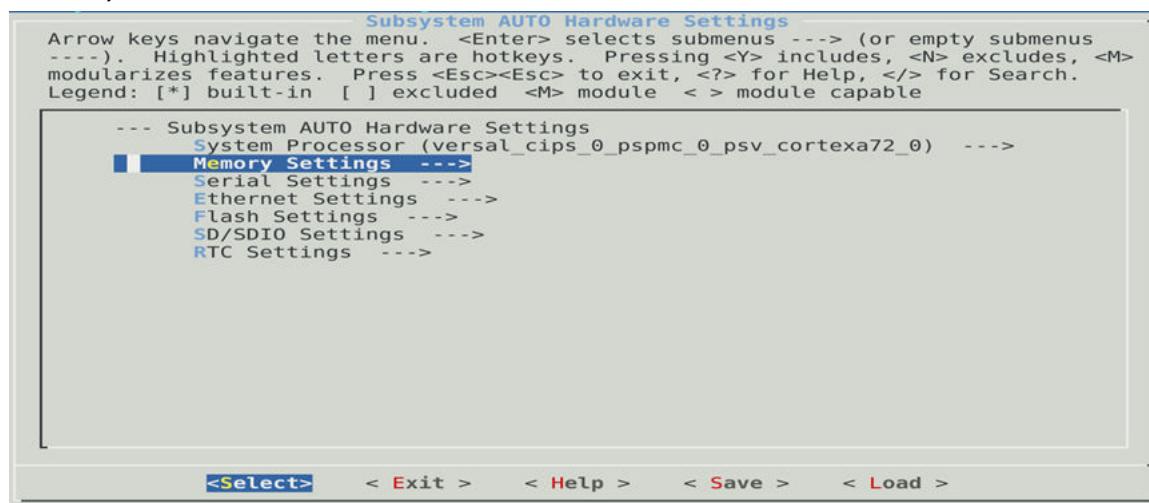
Boot from HBM/Higher DDR/LP DDR

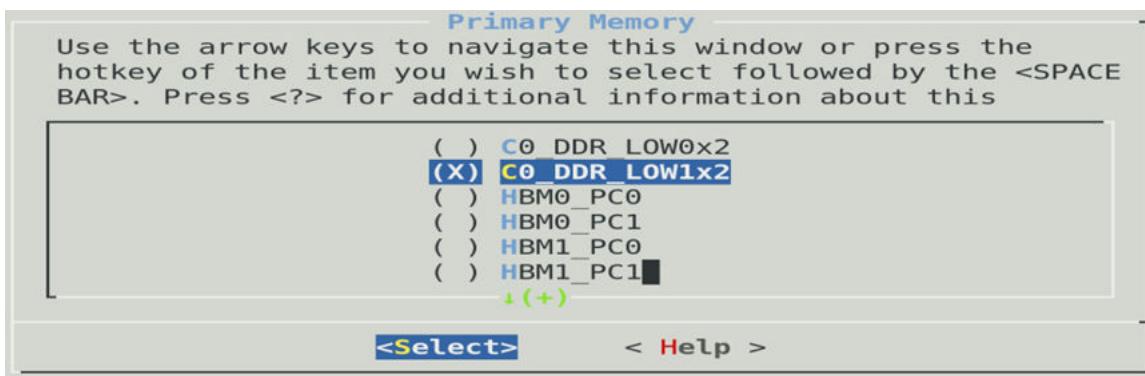
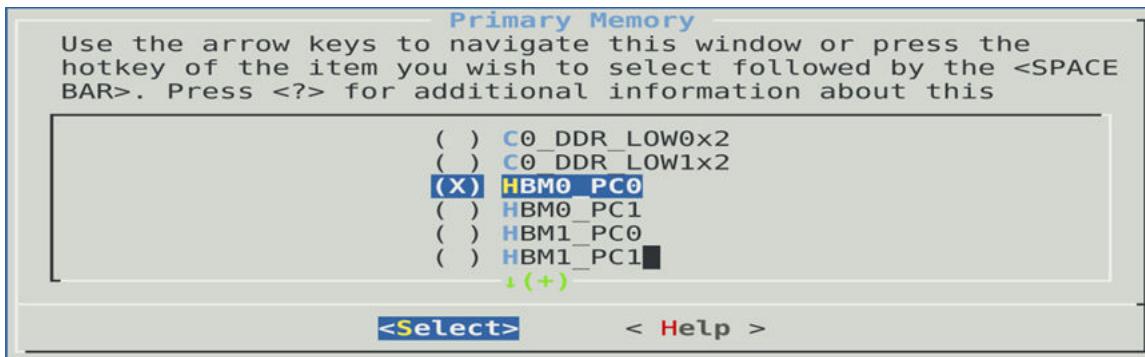
1. Source the latest PetaLinux tool.
2. Create plnx project with Versal template.
3. Point the HBM/higher ddr/lp ddr enabled design xsa file.

```
petalinux-config --get-wh-description <xsa file>
```

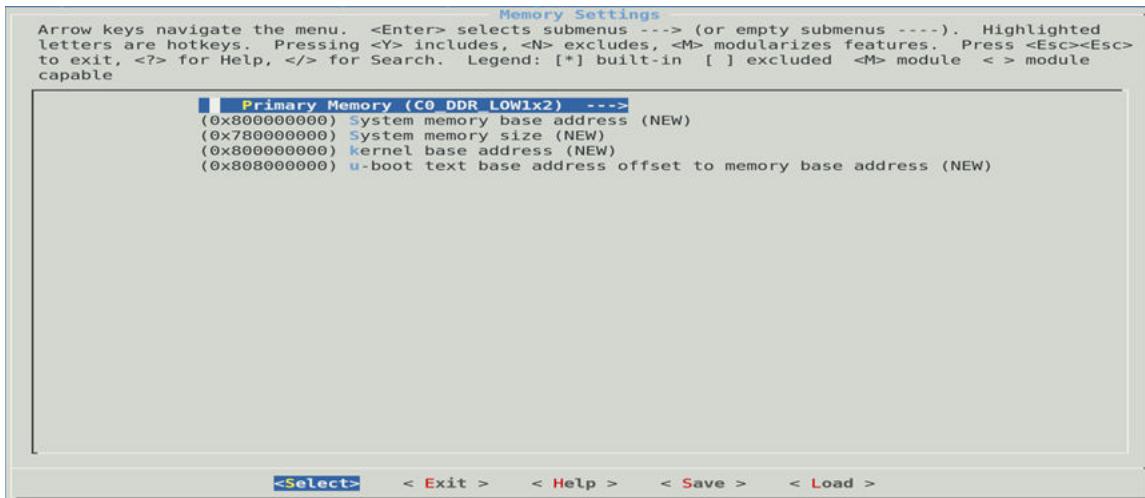
Note: The current VH158 Prod BSP has HBM enabled design.

4. Go to **Subsystem Auto Hardware Settings** → **Memory Settings** → select the memory region of DDR/HBM





5.



6. Save and Exit
7. petalinux-build
8. Boot the images.

The above-built images, by default, boot out of HBM/Higher DDR/LP DDR based on your selection for all boot modes except for jtag and pxe boot.

For jtag, U-Boot is expecting boot.scr at 0x20000000, and this was hardcoded. PetaLinux does not have any U-Boot config option to change this. But U-Boot is providing a mechanism soon to configure the boot script address.

For now, you have the following workaround to work with jtag boot for HBM:

```
petalinux-boot --jtag --kernel --bootscr-addr 0x20000000 --hw_server_url  
<hostname:3121>
```

For PXE boot, still the default, the images boot from Lower DDR. This will be enhanced in future releases.

Upgrading the Workspace

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

Upgrading Between Minor Releases (2023.1 Tool with 2023.X Tool)

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

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

Upgrade PetaLinux Tool

Upgrade from Local File

Download the target system software components content from the [server URL](#).

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

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

Note: Ensure the install area is writable.

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

Example:

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

Upgrade from Remote Server

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

1. Install the tool if you do not have it installed.
Note: The tool should have R/W permissions.
2. Go to installed tool.
3. Type: `source settings.sh`.
4. Enter command: `petalinux-upgrade -u <url>`.

Example:

```
petalinux-upgrade -u "https://petalinux.xilinx.com/sswreleases/rel-v2023/ sdkupdate"
```



IMPORTANT! Only minor version upgrades are supported.

Upgrading only Preferred Platforms in Tool

- To upgrade all platforms:

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

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

- To upgrade only Zynq 7000 platform:

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

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

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

- To upgrade eSDKs for MicroBlaze:

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

Upgrade PetaLinux Project

Upgrade an Existing Project with the Upgraded Tool

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

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

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

Now your project is built with the upgraded tool.

6. If you had used only the `petalinux-config` command in step 4, run the `petalinux-build` command to build the project.

Upgrading the Installed Tool with More Platforms

Once you have installed PetaLinux tool with only the arm platform as specified in [Installing the PetaLinux Tool](#), to install the aarch64 platform, follow these steps.

For the following use case, if you want to upgrade with 2023.2 platforms, use [SDK Update](#).

1. Change directory to the installed tool location.
2. Go to the installed tool.
3. Source `settings.sh` file.
4. Run: `petalinux-upgrade -u http://petalinux.xilinx.com/sswreleases/rel-v2023/sdkupdate/ -p aarch64`

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

Use Cases

- To get the Zynq platform only:

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

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

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

- To get the MicroBlaze platforms:

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

Upgrading the Installed Tool with your Customized Platform

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

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

The tool is upgraded with your new platform changes.

Note: These procedures work only between minor releases.

Customizing the Project

Configuring Firmware Version

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](#).

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`.

Note: Do not use '`_`' in hostname, product name and Firmware version

Configuring Root File System Type

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

Prerequisites

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

Steps for Root File System Type Configuration

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

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

2. Launch the top-level system configuration menu.

```
petalinux-config
```

3. Select **Image Packaging Configuration** → **Root File System Type**.

4. Select **INITRAMFS** or **INITRD** or **JFFS2** or **UBI/UBIFS** or **NFS** or **EXT4 (SD/eMMC/SATA/USB)** as per the requirement.

Note: EXT4 boot functionality expects the root file system to be mounted on ext4 partition and the `BOOT.BIN`, `system.dtb`, and kernel images in FAT32 partition.

Note: Set Image Package Configuration ->Root File System Type as "INITRAMFS/INITRD configuration" with `petalinux-initramfs-image` then the full rootfs should be mounted in ext4 partition to use the `switch_root`.

5. Save Configuration settings.

Configuring U-Boot Boot Script (boot.scr)

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

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

2. Launch the top level system configuration menu.

```
petalinux-config
```

3. Select **u-boot Configuration** → **u-boot script configuration**.

4. In the U-Boot script configuration submenu, you have the following options:

```

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

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

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

```

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

```

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

(0x100000) Devicetree offset
(0x200000) Kernel offset
(0x4000000) Ramdisk image offset
(0x10000000) Fit image offset

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

```

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

Table 16: Default Load Address on JTAG Boot

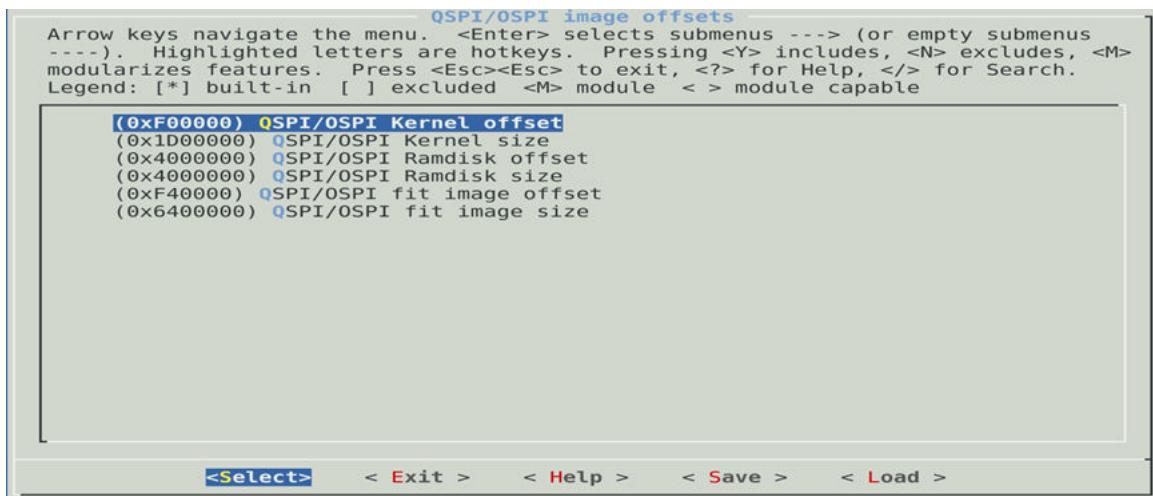
Device	Load Address of boot.scr
Zynq UltraScale+ MPSoC and Versal adaptive SoC	DDR base address + 0x20000000
Zynq 7000 devices	DDR base address + 0x3000000
MicroBlaze processors	(DDR base address + DDR size) - 0xe00000

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

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

Table 17: Default offset on QSPI/OSPI Boot

Device	offset of boot.scr
Zynq UltraScale+ MPSoC	0x3E80000
Versal adaptive SoC	0x7F80000
Zynq 7000 devices	0xFC0000
MicroBlaze processors	0xFC0000



Note: You should check and update offset which is specified in `petalinux-config` → **u-boot Configuration** → **u-boot** script configuration QSPI/OSPI fit image offset should be match with the kernel offset in `petalinux-config` → **Subsystem AUTO Hardware Settings** → **Flash Settings**. If they do not match, the process can fail at U-Boot prompt.

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

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

Table 18: Default Offset on NAND Boot

Device	Offset of boot.scr
Zynq UltraScale+ MPSoC	0x3E80000
Versal adaptive SoC	0x7F80000
Zynq 7000 devices	0xFC0000
MicroBlaze processors	0xFC0000

```

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

(0x4100000) NAND Kernel offset
(0x3200000) NAND Kernel size
(0x7800000) NAND Ramdisk offset
(0x3200000) NAND Ramdisk size
(0x4180000) NAND fit image offset
(0x6400000) NAND fit image size

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

```

Troubleshooting

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

Table 19: Boot Images Storage Troubleshooting

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

Configuring Primary Flash Partition

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.

Configuring INITRD Boot

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

Prerequisites

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

Configuring INITRD Boot

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

Note: Setting `petalinux-initramfs-image` enables the `switch_root` and searches for the rootfs in ext2/3/4 of SD partitions.

4. Build the system image. For more information, see [Building a System Image](#).
5. Use one of the following methods to boot the system image:
 - a. Boot a PetaLinux Image on Hardware with SD Card, see [Booting PetaLinux Image on Hardware with an SD Card](#).

- b. Boot a PetaLinux Image on Hardware with JTAG, see [Booting PetaLinux Image on Hardware with JTAG](#).
 - c. Boot a PetaLinux Image on Hardware with TFTP, see [Booting PetaLinux Image on Hardware with TFTP](#)
-

Configuring INITRAMFS Boot

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

This section describes the procedure to configure INITRAMFS boot.

Prerequisites

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

Configuring INITRAMFS Boot

1. Set the RootFS type to INITRAMFS. For more information, see [Configuring Root File System Type](#).
 2. Set INITRAMFS/INITRD Image name. The default image name is set to `petalinux-initramfs-image` to enable the `switch_root`. The build system generates two types of root file systems: ramdisk images and rootfs images. The ramdisk image is generated as specified in the INITRAMFS/INITRD Image name. It is packed into Fit image (`image.ub`) and Kernel image.
- Note:** Setting `petalinux-initramfs-image` enables the `switch_root` and searches for the rootfs in ext2/3/4 of SD partitions. If `switch_root` is disabled load the final rootfs from RAM.
3. Build the system image. For more information, see [Building a System Image](#).
 4. Use one of the following methods to boot the system image.
 - a. Boot a PetaLinux Image on QEMU, see [Booting PetaLinux Image on QEMU](#).
 - b. Boot a PetaLinux Image on Hardware with SD Card, see [Booting PetaLinux Image on Hardware with an SD Card](#).
 - c. Boot a PetaLinux Image on Hardware with JTAG, see [Booting PetaLinux Image on Hardware with JTAG](#).



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

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

- Image → Image (kernel) + ramdisk.cpio/rootfs.cpio (for Zynq UltraScale+ MPSoC and Versal adaptive SoC)
- ulimage → ulimage (kernel) + ramdisk.cpio/rootfs.cpio (for Zynq 7000 devices)
- linux.bin.ub → simpleImage.mb (kernel) + ramdisk.cpio/rootfs.cpio (for MicroBlaze processors)

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

Configuring TFTP/PXE Boot

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

TFTP/PXE 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 are satisfied:

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

PetaLinux Configuration and Build System Image

Follow these steps to configure PetaLinux for TFTP/PXE boot and build the system image:

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 the TFTP server setup of your host.

5. Save configuration settings and build a system image as explained in [Building a 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 are satisfied:

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

PetaLinux Configuration and Build System Image

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

1. Change to root directory of your PetaLinux project.

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

2. Launch the top level system configuration menu.

```
petalinux-config
```

3. Select **Image Packaging Configuration** → **Root File System Type**.

4. Select **NFS** as the RootFS type.

5. Select **Location of NFS root directory** and set it to `/home/NFSshare`.

6. Exit menuconfig and save configuration settings. The boot arguments in the auto generated DTSI is automatically updated after the build. You can check `<plnx-proj-root>/components/plnx_workspace/device-tree/device-tree/system-conf.dts`.

7. Launch Kernel configuration menu.

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

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

Note: For more information, see [Building a 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 are mentioned as follows:

1. JTAG: In this case, bootloader and kernel is downloaded on to the target through JTAG. For more information, see [Booting PetaLinux Image on Hardware with JTAG](#).

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

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

Configuring JFFS2 Boot

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

Prerequisites

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

Configuring JFFS2 Boot

1. Set the root file system type to JFFS2. For more information, see [Configuring Root File System Type](#).
2. Select **petalinux-config → Image Packaging Configuration → jffs2 erase block size (KB)** (jffs2 erase block size: 128 KB)

Note: 8 KB erase block size does not work with 6.1 kernel because they disabled the dependent kernel configs to support UBIFS boot support.



CAUTION! *The erase block size MUST match the erase size of your flash device*

3. Set Primary Flash as boot device and update the boot script configuration if required. For more information, see [Configuring U-Boot Boot Script \(boot.scr\)](#) and [Configuring Primary Flash Partition](#).
4. Build the system image. For more information, see [Building a System Image](#).
5. Boot a PetaLinux Image on hardware with JTAG, see [Booting PetaLinux Image on Hardware with an SD Card](#).
6. Make sure you have configured TFTP server in host.
7. Set the server IP address to the host IP address using the following command at U-Boot prompt.

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

- a. Detect Flash Memory.

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

- b. Erase Flash Memory.

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

- c. Read images onto Memory and write into Flash.

- Read BOOT.BIN.

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

- Write BOOT.BIN.

```
ZynqMP> sf write 0x80000 0x0 $filesize
```

Example: sf write 0x80000 0x0 0x10EF48

- Read image.ub.

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

- Write image.ub.

```
ZynqMP> sf write 0x80000 <Fit Image Flash Offset Address> $filesize
```

Example: sf write 0x80000 0xF40000 0x6cb0e4

- Read rootfs.jffs2.

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

- Write rootfs.jffs2.

```
ZynqMP> sf write 0x80000 <Rootfs Flash Offset Address> $filesize
```

Example: sf write 0x80000 0x04000000 \$filesize

- Read boot.scr

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

- Write boot.scr

```
ZynqMP> sf write 0x80000 <boot.scr Flash Offset Address> $filesize
```

Example: sf write 0x80000 0x03e80000 \$filesize

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

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

Table 20: Error Message while loading Image and ramdisk.cpio.gz.u-boot

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

Configuring UBIFS Boot

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

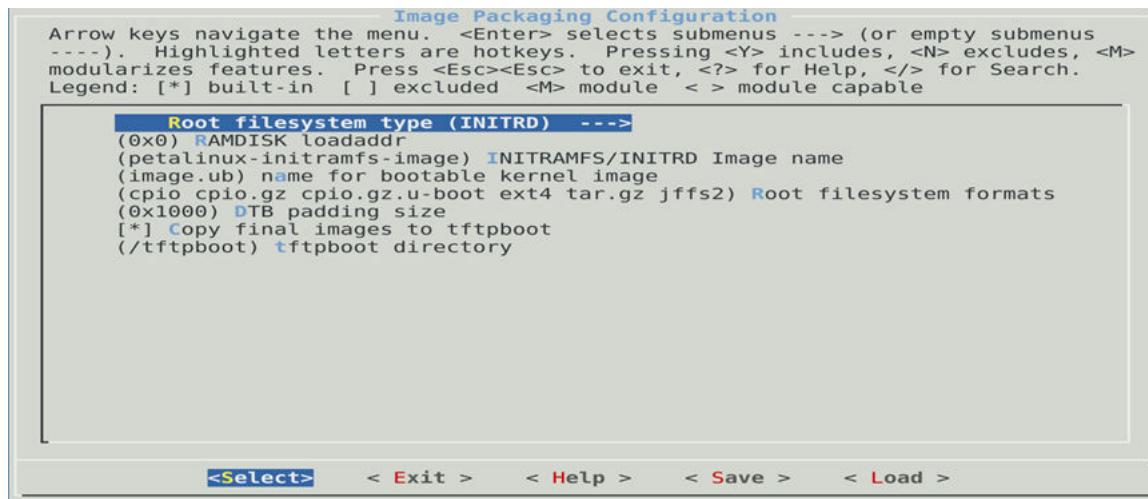
Prerequisites

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

Configuring UBIFS Boot

Follow these steps to configure PetaLinux for UBIFS:

1. Set the root file system to UBIFS. For more information, see [Configuring Root File System Type](#).



- Select **petalinux-config** → **Image Packaging Configuration** → **Root filesystem type (UBI/ UBIFS)**.
 - Specify the arguments used in `mkfs.ubifs` to create the UBIFS partition `-m 2 -e 130944 -c 400` For QSPI 128k erase block size.
 - Specify the arguments used in `ubinize` to create the UBIFS partition. `-m 2048 -p 128KiB -s 512` For QSPI 128k erase block size.
 - Specify the ubifs part name to add in bootargs, for example, `ubifs`.
- Note:** Ensure that you create the QSPI partition with the specified name.
- Specify the root file system format to build `rootfs.ubi` during the build.

For example,

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

2. Set Primary Flash as boot device and configure boot script. For more information, see [Configuring U-Boot Boot Script \(boot.scr\)](#) and [Configuring Primary Flash Partition](#).
3. Build the system image. For more information, see [Building a System Image](#).
4. Boot a PetaLinux Image on Hardware with SD Card. For more information, see [Booting PetaLinux Image on Hardware with an SD Card](#).
5. Ensure that you have configured the TFTP server in the host.

6. Check the offsets for kernel and root file system at **petalinux-config → u-boot Configuration → u-boot script configuration**. If they do not match, the process can fail at the U-Boot prompt.
7. Set the server IP address to the host IP address using the following command at U-Boot prompt.

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

- a. Detect Flash Memory.

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

- b. Erase Flash Memory.

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

- c. Read images onto Memory and write into Flash.

- Read BOOT.BIN.

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

- Write BOOT.BIN.

```
ZynqMP> sf write 0x80000 0x0 $filesize
```

Example: sf write 0x80000 0x0 0x10EF48

- Read image.ub.

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

- Write image.ub.

```
ZynqMP> sf write 0x80000 <Fit Image Flash Offset Address> $filesize
```

Example: sf write 0x80000 0xF40000 0x6cb0e4

- Read rootfs.ubi.

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

- Write rootfs.ubi.

```
ZynqMP> sf write 0x80000 <Rootfs Flash Offset Address> $filesize
```

Example: sf write 0x80000 0x40000000 0x7d4000

- Read boot.scr

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

- Write boot.scr

```
ZynqMP> sf write 0x80000 <boot.scr Flash Offset Address> $filesize
```

Example: sf write 0x80000 0x03e80000 0x80000

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

Calculating the Arguments for `mkfs.ubifs` and `ubinize`

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

Calculating `mkfs.ubifs` Arguments

Table 21: mkfs.ubifs Args

Argument	Description
<code>-m / --min-io-size</code>	The minimum I/O unit. Values are static based on the Flash type. <ul style="list-style-type: none"> • QSPI single and dual stacked value is 1 • QSPI dual parallel value is 2 • OSPI value is 1
<code>-e / --leb-size</code>	The logical erase block size of the UBI volume. <code>leb-size/-e = PEB size - EC - VID = 128 K - 64 B - 64 B = 130944 B</code>
<code>-c / --max-leb-cnt</code>	Specifies maximum file-system size in logical erase blocks <ul style="list-style-type: none"> • PEB(physical erase block) size = 128 KB . Spi datasheet or <code>sf probe 0 0 0</code> command from U-Boot terminal provide this. • EC header for spi flash = 64 B. The link shows the default values for QSPI. <code>max-leb-cnt/-c = QSPI ubifs partition size ÷ (division) leb-size = 50 MB ÷ 130944 B = 400 B</code>

The final expanded values for `mkfs.ubifs` arguments are

```
-m 2 -e 130944 -c 400
```

`mkfs.ubifs` Arguments

Table 22: mkfs.ubifs Args

Argument	Description
<code>-m / --min-io-size</code>	The minimum I/O unit. The following values are static based on the Flash type. <ul style="list-style-type: none"> • QSPI single and dual stacked value is 1 • QSPI dual parallel value is 2 • OSPI value is 1

Table 22: **mkfs.ubifs Args** (cont'd)

Argument	Description
-p / --peb-size	Tells ubinize that the physical erase block size of the flash chip for which the UBI image is created is 128 KiB. -peb-size / -p = 128 KB . The SPI datasheet or command from U-Boot terminal provide this value. sf probe 0 0 0
-s / --sub-page-size	Tells ubinize that the flash supports sub-pages. This should not be more than the min-io-size. --sub-page-size / -s = 1 based on the min-io-size.

The final expanded values for ubinize arguments are

```
-m 2 -p 128KiB -s 1
```

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

Configuring SD Card ext File System Boot

Prerequisites

This section assumes that the following prerequisites are satisfied:

- You have created a new PetaLinux project (see [Creating an Empty Project from a Template](#)) and imported the hardware platform (see [Importing Hardware Configuration](#)).
- It is recommended to use a card with class 6 or higher to achieve optimal file transfer performance.

Preparing the SD Card

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

For more information on how to format and partition the SD card, see [Appendix I: Partitioning and Formatting an SD Card](#).

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

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

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

PetaLinux Configuration and Build System Image

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

1. Change to root directory of your PetaLinux project.

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

2. Launch top level system configuration menu.

```
petalinux-config
```

3. Select **Image Packaging Configuration** → **Root file system type**.

4. Select **EXT4 (SD/eMMC/SATA/USB)** as the root file system type.

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

- eMMC or SD root = `/dev/mmcblk0pX`
- SATA or USB root = `/dev/sdX`

5. Exit menuconfig and save configuration settings.

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

6. Build PetaLinux images. For more information, see [Building a System Image](#).
7. Generate boot image. For more information, see [Generate Boot Image for Zynq UltraScale+ MPSoC](#).
8. The generated `rootfs.tar.gz` file is present in `images/linux` directory.

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 the root directory of your PetaLinux project.

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

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

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

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

```
sudo tar xvfp rootfs.tar.gz -C /media/rootfs
```

In order to boot this SD card ext image, see [Booting PetaLinux Image on Hardware with an SD Card](#). To Create an SD image using PetaLinux use the `petalinux-package --wic` command.

Troubleshooting

Table 23: 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 .

Managing Image Size

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

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

Prerequisites

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

Steps for Managing Image Size

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

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

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

2. Select **File System Packages**.

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

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

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

4. Select **General Setup**.

Under this submenu, 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: Disabling of some `config` items can lead to unsuccessful boot. It is expected that you have the knowledge of `config` items before disabling them.

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

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

- Any U-Boot configuration and environment variables that are added to `bsp.cfg` are included in the U-Boot build.
- Mention the Bootm length in `<plnx-proj-root>/project-spec/meta-user/recipes-bsp/u-boot/files/bsp.cfg`.

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

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

Customizing the Root File System

Including Prebuilt Libraries

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

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

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

Prerequisites

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

For detailed steps to include Prebuilt Libraries refer to [PetaLinux Yocto Tips](#).

Including Prebuilt Applications

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

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

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

Prerequisites

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

Including 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.

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

Prerequisites

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

For detailed Steps to Add Custom Libraries refer to [PetaLinux Yocto Tips](#).

Creating Apps in 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](#).

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

For detailed Steps to Testing User Libraries [PetaLinux Yocto Tips](#).

Creating and Adding Custom Applications

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

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

Prerequisites

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

Adding Custom Applications

The basic steps are as follows:

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

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

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

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

or:

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

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

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

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

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

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

2. Change to the newly created application directory.

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

You should see the following PetaLinux template-generated files:

Table 24: Adding Custom Applications Files

Template	Description
<code><plnx-proj-root>/project-spec/meta-user/conf/user-rootfsconfig</code>	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.
<code>myapp.c</code> for C; <code>myapp.cpp</code> 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/<arch>/myapp/1.0./r0/`

where `arch` can be:

- aarch64(zynqmp and versal) - cortexa72-cortexa53-xilinx-linux
- arm(zynq) - cortexa9t2hf-neon-xilinx-linux-gnueabi/
- microblaze - microblazeel-v11.0-bs-cmp-re-mh-div-xilinx-linux/

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



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

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



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

Creating and Adding Custom Kernel Modules

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

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

Prerequisites

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

Adding Custom Modules

This section explains how to add custom kernel modules:

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

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

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

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

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

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

2. Change to the newly created module directory.

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

You should see the following PetaLinux template-generated files:

Table 25: Adding Custom Module Files

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

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

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

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

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

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



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

Building User Applications

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

Prerequisites

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

Building User Applications

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

- aarch64(zynqmp and versal) - cortexa72-cortexa53-xilinx-linux
- arm(zynq) - cortexa9t2hf-neon-xilinx-linux-gnueabi/
- microblaze - microblazeel-v11.0-bs-cmp-re-mh-div-xilinx-linux/

```
petalinux-build
```

To build `myapp` into an existing system image:

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

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

- To clean the selected user application:

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

- To rebuild the selected user application:

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

This compiles the application. The compiled executable files are in the `<TMPDIR>/work/<arch>/myapp/1.0-r0/` directory where `arch` can be:

- aarch64(zynqmp and versal) - cortexa72-cortexa53-xilinx-linux
- arm(zynq) - cortexa9t2hf-neon-xilinx-linux-gnueabi/
- microblaze - microblazeel-v11.0-bs-cmp-re-mh-div-xilinx-linux/

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`. Refer [Chapter 9: Debugging](#) for more information.
Without this line, the BitBake removes all the build artifacts after building successfully.

- To see all list of tasks for `myapp`:

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

- To install the selected user application:

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

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

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

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

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

Testing User Applications

Prerequisites

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

Testing User Application

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

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

Unless you have changed the location of the user application through its Makefile, the user application is placed into the `/usr/bin` directory.

3. Run your user application on the target system console. For example, to run the user application `myapp`:

```
# myapp
```

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

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

Building User Modules

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

Prerequisites

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

Building User Modules

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

```
petalinux-build
```

To build `mymodule` into an existing system image:

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

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

- To clean the selected user module:

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

- To rebuild the selected user module:

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

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

- To see all list of tasks for this module:

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

- To install the selected user module:

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

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

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

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

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

PetaLinux Auto Login

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

Prerequisites

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

Steps to Enable Serial Auto Login

To enable autologin to root on serial console devices, this option works in conjunction with empty-root-password. When you select this option, it automatically selects the empty-root-password.

Follow these steps to enable the Serial Auto Login feature:

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

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

2. Run `petalinux-config -c rootfs`.
3. Select **Image Features** → **serial-autologin-root**.
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.

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

Prerequisites

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

For the steps on how to auto run application at start up, see [PetaLinux Yocto Tips](#).

Adding Layers

You can add layers to the PetaLinux project. The upstream layers for the Langdale version of Yocto can be found at [+.](#)

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

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

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

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

Note: The 2023.2 release of the PetaLinux tool is on the Langdale baseline. Choose the layers/recipes from the Langdale branch only. Some of the layers/recipes might not be compatible with the current architecture. You are responsible for all additional layers/recipes.

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



IMPORTANT! If you want to change the layer priority, you can update \${PROOT}/project-spec/meta-mylayer/conf/layer.conf to set BBFILE_PRIORITY_meta-mylayer = 7 (0 to 99, higher values have higher priority).

Note: By default, the meta-user layer in the PetaLinux project has a PRIORITY of 7. To override or use the meta-user changes before the custom layer changes, use a PRIORITY level that is equivalent or higher than meta-user. For more information, see [Support](#).

Adding an Existing Recipe into the Root File System

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

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

1. The location of the recipe is <plnx-proj-root>/components/yocto/layers/meta-openembedded/meta-oe/recipes-benchmark/iperf3/iperf3_3.2.bb.
2. Add the following line in <plnx-proj-root>/project-spec/meta-user/conf/user-rootfsconfig.

```
CONFIG_iperf3
```

3. Run `petalinux-config -c rootfs`.
4. Select **user packages** → **iperf3**. Enable it, save, and exit.
5. Run `petalinux-build`.

Note: It is your responsibility to add the recipes in the layers available in PetaLinux tools, apart from PetaLinux default RootFS menuconfig.

Note: The previous procedure is applicable only to the recipes from the existing layers.



IMPORTANT! All recipes which are in `petalinux-image-full` have `sstate` locked. It does not allow any build with modifications and uses the binary from `sstate`. To unlock, ensure to add `SIGGEN_UNLOCKED_RECIPES += "my-recipe"` in `project-spec/meta-user/conf/petalinuxbsp.conf`.

For example, you have changes to be made in `mtd-utils` package, so you have created a `.bbappend` for the same without `SIGGEN_UNLOCKED_RECIPES += "mtd-utils"` in `project-spec/meta-user/conf/petalinuxbsp.conf`. During project build, you should see the following warning. Your changes for the package are not included in the build.

```
"The mtd-utils:do_fetch sig is computed to be  
92c59aa3a7c524ea790282e817080d0a, but the sig is locked to  
9a10549c7af85144d164d9728e8fe23f in SIGGEN_LOCKEDSIGS_t"
```

Adding a Package Group

One of the best approaches for customizing images is to create a custom package group to be used for building the images. Some of the package group recipes are shipped with the PetaLinux tools.

For example:

```
<plnx-proj-root>/components/yocto/layers/meta-petalinux/recipes-core/
packagegroups/packagegroup-petalinux-self-hosted.bb
```

The name of the package group should be unique and should not conflict with the existing recipe names.

You can create custom package group, for example, an ALSA package group would look like:

```
DESCRIPTION = "PetaLinux ALSA supported Packages"

inherit packagegroup

ALSA_PACKAGES = " \
    alsa-lib \
    alsa-plugins \
    alsa-tools \
    alsa-utils \
    alsa-utils-scripts \
    pulseaudio \
"
RDEPENDS:${PN}:append = " \
    ${ALSA_PACKAGES} \
"
```

This can be added to `<plnx-proj-root>/meta-user/recipes-core/packagegroups/packagegroup-petalinux-alsa.bb`.

To add this package group in RootFS menuconfig, add `CONFIG_packagegroup-petalinux-alsa` in `<plnx-proj-root>/project-spec/meta-user/conf/user-rootfsconfig` to reflect in menuconfig.

Launch `petalinux-config -c rootfs`, select **user packages** → **packagegroup-petalinux-alsa**, save, and exit. Run `petalinux-build`.

Appending Root File System Packages

In earlier releases, to add new packages to the root file system, you had to edit the <plnx-proj-root>/project-spec/meta-user/recipes-core/images/petalinux-image-full.bbappend file. For example:

```
IMAGE_INSTALL:append = "opencv"
```

From 2020.1 release onwards, you have to use the <plnx-proj-root>/project-spec/meta-user/conf/user_rootfsconfig file to append new root file system packages to PetaLinux images. For example:

```
CONFIG_opencv
```

Debugging

Debugging the Linux Kernel in QEMU

This section describes how to debug the Linux Kernel inside QEMU using the GNU debugger (GDB). This function is only tested with AMD Zynq™ 7000 devices. For more information, see *Vitis Unified Software Platform Documentation: Embedded Software Development* ([UG1400](#)).

Prerequisites

This section assumes that you have built PetaLinux system image. For more information, see [Building a 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:

```
petalinux-util --gdb vmlinux
GNU gdb (GDB) 12.1
Copyright (C) 2022 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-petalinux-linux --
```

```
target=aarch64-xilinx-linux".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
vmlinux: No such file or directory.
(gdb)
```

5. Attach to the QEMU target in GDB by running the following GDB command:

```
(gdb) target remote :9000
```

6. To let QEMU continue execution:

```
(gdb) continue
```

7. You can use Ctrl+C to interrupt the kernel and get back the GDB prompt.

8. You can set break points and run other GDB commands to debug the kernel.



CAUTION! If another process is using port 9000, *petalinux-boot* attempts to use a different port.

See the output of *petalinux-boot* to determine what port was used. In the following example, port 9001 is used: *INFO: qemu-system-arm ... -gdb tcp::9001 ...*



TIP: It can 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 can experience while debugging the Linux kernel in QEMU.

Table 26: Debugging the Linux Kernel in QEMU Troubleshooting

Problem / Error Message	Description and Solution
(gdb) target remote W.X.Y.Z:9000:9000: Connection refused.	<p>Problem Description: GDB failed to attach the QEMU target. This is most likely because the port 9000 is not the one QEMU is using</p> <p>Solution: Check your QEMU console to ensure QEMU is running. Watch the Linux host command line console. It should show the full QEMU commands and you should be able to see which port is being used by QEMU.</p>

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 AMD Versal™ platform, AMD 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 are satisfied:

- Working knowledge of the AMD Vitis™ software platform. For more information, see *Vitis Unified Software Platform Documentation: Embedded Software Development* ([UG1400](#)).
- The PetaLinux Working Environment is properly set. For more information, see [PetaLinux Working Environment Setup](#).
- You have created a user application and built the system image including the selected user application. For more information, see [Building User Applications](#).

Preparing the Build System for Debugging

1. Change to the project directory:

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

2. Run `petalinux-config -c rootfs` on the command console:

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

3. Scroll down the Linux/RootFS configuration menu to file system packages.

```
admin    --->
audio   --->
base    --->
baseutils  --->
benchmark  --->
bootloader  --->
console   --->
devel    --->
fonts    --->
kernel   --->
libs     --->
misc     --->
multimedia  --->
net      --->
network   --->
optional  --->
power management --->
utils    --->
x11     --->
```

4. Select **misc** submenu:

```
admin    --->
audio    --->
base     --->
baseutils   --->
benchmark   --->
bootloader   --->
console    --->
devel     --->
fonts     --->
kernel    --->
libs      --->
misc      --->
multimedia  --->
net       --->
network   --->
optional   --->
power management --->
utils     --->
x11      --->
```

5. Packages are in alphabetical order. Navigate to the letter 't', as follows:

```
serf    --->
sysfsutils  --->
sysvinit-inittab  --->
tbb     --->
tcf-agent  --->
texi2html  --->
tiff    --->
trace-cmd  --->
util-macros  --->
v4l-utils  --->
```

6. Ensure that tcf-agent is enabled and gdbserver is disabled.

```
[ * ] tcf-agent
[   ] tcf-agent-dev
[   ] tcf-agent-dbg
```

7. Select **console/network** submenu, click **dropbear** submenu. Ensure "dropbear-openssh-sftp-server" is enabled.

```
[ * ] dropbear
```

8. Select **console/network** → **openssh**. Ensure that "openssh-sftp-server" is disabled.

9. Exit the menu.

10. Rebuild the target system image, including hello_linux.elf. For more information, see [Building a System Image](#).

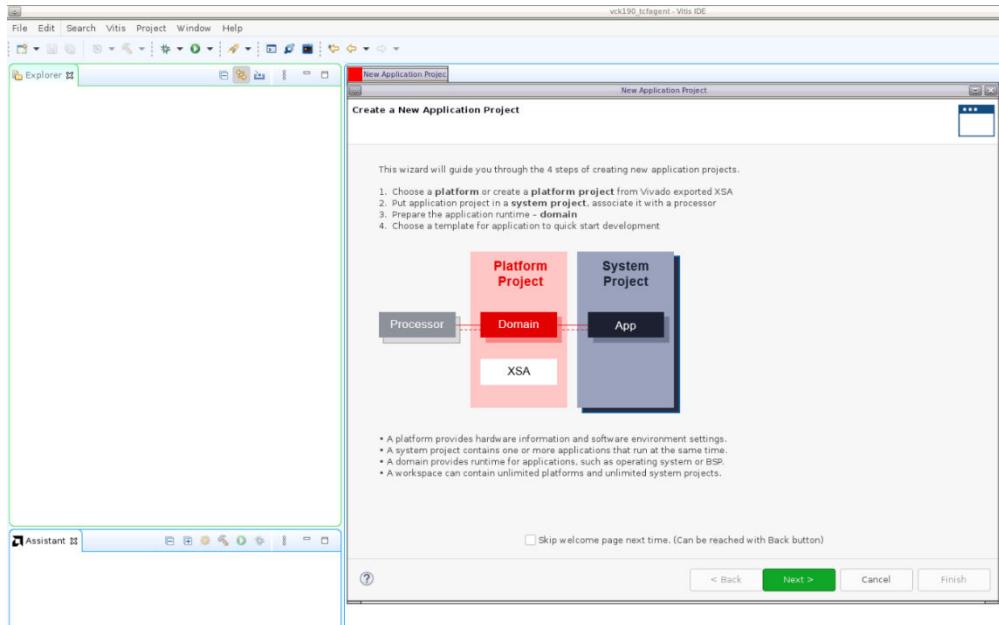
Performing a Debug Session

Creating a Linux Application and Debugging in the Vitis Software Platform

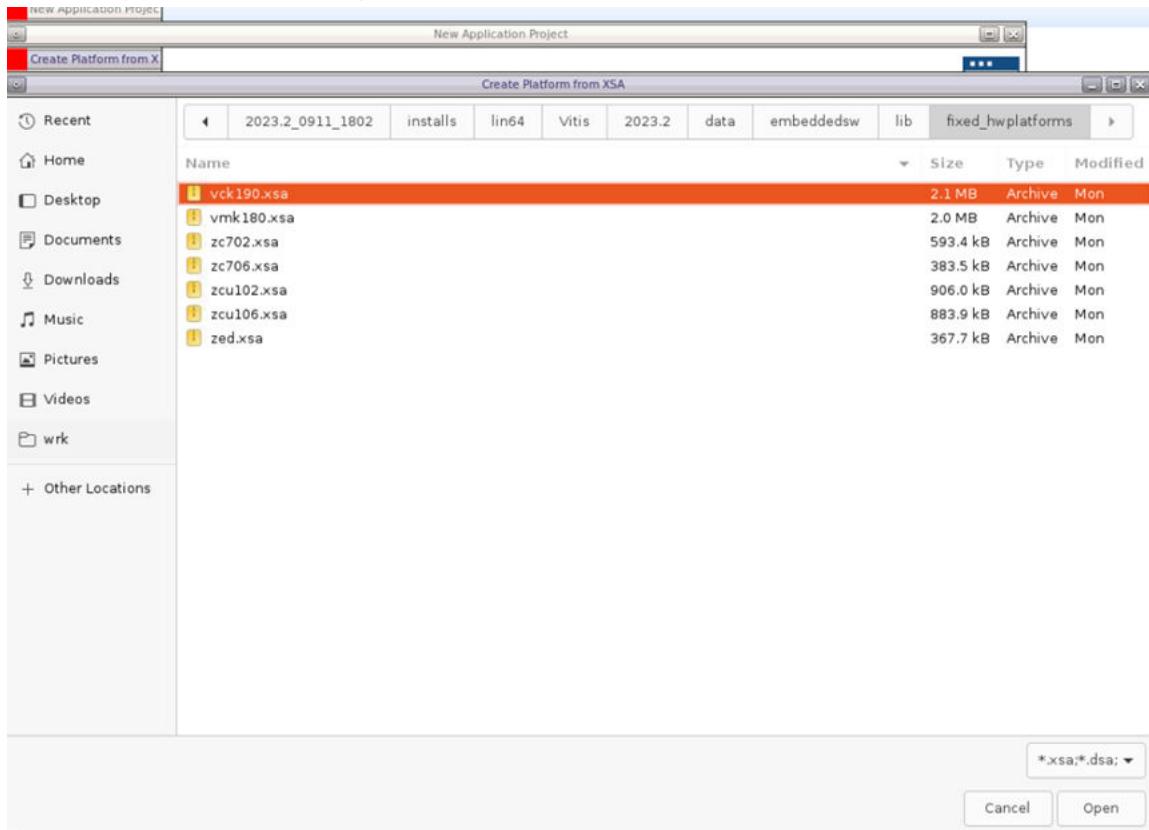
1. Launch the Vitis software platform.

2. Create a Linux application by following these steps:

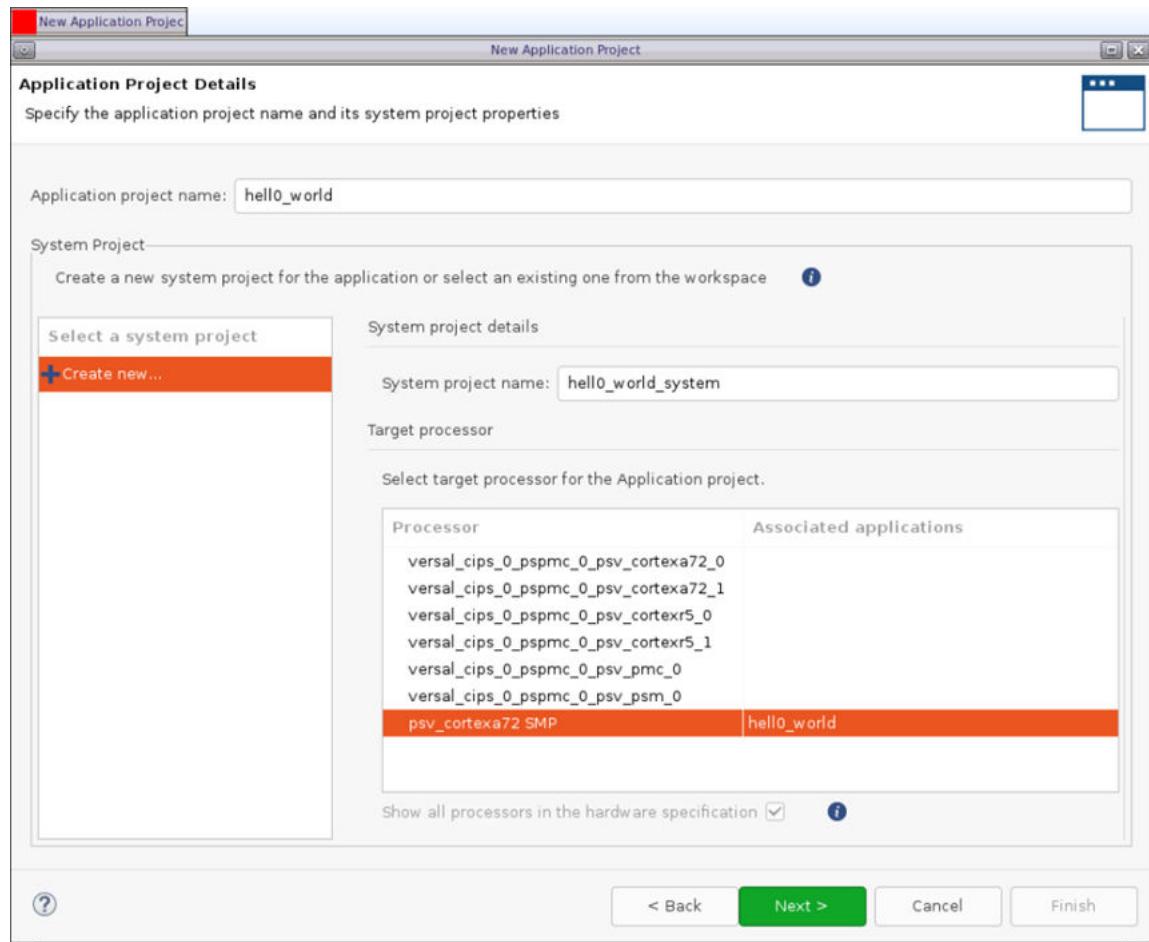
- Select File -> Go to New Application Project.



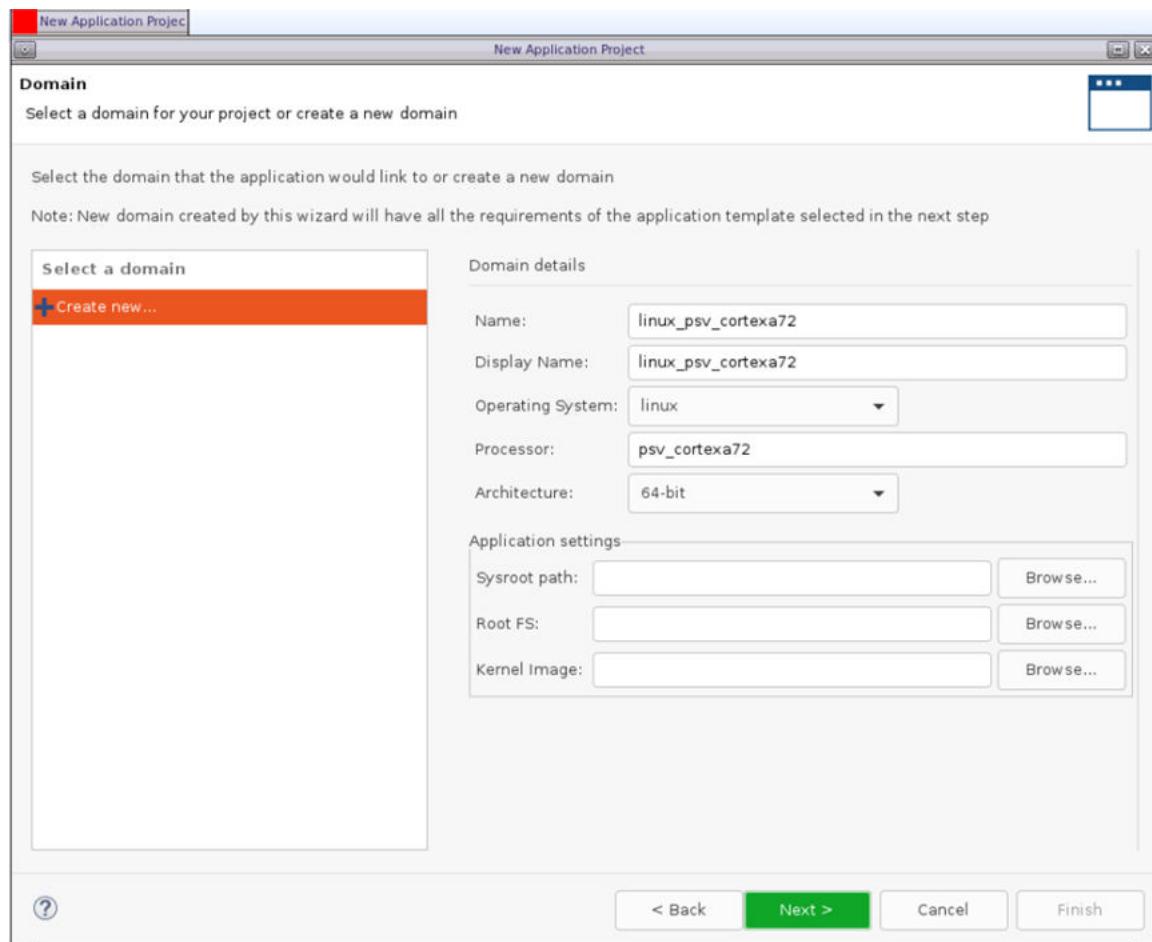
- Create a platform from the provided .xsa file.



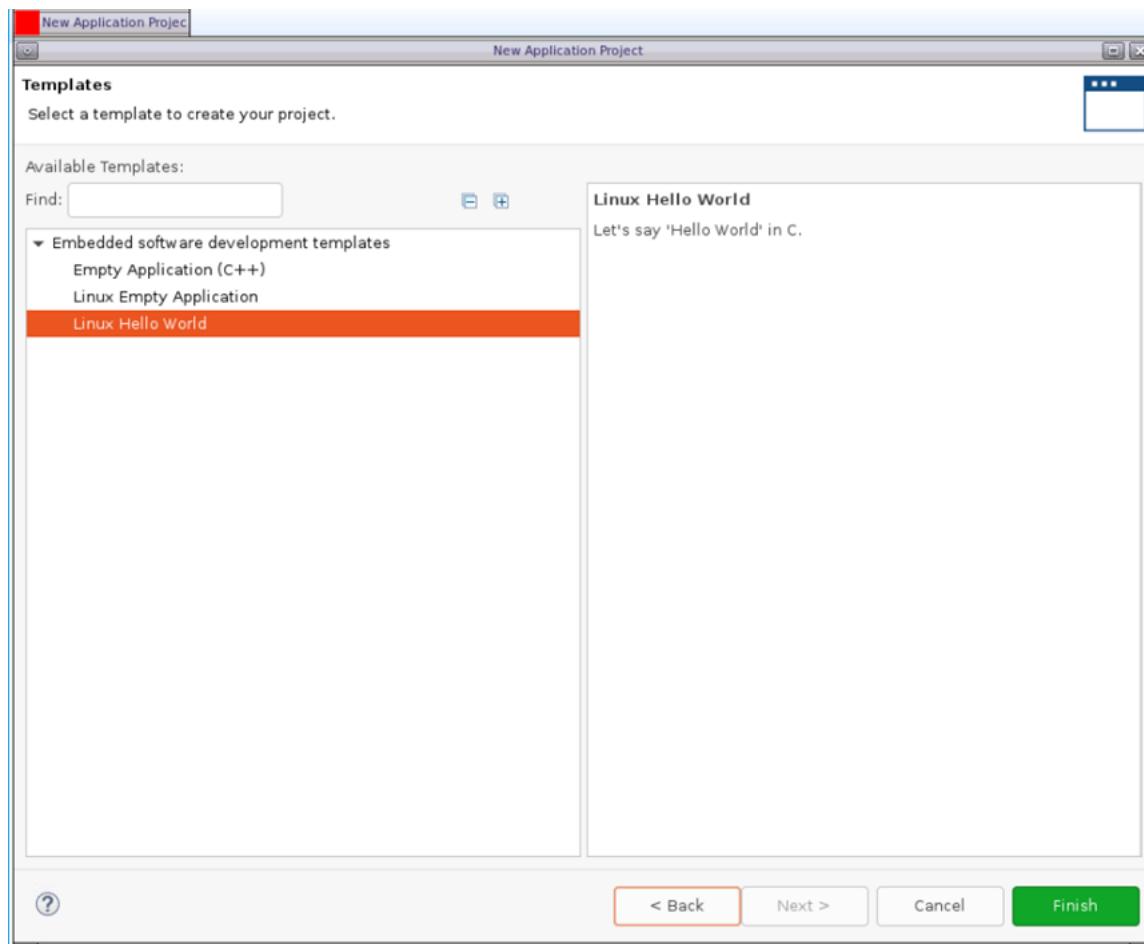
- Choose a name for your application and select the target processor.



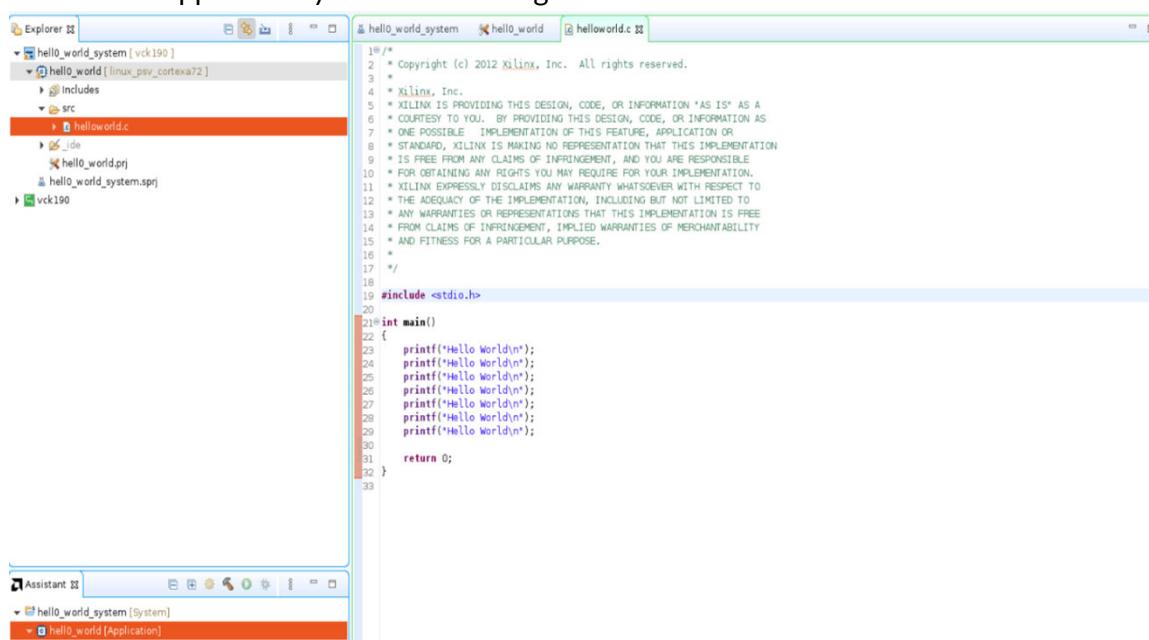
- d. Specify domain details, setting the Operating System to Linux.



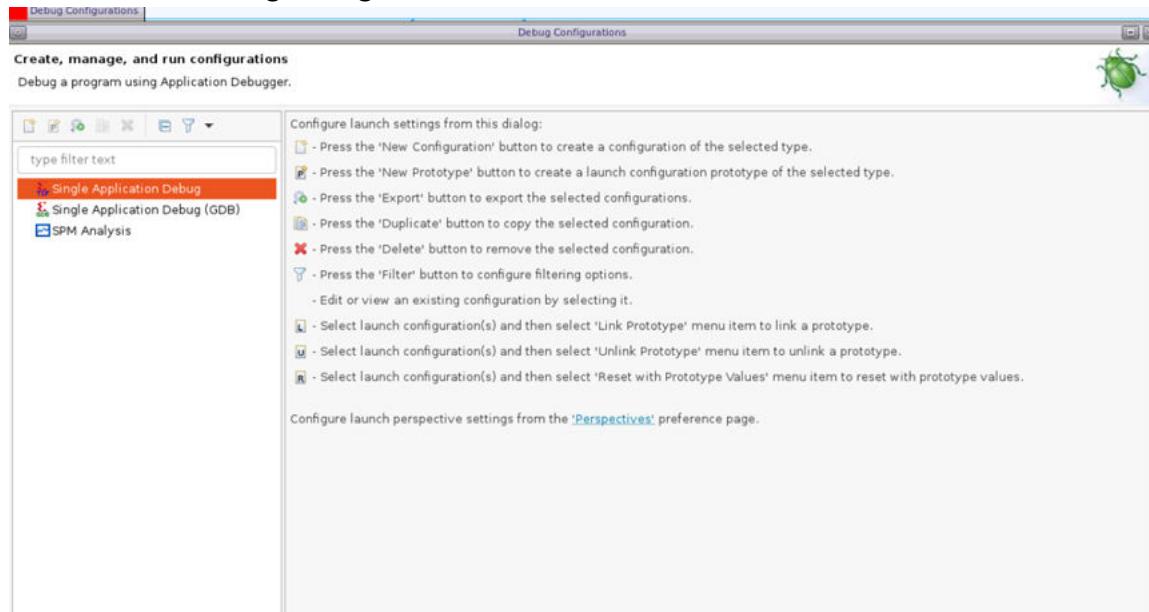
- e. Select an application template (for example, Hello World).



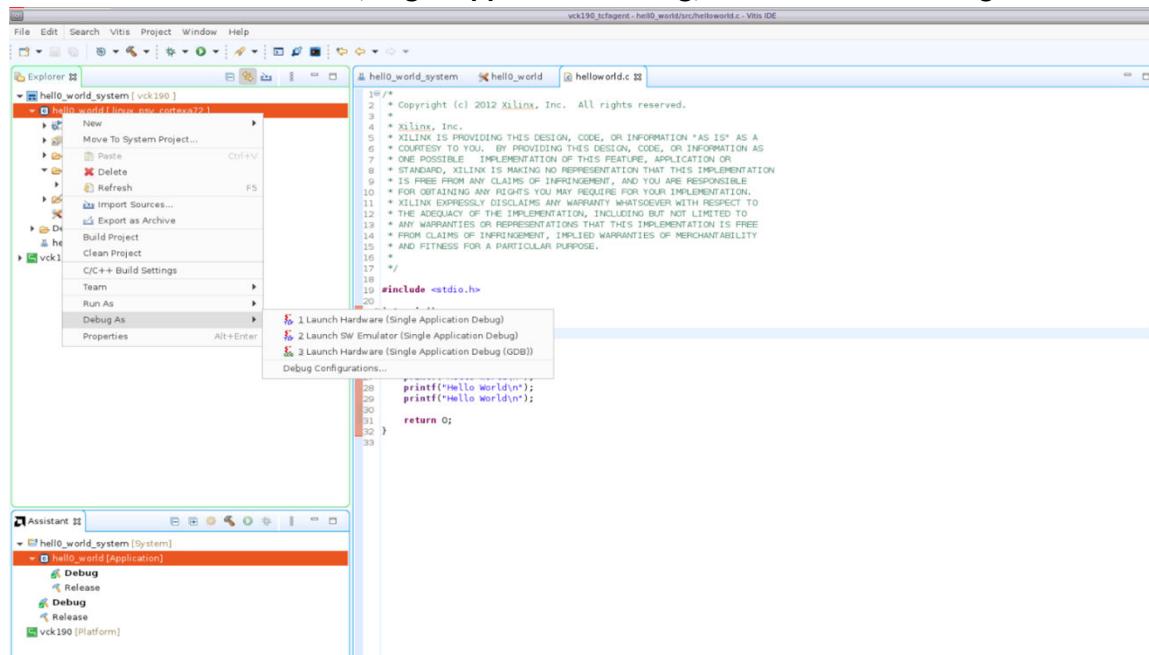
3. Choose the application you wish to debug.



4. Select Run -> Debug Configuration.

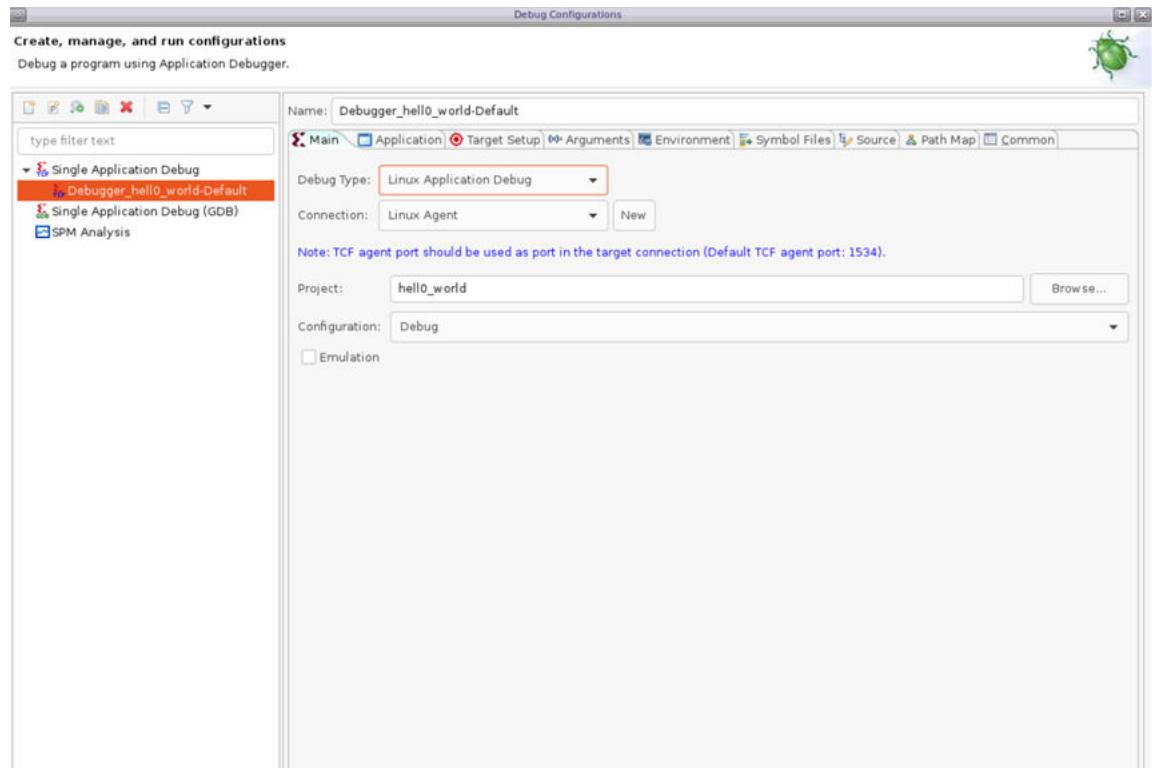


5. Click Launch on Hardware (Single Application Debug) to create a new configuration.

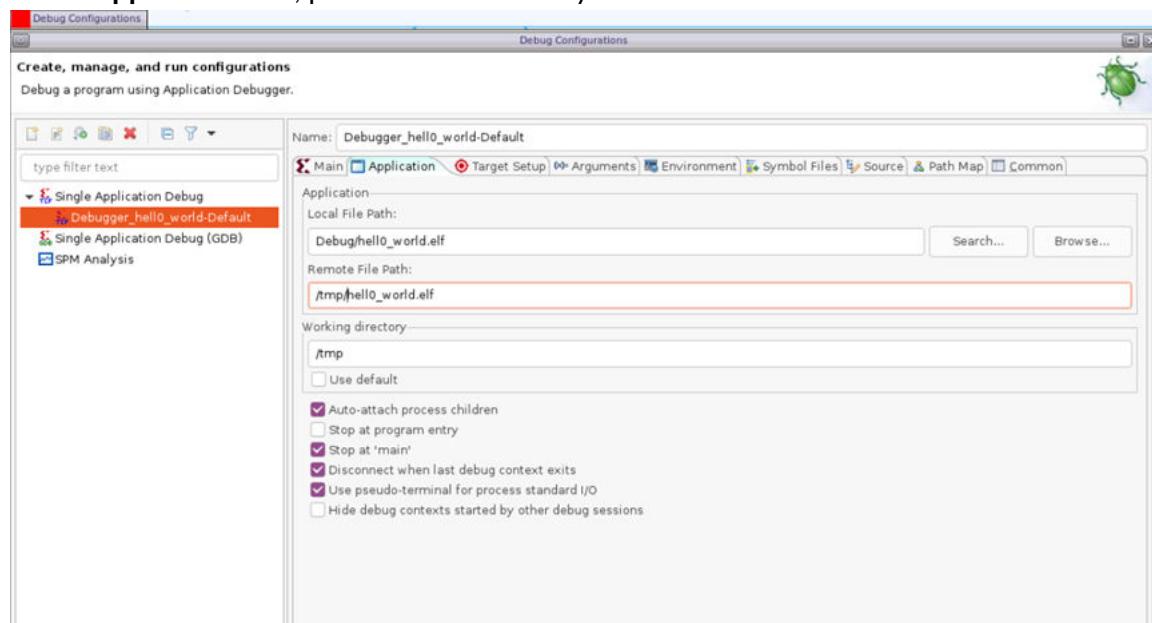


6. In the Debug Configuration window, follow these steps:

- In the **Main Tab**, do the following:

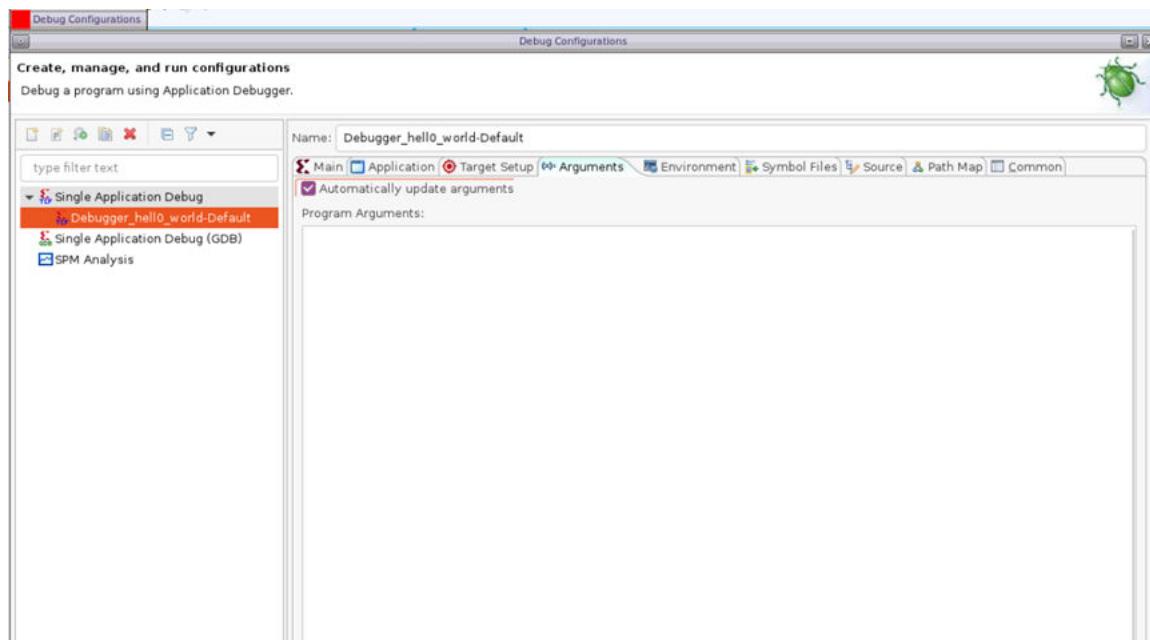


- From the **Debug Type** drop-down list, select **Linux Application Debug**.
 - From the **Connection** drop-down list, choose **Linux agent**.
- b. In the **Application Tab**, provide the necessary information:

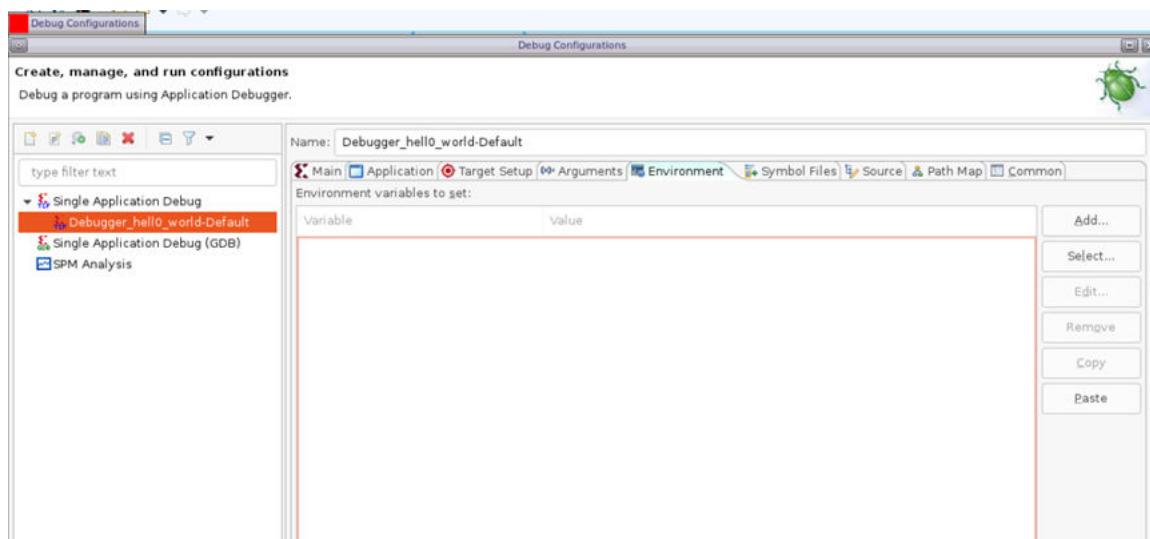


- i. In the **Remote File Path** field, specify the path in Linux where you want to download the application.

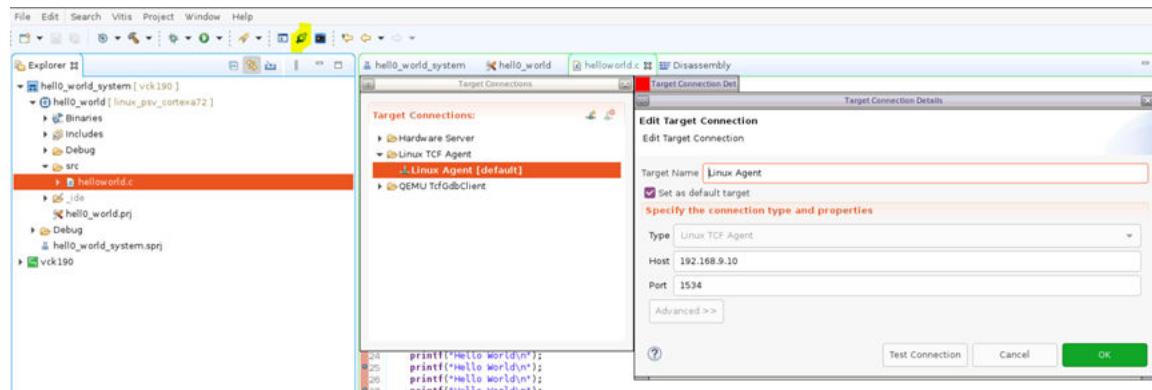
- c. If your application expects any command-line arguments, specify them in the **Arguments** tab.



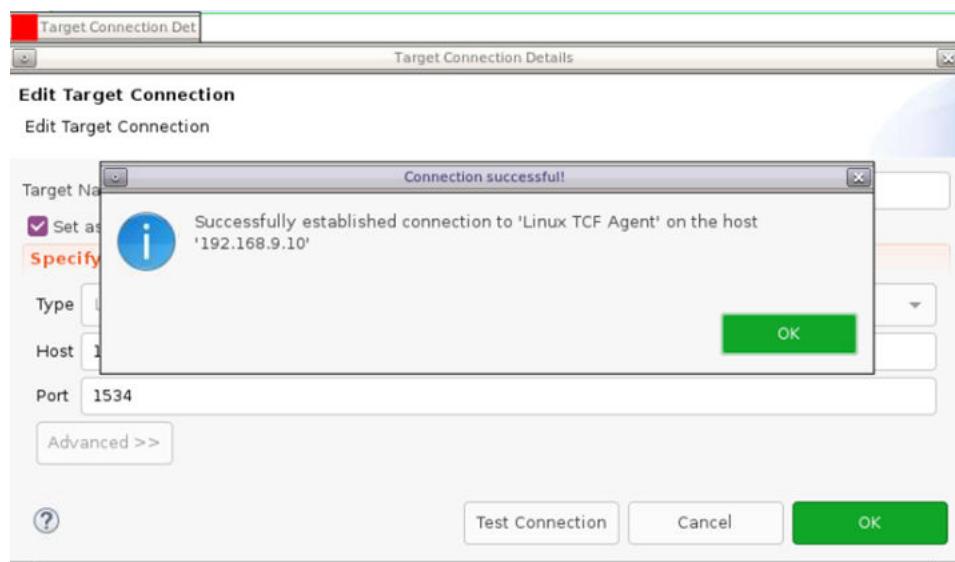
- d. If your application relies on specific environment variables, define them in the **Environment** tab.



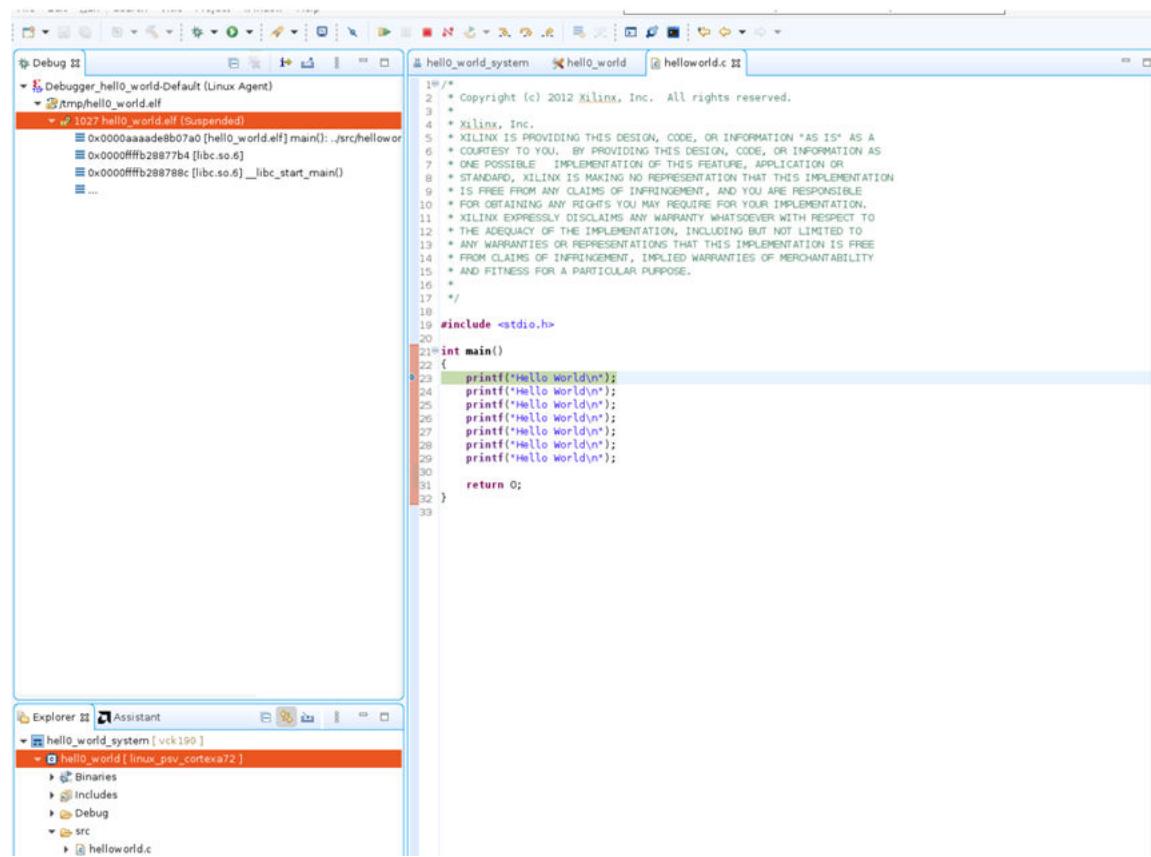
- e. Set up the target connection by entering the Linux hostname or IP address in the **Hostname** field.



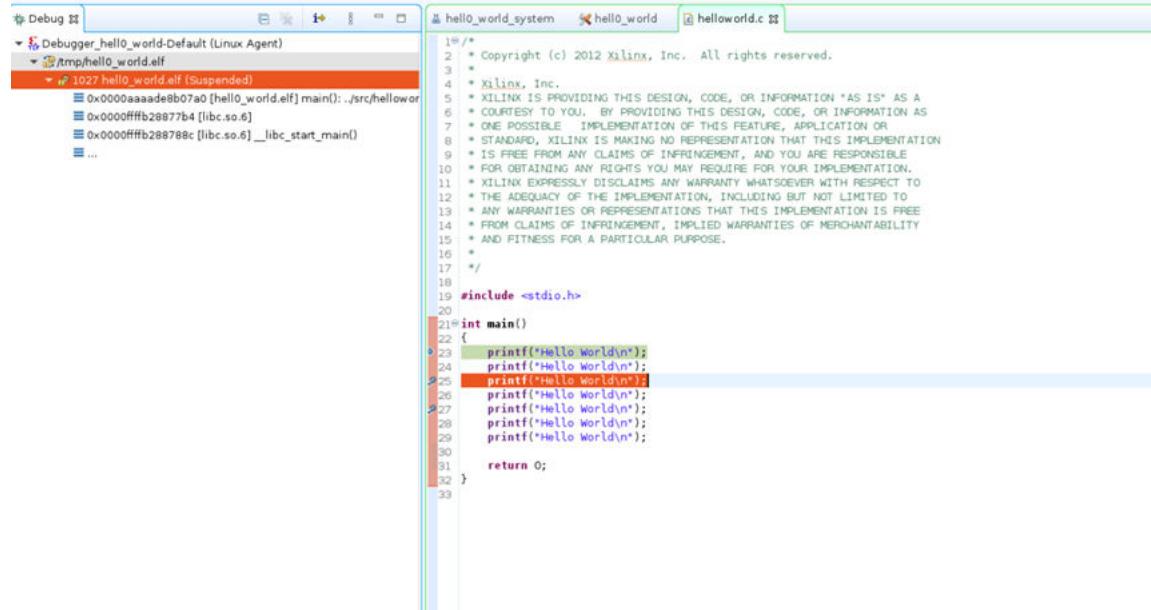
Note: By default, the tcf-agent runs on port 1534 on the Linux machine. If you use a different port, update the **Port** field with the correct port number.



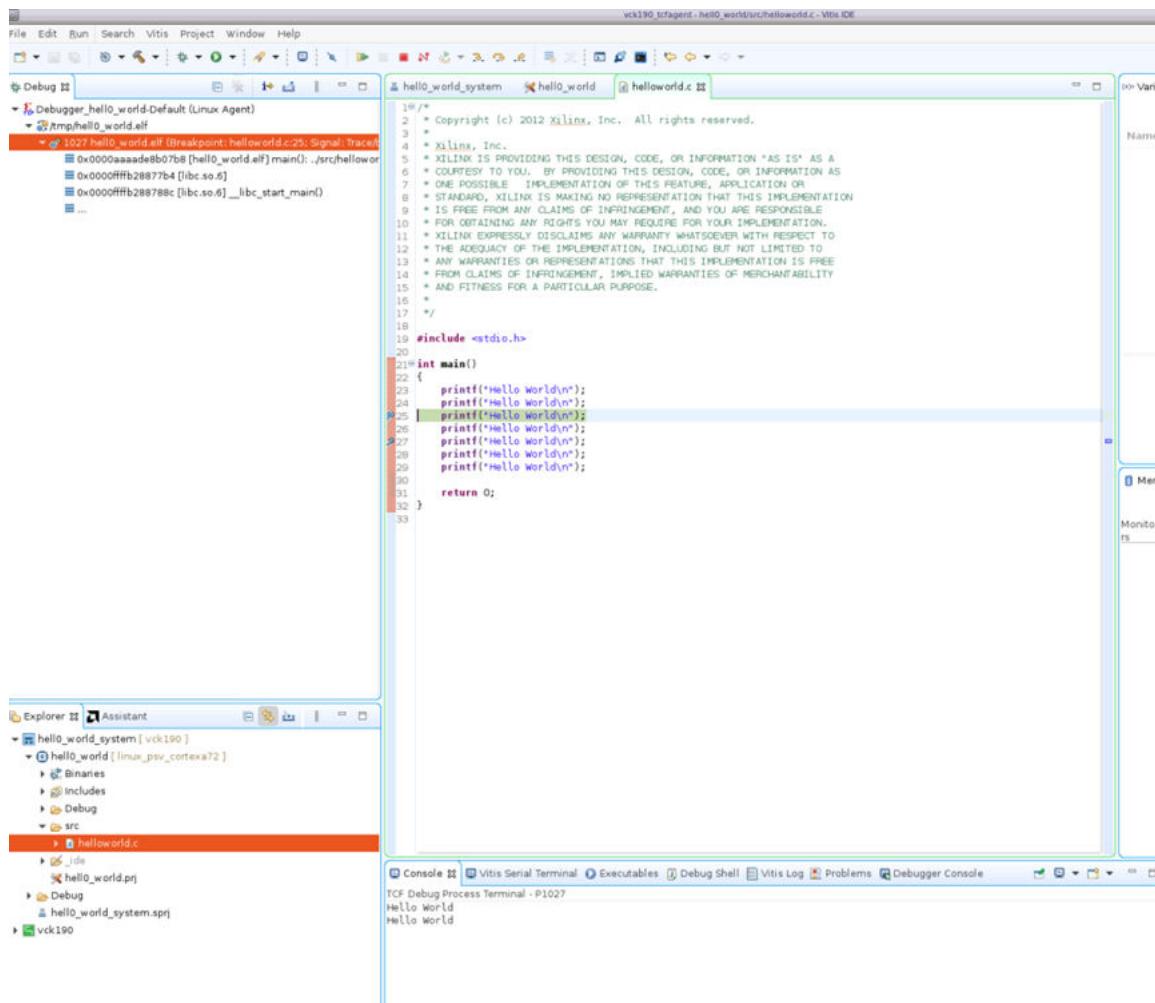
7. Click the **Debug** button. This will open a separate console for processing standard I/O operations.



8. Set breakpoints by double-clicking on the desired lines of code.

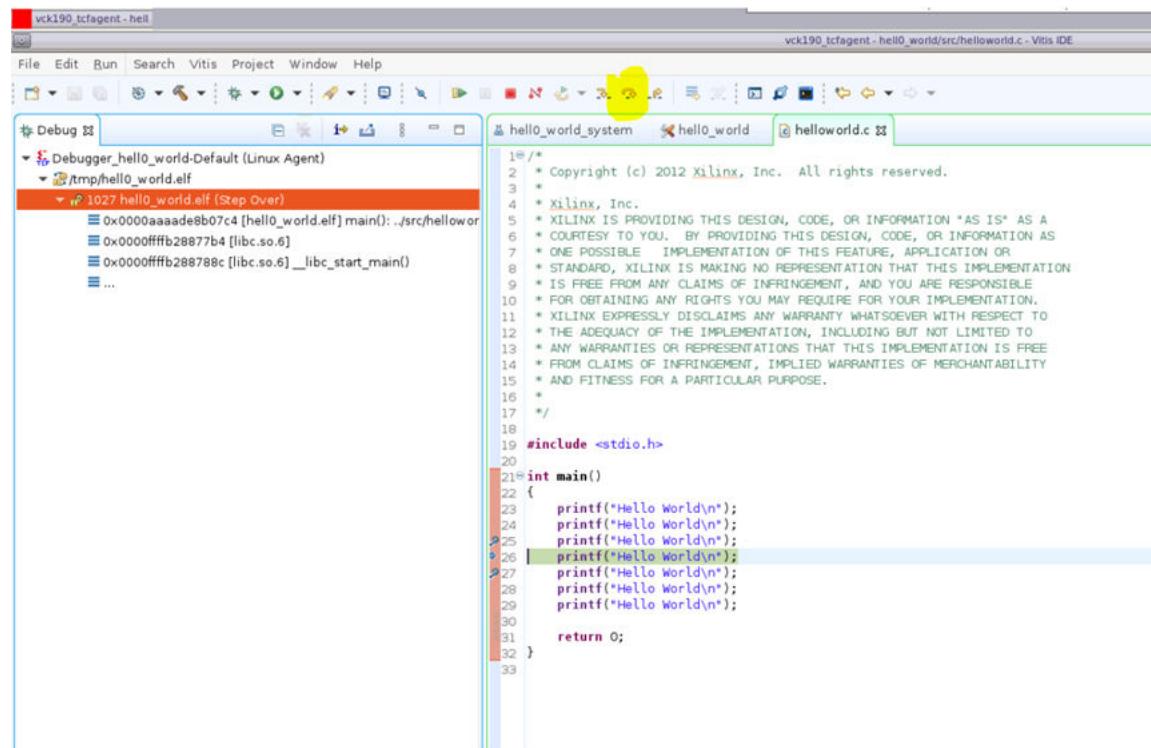


9. To continue execution, click **Resume**.

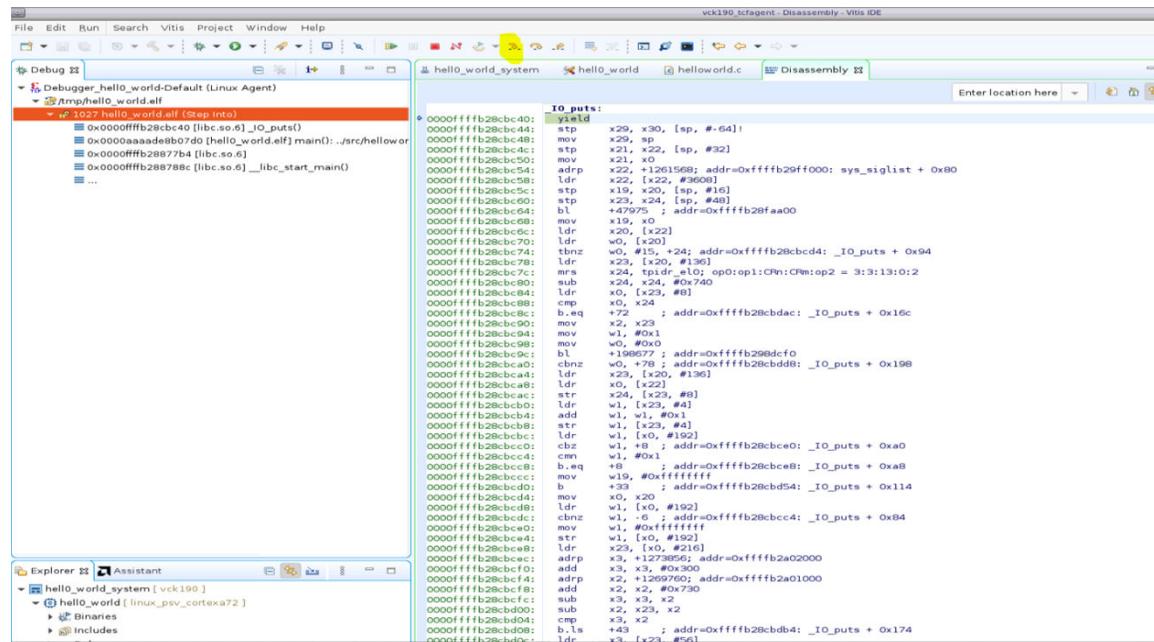


10. For stepping through the code, follow these options:

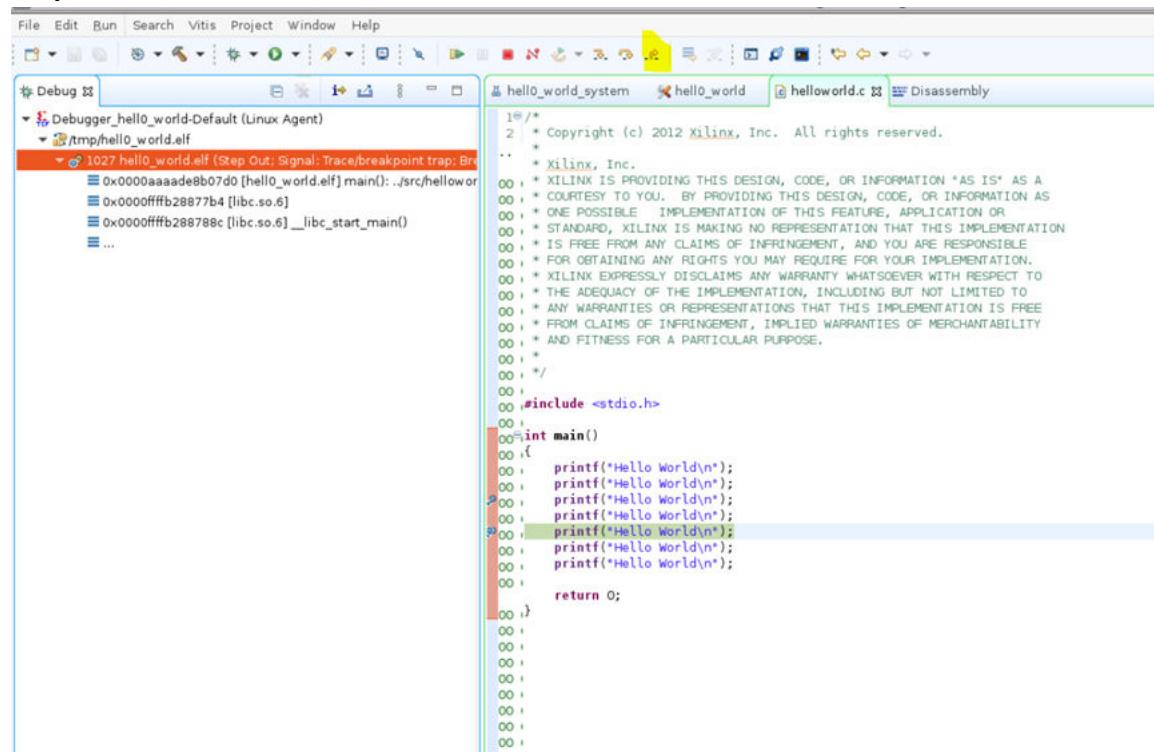
- **Step Over:** Click this to execute the current line and move to the next line in the same function.



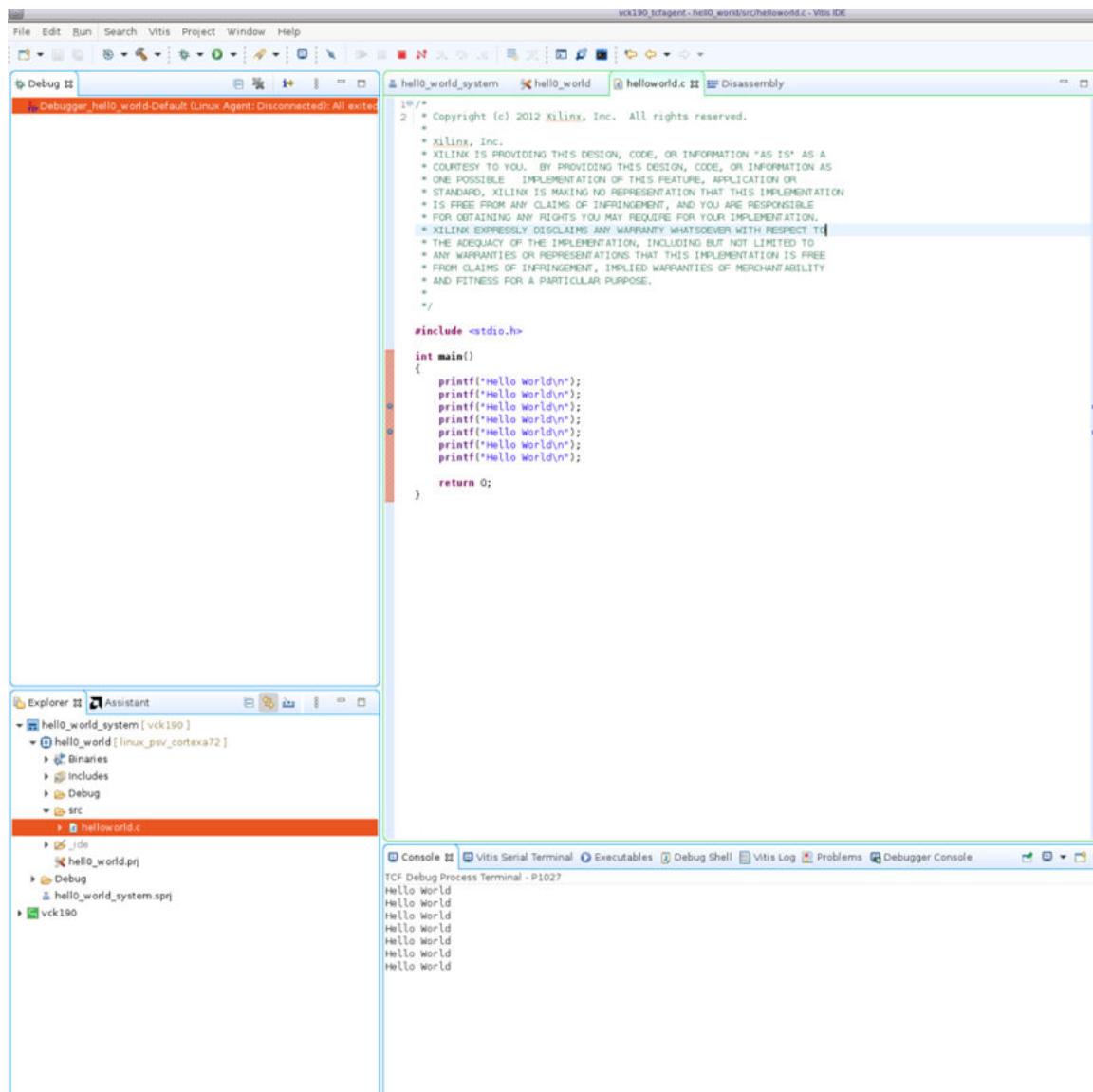
- **Step-In:** Click this to enter into a called function.



- **Step-Rerun:** Click this to re-execute the current line.



11. To continue running the application after stepping through the code, click **Resume** again.



Ensure that you follow these steps carefully to successfully create a Linux application and debug it using the Vitis software platform.

Debugging Zynq UltraScale+ MPSoC and Versal Adaptive SoC Applications with GDB

PetaLinux supports debugging AMD Zynq™ UltraScale+™ MPSoC and AMD Versal™ user applications with GDB. This section describes the basic debugging procedure. For more information, refer to *Vitis Unified Software Platform Documentation: Embedded Software Development* ([UG1400](#)).

Prerequisites

This section assumes that the following prerequisites are satisfied:

- The PetaLinux Working Environment is properly set. For more information, see [PetaLinux Working Environment Setup](#).
- You have created a user application and built the system image including the selected user application. For more information, see [Building User Applications](#).

Preparing the Build System for Debugging

1. Change to the project directory:

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

2. Add the following lines in <plnx-proj-root>/project-spec/meta-user/conf/user-rootfsconfig:

```
CONFIG_myapp-dev  
CONFIG_myapp-dbg
```

3. Add the following lines in <plnx-proj-root>/project-spec/meta-user/recipes-apps/myapp/myapp.bb.

```
DEBUG_FLAGS = "-g3 -O0"  
  
# Specifies to build packages with debugging information  
DEBUG_BUILD = "1"  
  
# Do not remove debug symbols  
INHIBIT_PACKAGE_STRIP = "1"  
  
# OPTIONAL: Do not split debug symbols in a separate file  
INHIBIT_PACKAGE_DEBUG_SPLIT = "1"
```

4. Run petalinux-config -c rootfs on the command console:

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

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

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

6. Select user packages.

```
[X] myapp-dbg  
[ ] myapp-dev
```

7. Select myapp-dbg. Exit the myapp submenu.

8. Exit the user packages submenu, and select **Filesystem Packages** → **misc** → **gdb**.
9. Select **gdb**, and ensure that the GDB server is enabled:

```
[ ] gdb
[ ] gdb-dev
[X] gdbserver
[ ] gdb-dbg
```

10. Exit the menu and select <Yes> to save the configuration.
11. Select **Filesystem Packages** → **misc** → **tcf-agent**.
12. Deselect **tcf-agent** as it uses the same 1534 port. Exit the menu and save the configuration.
13. Rebuild the target system image. Add the following line in <p1nx-proj-root>/project-spec/meta-user/conf/petalinuxbsp.conf.

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

For more information, see [Building a System Image](#).

Performing a Debug Session

To perform a debug session, follow these steps:

1. Boot your board (or QEMU) with the previously created new image.
2. For the QEMU boot to redirect the host port, issue the following qemu-args.

```
petalinux-boot --qemu --kernel --qemu-args "-net nic -net nic -net nic -
net nic,netdev=gem3 -netdev user,id=gem3,hostfwd=tcp::1534-:1534"
```

Note: Based on the gem number, provide the `-net nic` options. You can find the gem number in the <p1nx-proj-root>/components/p1nx_workspace/device-tree/device-tree/`pcw.dtsi` file.

3. Run `gdbserver` with the user application on the target system console (set to listening on port 1534):

```
root@p1nx_aarch64:~# gdbserver :1534 /usr/bin/myapp
Process /bin/myapp created; pid = 73
Listening on port 1534
```

1534 is the `gdbserver` port - it can be any unused port number

4. On the workstation, navigate to the compiled user application's directory:

```
cd <>TMPDIR>/work/cortexa72-cortexa53-xilinx-linux/myapp/1.0-r0/
image/usr/bin/
```

5. Run GDB client.

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

The GDB console starts:

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

6. In the GDB console, connect to the target machine using the command:

- Use the IP address of the target system, for example: 192.168.0.10. If you are not sure about the IP address, run `ifconfig` on the target console to check.
- Use the port 1534. If you select a different GDB server port number in the earlier step, use that value instead.



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

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

The GDB console attaches to the remote target. The GDB server on the target console displays the following confirmation, where the host IP is displayed:

```
Remote Debugging from host 192.168.0.9
```

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

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

8. Run the program by executing the `continue` command in the GDB console. GDB begins the execution of the program.

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

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

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

10. Try the `step`, `next`, and `continue` commands. Breakpoints can be set and removed using the `break` command. More information on the commands can be obtained using the GDB console `help` command.
11. The GDB server application on the target system exits when the program has finished running. Here is an example of messages shown on the console:

```
Hello, PetaLinux World!
cmdline args:
/usr/bin/myapp
Child exited with status 0
GDBserver exiting
root@plnx_aarch64: ~#
```



TIP: A `.gdbinit` file is automatically created to setup paths to libraries. You can 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 can experience while debugging applications with GDB.

Table 27: 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 <code>gdbserver</code> is running on the target system. Check whether there is another GDB client already connected to the GDB server. This can be done by looking at the target console. If you can see Remote Debugging from host <IP>, it means there is another GDB client connecting to the server. Check whether the IP address and the port are correctly set.

Debugging Individual PetaLinux Components

PMU Firmware

<https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841724/PMU+Firmware#PMUFirmware-DebuggingPMUFWusingSDK>

FSBL

<https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842019/FSBL#FSBL-WhatarevariouslevelsofdebugprintsinFSBL>

U-Boot

<https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842557/Debug+U-boot>

Linux

<https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/123011167/Linux+Debug+infrastructure+KProbe+UProbe+LTTng>

<https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/123011146/Linux+Debug+infrastructure+Kernel+debugging+using+KGDB>

Advanced Configurations

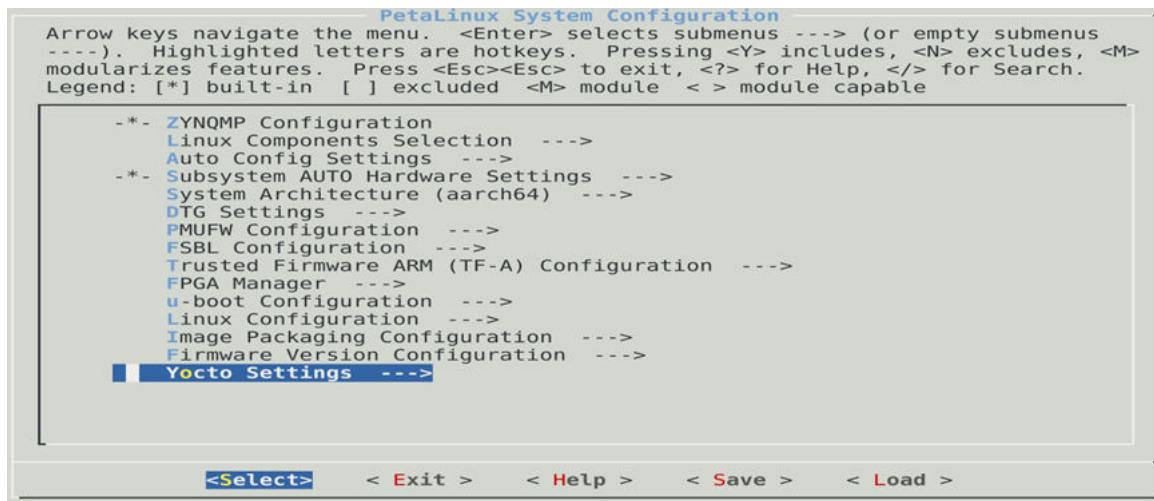
Menuconfig Usage

To select a menu/submenu which was deselected before, press the down arrow key to scroll down the menu or the up arrow key to scroll up the menu. Once the cursor is on the menu, press **y**. To deselect a menu/submenu, follow the same process and press **n** at the end.

PetaLinux Menuconfig System

The PetaLinux menuconfig system configuration for AMD Zynq™ UltraScale+™ MPSoC is shown in the following figure:

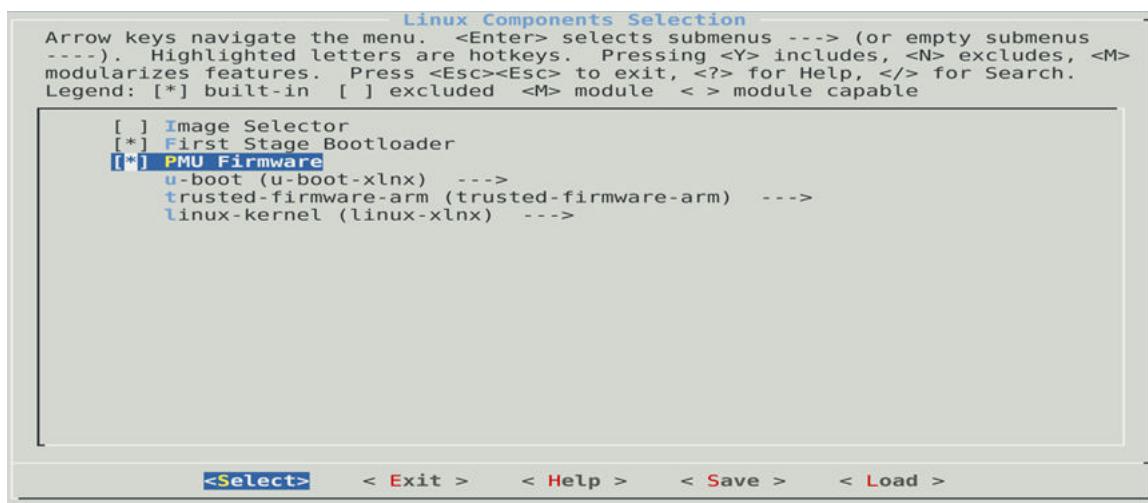
Figure 3: PetaLinux Menuconfig System



Linux Components Selection

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

Figure 4: Linux Components Selection



- Image Selector (Zynq UltraScale+ MPSoC only)
- First stage boot loader
- PMU firmware, for AMD Zynq™ UltraScale+™ MPSoC only
- PLM and PSM firmware for AMD Versal™ platform only
- U-Boot
- Kernel
- TF-A, for Zynq UltraScale+ MPSoC and Versal platforms only

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

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

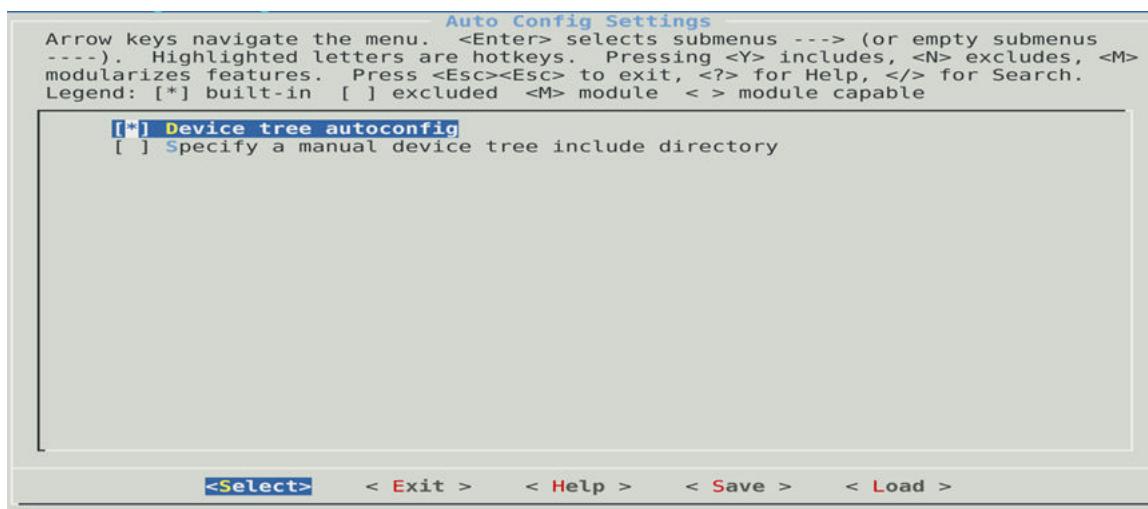
Note: The external source folder is required to be unique to a project and its user, but the content can be modified. If the external source is a git repository, its checked out state should be appropriate for building this project.

- **Remote:** If you want to build a component which was on a custom git repository, this configuration option has to be used.

Auto Config Settings

When a component is selected to enable automatic configuration (autoconfig) in the system-level menuconfig, its configuration files are automatically updated when the `petalinux-config` is run.

Figure 5: Auto Config Settings



Note: The kernel autoconfig and U-Boot auto config options are only available for MicroBlaze processors. They are unavailable for other architectures.

Note: If "Auto config Settings → Device tree autoconfig" is disabled and if you provide any dts files through "Extra dts/dtsi files" and kernel boot args in "DTG settings" menu does not work.

Table 28: Components and their Configuration Files

Component in the Menu	Files Impacted when the Autoconfig is enabled
Device tree	<p>The following files are in <plnx-proj-root>/components/plnx_workspace/device-tree/device-tree/</p> <ul style="list-style-type: none"> • skeleton.dtsi (Zynq 7000 devices only) • zynq-7000.dtsi (Zynq 7000 devices only) • zynqmp-clk-ccf.dtsi (Zynq UltraScale+ MPSoC only) • pcw.dtsi (Zynq 7000 devices, Zynq UltraScale+ MPSoC, and Versal devices) • pl.dtsi • system-conf.dtsi • system-top.dts • <board>.dtsi • versal.dtsi (for Versal devices only) • versal-clk.dtsi (for Versal devices which have clock framework)
kernel (only for MicroBlaze processors)	<p>The following files are in <plnx-proj-root>/project-spec/configs/linux-xlnx/plnx_kernel.cfg</p> <p>Note: plnx_kernel.cfg is generated only when petalinux-config → Auto Config Settings → kernel autoconfig is enabled.</p>

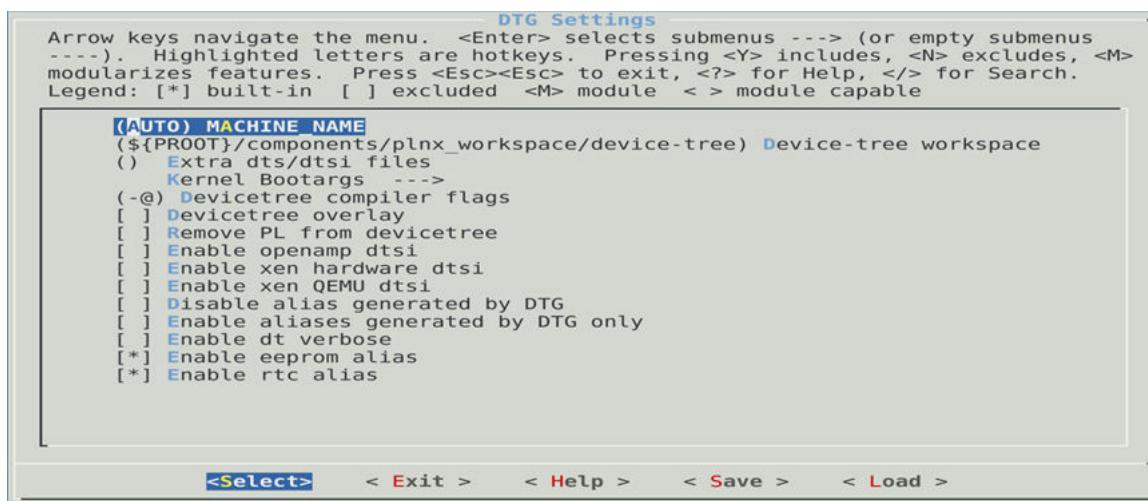
Table 28: Components and their Configuration Files (cont'd)

Component in the Menu	Files Impacted when the Autoconfig is enabled
U-Boot (only for MicroBlaze processors)	<p>The following files are in <plnx-proj-root>/project-spec/configs/u-boot-xlnx/config.cfg</p> <ul style="list-style-type: none"> config.mk (MicroBlaze only) <p>Note: config.cfg are generated only when petalinux-config → Auto Config Settings → u-boot autoconfig is enabled.</p>

Building Device Tree Overlays for PS

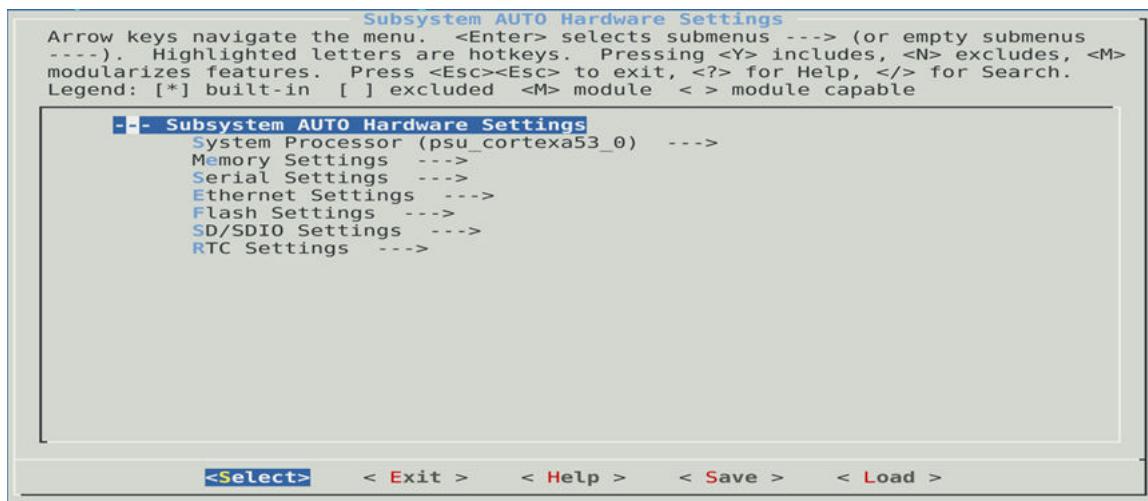
Sometimes you can have the base target with carrier cards. You can initialize the carrier card peripherals as overlays. You can use the following menu config to build the device tree overlays for those carrier cards. You can provide dts/dtsi files with relative/absolute path with space separation. It builds the dtbo files and packs them as part of the root filesystem in /tmp/ <build>.

Figure 6: PS Overlay dtbos



Subsystem AUTO Hardware Settings

The Subsystem AUTO Hardware settings menu allows you to customize how the Linux system interacts with the underlying hardware platform.

Figure 7: Subsystem AUTO Hardware Settings

System Processor

The System processor menu specifies the CPU processor on which the system runs.

Memory Settings

The memory settings menu allows you to:

- Select which memory IP is the primary system memory
- Set the system memory base address
- Set the size of the system memory
- Set the U-Boot text base address offset to a memory high address

The configuration in this menu impacts the memory settings in the device tree and U-Boot automatic configuration (autoconfig) files.

If manual is selected as the primary memory, you are responsible for ensuring proper memory settings for the system.

Serial Settings

The Serial Settings submenu 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 submenu allows you to:

- Select which Ethernet is the systems' primary Ethernet

- Select to randomize MAC address
- Set the MAC address of the selected primary Ethernet. By default, the address is set to ff:ff:ff:ff:ff:ff invalid mac.

If a MAC address is programmed into the EEPROM and you want to use EEPROM mac, due to the previous invalid MAC address, it reads from the EEPROM if there is no ENV MAC from U-Boot. If no MAC address is configured in the EEPROM, the U-Boot generates a random MAC address if mac address is not defined from the U-Boot environment.

The order in which the U-Boot picks the MAC address is as follows. Refer to the U-Boot documentation for commands to program EEPROM and to configure for the same.

- The MAC address from the U-Boot environment has the highest priority.
- The MAC address from the device tree has the next highest priority. Anything configured in the **petalinux-config → Subsystem → Auto Hardware Settings → Ethernet Settings → MAC address** goes to the device tree.
- If there is no MAC address in the device tree or invalid mac address(ff:ff:ff:ff:ff:ff), it checks in EEPROM.
- If there is no MAC address in the EEPROM, it generates a random MAC address.
- Set whether to use DHCP or static IP on the primary Ethernet.

If manual is selected as the primary Ethernet, you are responsible for ensuring proper Ethernet settings for the system.

Netmask address changes for sysvinit and systemd

- Set the static IP from **petalinux-config → Subsystem Auto Hardware Settings → Ethernet Settings**. By default , obtain IP address is enabled. Disable it to set the static IP address. By default Static IP netmask address in 255.255.255.0, Edit/Specify netmask address accordingly.
- In case of sysvinit /etc/network/interfaces file is invoked, it generates with netmask address in dot-decimal notation for example, 255.255.255.0, dot-decimal notation format is supported.

```
address 192.168.0.10  
netmask 255.255.255.0
```

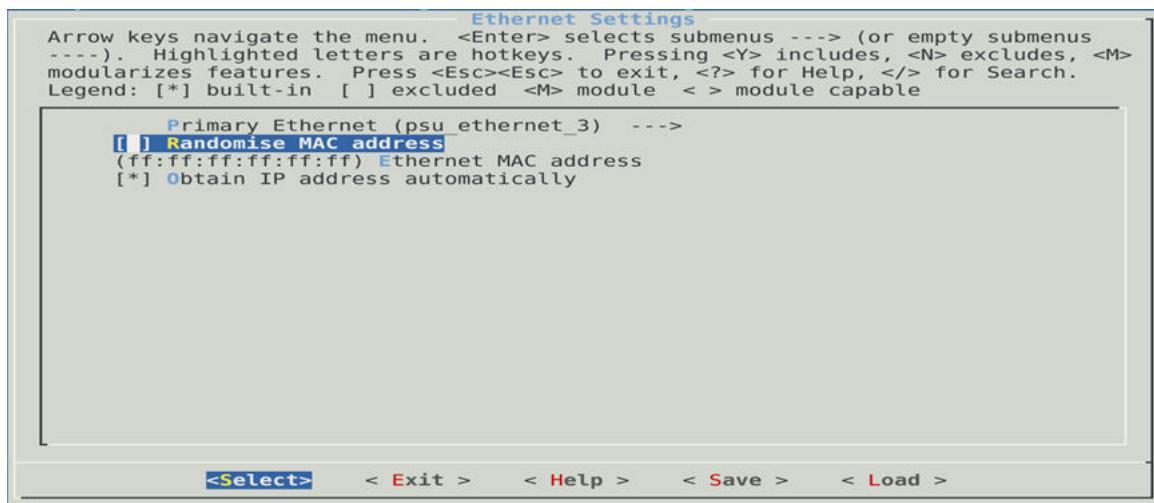
- In case of systemd wired.network file is invoked, it is generated with netmask address in CIDR notation for example, 24 CIDR notation format is supported.

```
Address=192.168.0.10/24
```

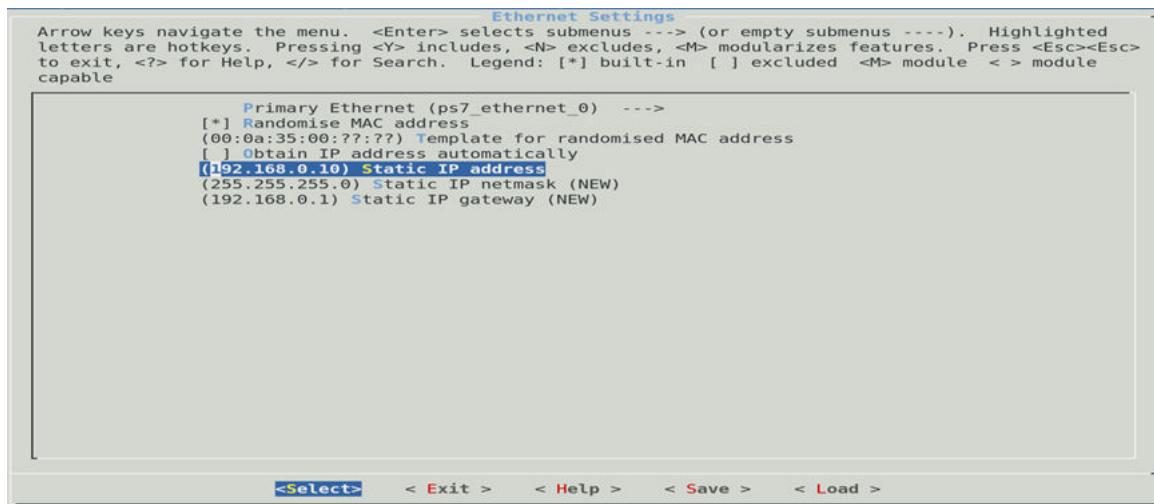
Ethernet submask changes when static IP generation is enabled

Set the static IP from **\$ petalinux-config → Subsystem Auto Hardware Settings → Ethernet Settings**.

Specify the netmask address accordingly.

Figure 8: Obtain IP Address

In the case of sysvinit, /etc/network/interfaces file is invoked, and 255.255.255.0 netmask address is supported.

Figure 9: Static IP netmask

In the case of systemd, specify the address (255.255.255.0) and calculate the bits. Pass netmask as bits (24) to the wired network.

System Architecture Menu

From 2023.1 release, a new menu configuration option to the know system architecture **petalinux-config -> System Architecture** based on the design is created. The system architecture menu is for user information.

Figure 10: System Architecture Menu

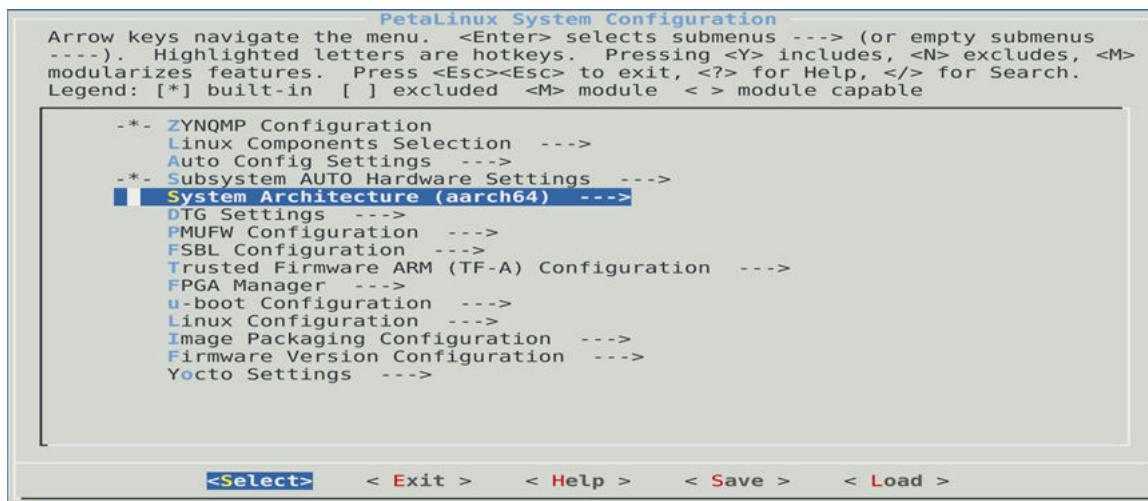
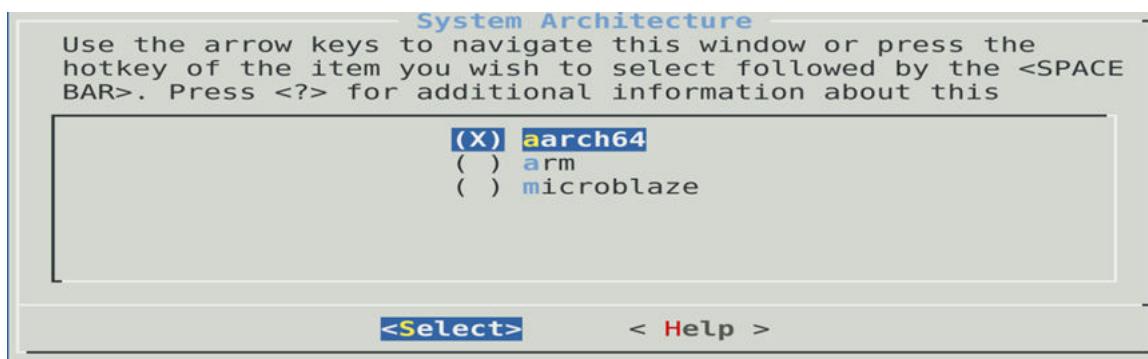


Figure 11: System Architecture



Flash Settings

The Flash Settings submenu allows you to:

- Select a primary flash for the system
- Set the flash partition table

If manual is selected as the primary flash, you are responsible for the flash settings for the system.

SD/SDIO Settings

The SD/SDIO Settings submenu 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 submenu 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 submenu is for MicroBlaze processors only. It allows you to select which GPIO is the system reset GPIO.



TIP: MicroBlaze systems use GPIO as a reset input. If a reset GPIO is selected, you can reboot the system from Linux.

RTC Settings

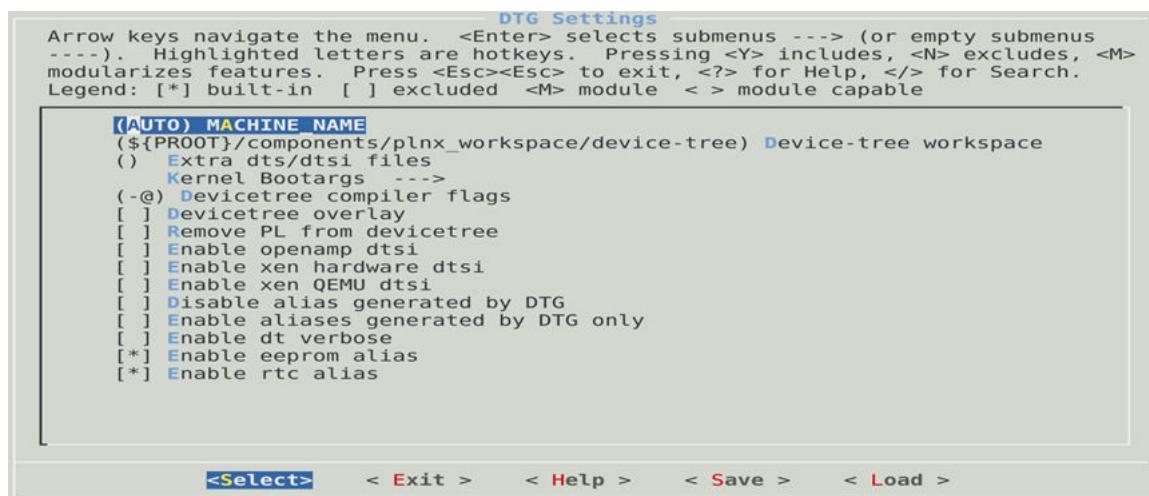
Select an RTC instance that is used as a primary timer for the Linux kernel. If your preferred RTC is not on the list, select **manual** to enable the proper kernel driver for your RTC.

Table 29: Flash Partition Table

Bootable Image/U-Boot Environment Partition	Default Partition Name	Description
Boot Image	boot	BOOT.BIN for Zynq 7000 devices, Zynq UltraScale+ MPSoC, and Versal adaptive SoC. Relocatable U-Boot BIN file (<code>u-boot-s.bin</code>) 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 <code>image.ub</code> (FIT format)
DTB Image	dtb	If Advanced bootable images storage Settings is disabled and a DTB partition is found in the flash partition table settings, PetaLinux configures U-Boot to load the DTB from the partition table. Else, it assumes a DTB is contained in the kernel image.

DTG Settings

Figure 12: DTG Settings



Machine Name

For custom boards, do not change the configuration. For AMD evaluation boards, see [Table 9: BSP and Machine Names](#).

Extra dts/dtsi file

Provide the extra device tree files path is separated with space. This config should be set with dts/dtsi absolute/full path. Provided dts/dtsi is build as part of the device-tree and deployed into <PR0OT>/images/linux/dtbos.

Note: All dtsi files which are included in the dts file should be added to this config. For example, the zynqmp-test.dts file includes the custom.dtsi and main.dtsi files and you must add all three of them separated by spaces to this config.

```
{PR0OT}/project-spec/dts_dir/zynqmp-test.dts  
{PR0OT}/project-spec/dts_dir/custom.dtsi  
{PR0OT}/project-spec/dts_dir/main.dtsi
```

Kernel Bootargs

The Kernel Bootargs submenu 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 -- console=ttyS0,115200 earlyprintk
zyng          -- console=ttyPS0,115200 earlyprintk
zynqmp        -- console=ttyPS0,115200 earlycon clk_ignore_unused
root=/dev/ram0 rw
versal         -- console=ttyAMA0earlycon=p1011,mmio32,0xFF000000,115200n8
clk_ignore_unused
```

Note: If you want to see kernel panic prints on the console for Zynq UltraScale+ MPSoCs and Versal devices, add `earlycon console=<device>,<baud rate> clk_ignore_unused root=/dev/ram rw`. For example, `earlycon console=/dev/ttys0,115200 clk_ignore_unused root=/dev/ram rw` is added in the `system_user.dtsi`.

For more information, see kernel documentation.

Device Tree Overlay Configuration for Zynq 7000 Devices, and Zynq UltraScale+ MPSoCs

If you want to load PL after Linux boot, 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 steps to add overlay support:

1. Go to `cd <proj root directory>`.
2. In the `petalinux-config` command, select **DTG Settings → Device tree overlay**.
3. Run `petalinux-build` to generate the `pl.dtbo` in `images/linux` directory.

FPGA manager overrides all options of the device tree overlay. Device Tree Overlay comes into play only when the 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: system.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`).

3. Once the target is up, load the dtbo and bit.bin file.

```
fpgautil -b design_1_wrapper.bit.bin -o pl.dtbo
```

Removing PL from the Device Tree

Select this configuration option to skip PL nodes if the user does not depend on the PL IPs. Also, if any PL IP in DTG generates an error, you can simply enable this flag to ensure that the DTG does 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`.

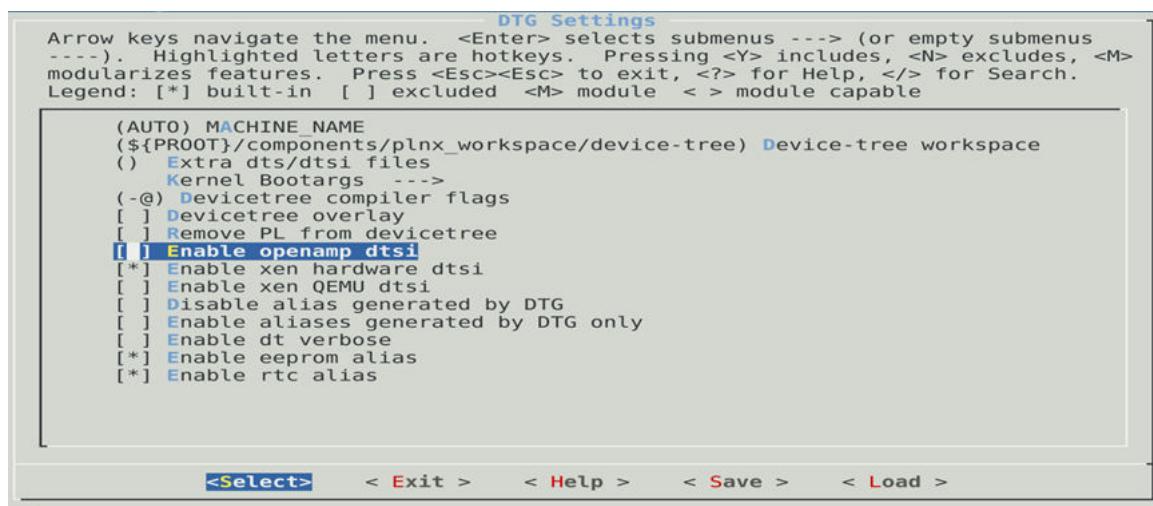
Note: FPGA manager overrides all the Remove PL options. Remove PL option comes into play only when the FPGA manager is not selected.

Note: If you select both the device tree overlay and remove the PL from the device tree, the base DTB adds an entry for overlay support but there is no `pl.dtbo` generated.

Enable openamp dtsi

Earlier, `openamp.dtsi` file was in `<proj>/project-spec/meta-user/device-tree/files/openamp.dtsi` and it was included in `system-user.dtsi` as a part of the `system.dtb`.

Figure 13: Enable openamp dtsi



From 2023.1 release, the dtsi files are moved to Yocto layers **petalinux-config** → **DTG Settings** → **Enable Openamp Dtsi** configuration, it can be added to `system.dtb` or built as an overlay by applying `openamp dtsi` nodes as an overlay.

Build openamp dtb in PetaLinux

There are two ways to build openamp dtb:

- **Building openamp dtb within system.dtb:** Enable the **petalinux-config → DTG Settings -> Enable openamp dtsi** configuration and add the openamp packages from `petalinux-config -c rootfs` with in `system.dtb`

Select the following packages and save:

- `openamp-fw-echo-testd`
- `openamp-fw-mat-muld`
- `openamp-fw-rpc-demo`
- `packagegroup-petalinux-openamp`

Note: The previous configuration is enabled for all in Zynq, Zynq UltraScale+ MPSoC, and Versal prebuilt images.

- **Applying opemamp dtsi nodes as an overlay:**

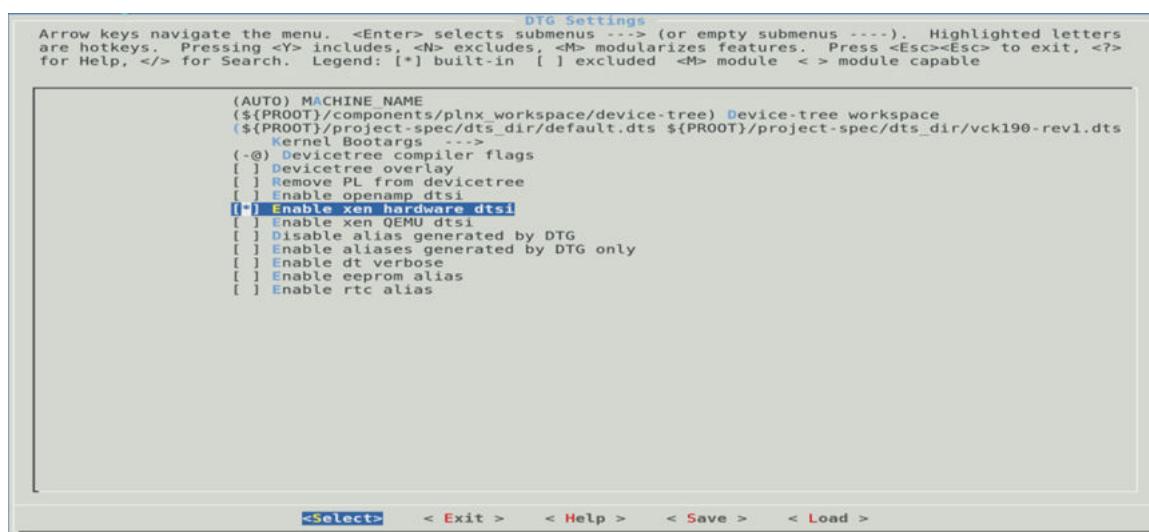
Disable the **petalinux-config -> DTG Settings -> Enable Openamp dtsi** configuration and enable the openamp package group through `petalinux-config -c rootfs`. It builds the `openamp.dtbo` as part of rootfs and it is available in `/boot/devicetree`.

Bootscript(`boot.scr`) auto-loads the `openamp.dtbo` if it is available in `/boot/devicetree`.

Enable xen hardware dtsi

Select `Enable xen hardware dtsi` configuration in **petalinux-config → DTG Settings** to build the xen hardware dtsi changes as part of `system.dtb`.

Figure 14: Enable xen hardware dtsi



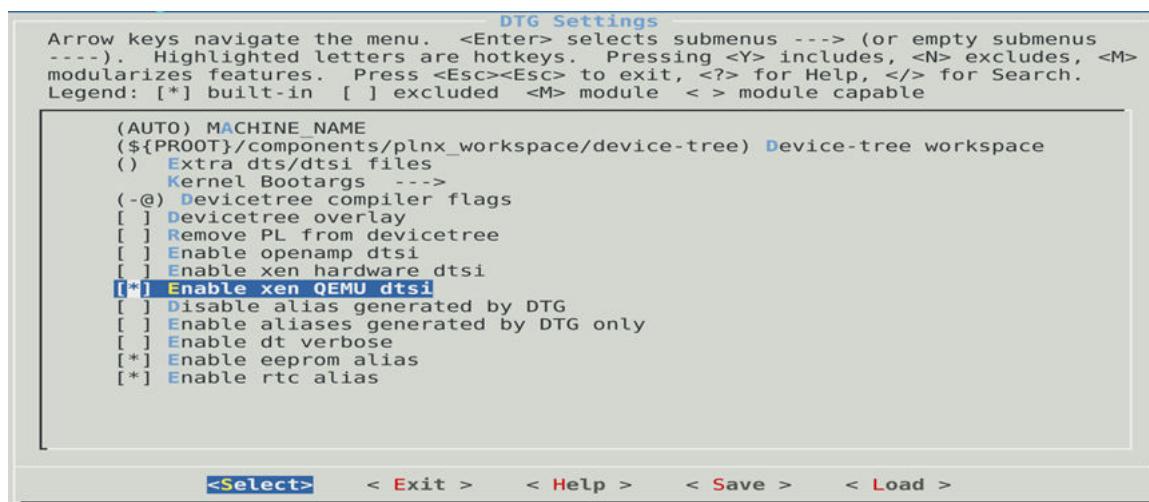
Enable the `Enable xen hardware dtci` configuration to make xen changes part of system.dtb.

Earlier, the xen hardware dtci files were in `<plnx-proj>/project-spec/meta-user/recipes-bsp/device-tree/files/xen.dtsi`. You have to include these files in system-user.dtsi and build it as a part of system.dtb.

From 2023.1, the `xen.dtsi` file is moved into the meta-petalinux layer, and by enabling the `Enable xen hardware dtci` configuration in petalinux-config, the xen changes are included as part of system.dtb.

Enable xen qemu dtci

Figure 15: Enable xen qemu dtci



Enable `xen qemu dtci` configuration option in petalinux-config builds the xen qemu dtci changes as part of system.dtb.

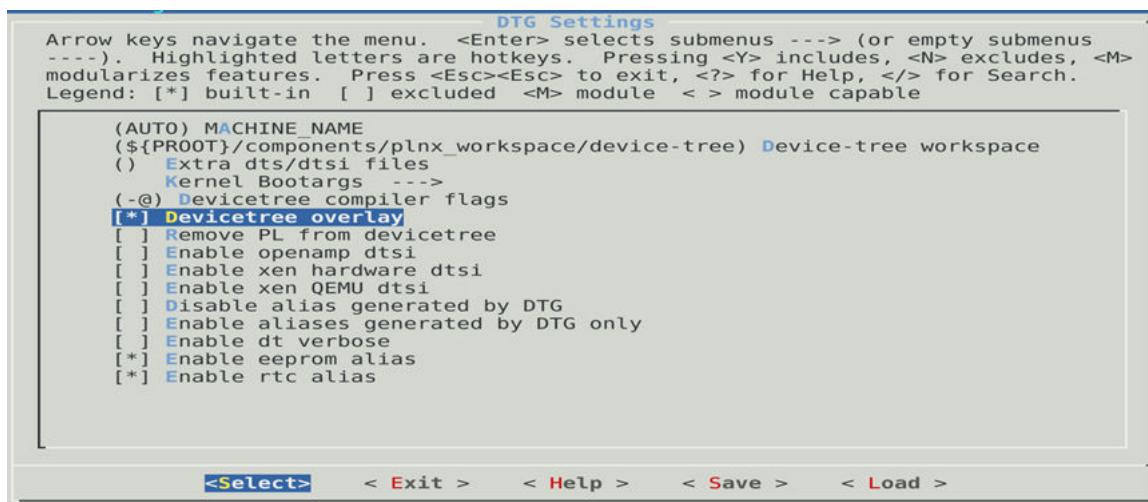
Enable the `Enable xen qemu dtci` configuration to include the xen qemu changes as a part of system.dtb.

Earlier, the `xen qemu dtci` files were in `<plnx-proj>/project-spec/meta-user/recipes-bsp/device-tree/files/xen-hw.dtsi`. You have to include these in system-user.dtsi and build as a part of system.dtb.

From 2023.1, the `xen-qemu.dtsi` file is moved into meta-petalinux layer and by enabling the `Enable xen qemu dtci` configuration, it is included as a part of system.dtb.

Devicetree overlay

Figure 16: Devicetree overlay



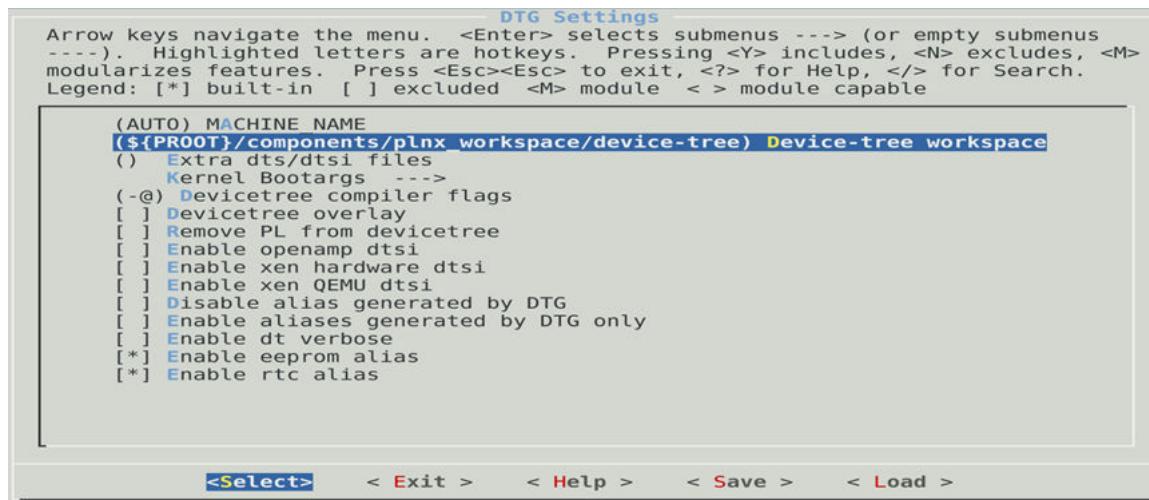
If you want to build pl nodes dtbo files separately, enable the Devicetree overlay configuration to build separate pl.dtbo as an overlay or any other dtsi selected, such as openamp.

Run **petalinux-config → DTG settings → Add extra bootargs**

1. Enter the bootargs which you want to add. For example, ext4=/dev/mmcblk0p3:/rootfs
2. Save and exit
3. Execute the command `petalinux-build`
4. Verify by viewing this file in `<plnx-proj-root>/components/plnx_workspace/device-tree/device-tree/system-conf.dtsi`

Device-tree Workspace

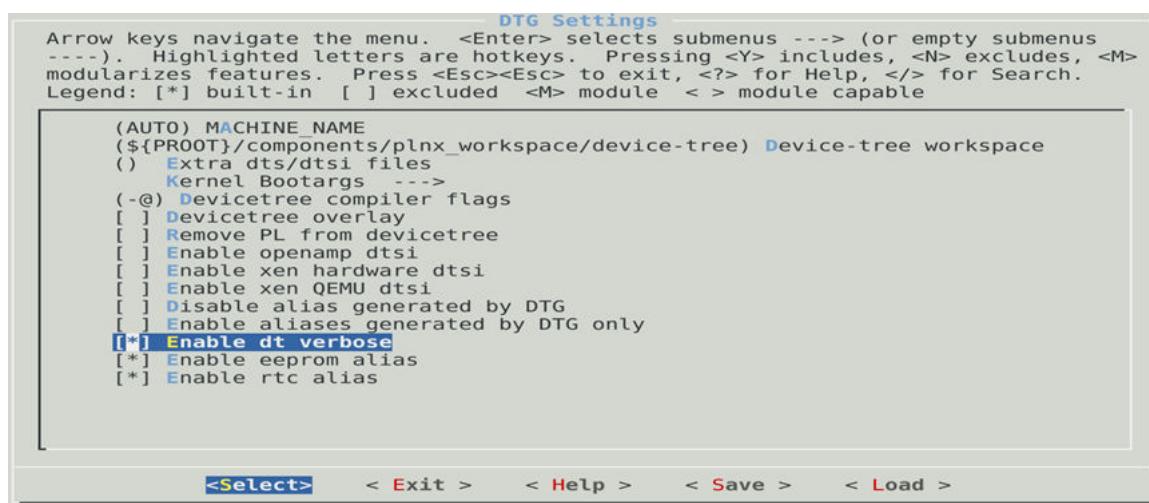
Figure 17: Device-tree workspace



Select the device tree workspace using the `Device-tree workspace` configuration and dtg generates the dtsi files in the configured path.

Enable DTG logs

Figure 18: Enable DTG logs

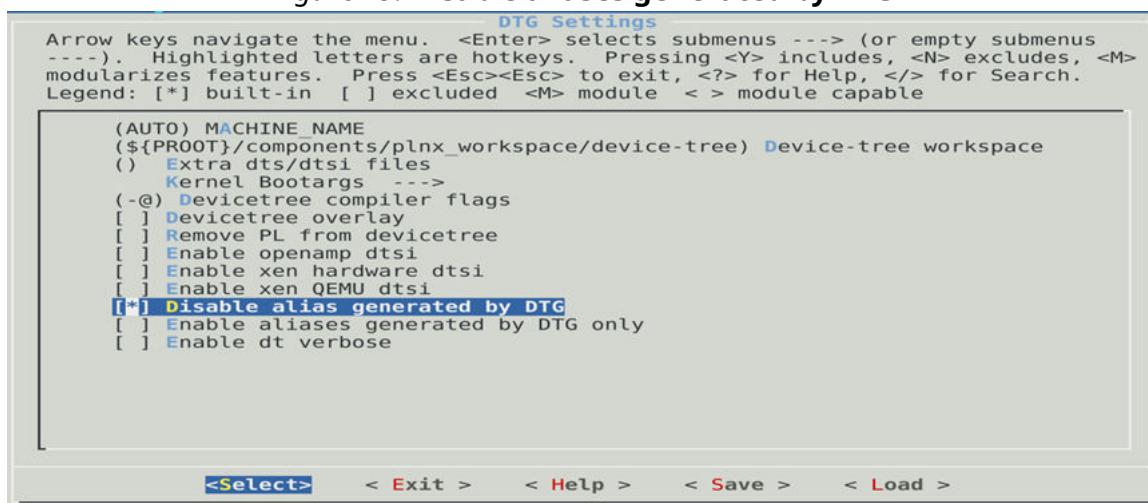


If you want to enable the debug logs in the device tree generator, enable the `Enable dt verbose` configuration.

Note: By default, `Enable dt verbose` is disabled in the tool.

Disable alias generated by DTG

Figure 19: Disable aliases generated by DTG



By default, DTG generates aliases in `system-top.dts` file. You can enable or disable these by using **Disable alias generated by DTG configuration**. By enabling the **Disable alias generated by DTG** configuration, the final generated `system.dtb` contains the aliases from DT board file if you specified any.

Enable aliases generated by DTG only

In default, the `system.dtb` contains the aliases from the DTG board file and the `system-top.dts` file. Enabling **Enable aliases generated by DTG only** configuration, the `system.dtb` has the aliases generated by the DTG(`system-top.dts`) only.

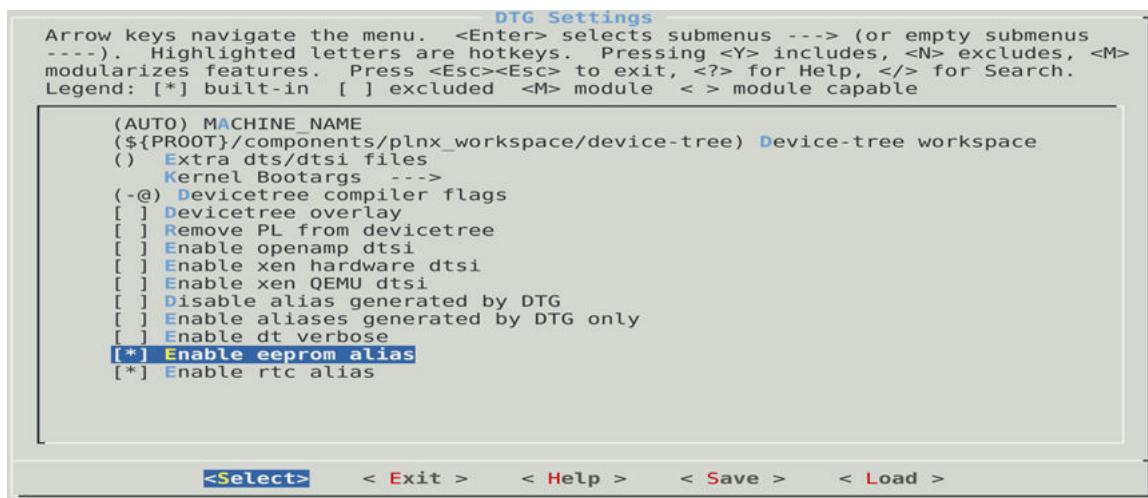
Note: This config does not have any impact if **Disable alias generated by DTG** is enabled.

Table 30: Disable alias generated by DTG vs Enable aliases generated by DTG only

Disable alias generated by DTG	Enable aliases generated by DTG only	Final system.dtb
Disable	Disable	Contains aliases from both DTG board and system-top.dts
Enable	Disable	Contains DTG board aliases
Disable	Enable	Contains system-top.dts aliases
Enable	Enable	Contains DTG board aliases.

Enable eeprom alias

Figure 20: Enable eeprom alias

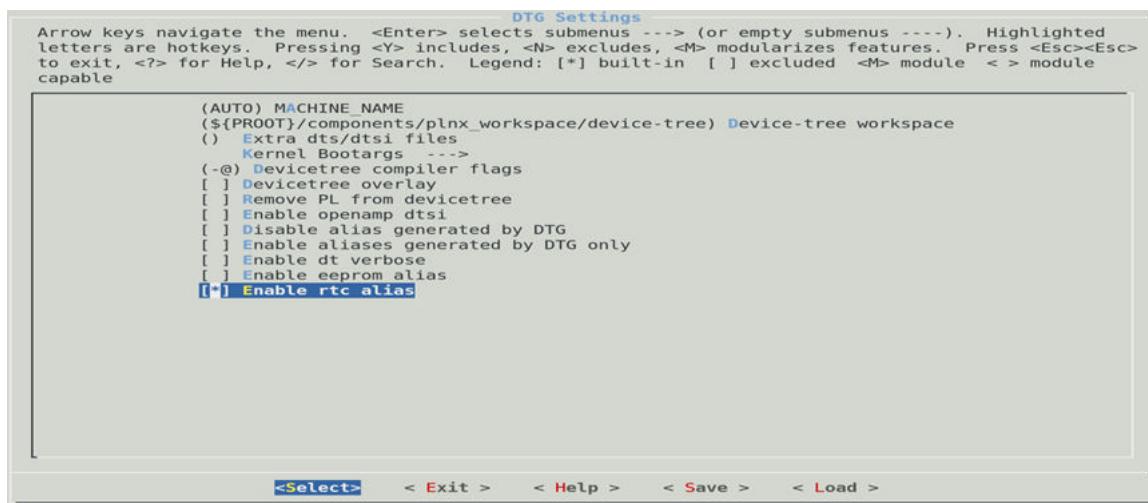


By Default, the tool enables the eeprom to address the user-selected eeprom for Zynq UltraScale+ MPSoC only.

For other platforms, you can enable using the `Enable eeprom alias` configuration option.

Enable rtc alias

Figure 21: Enable rtc alias

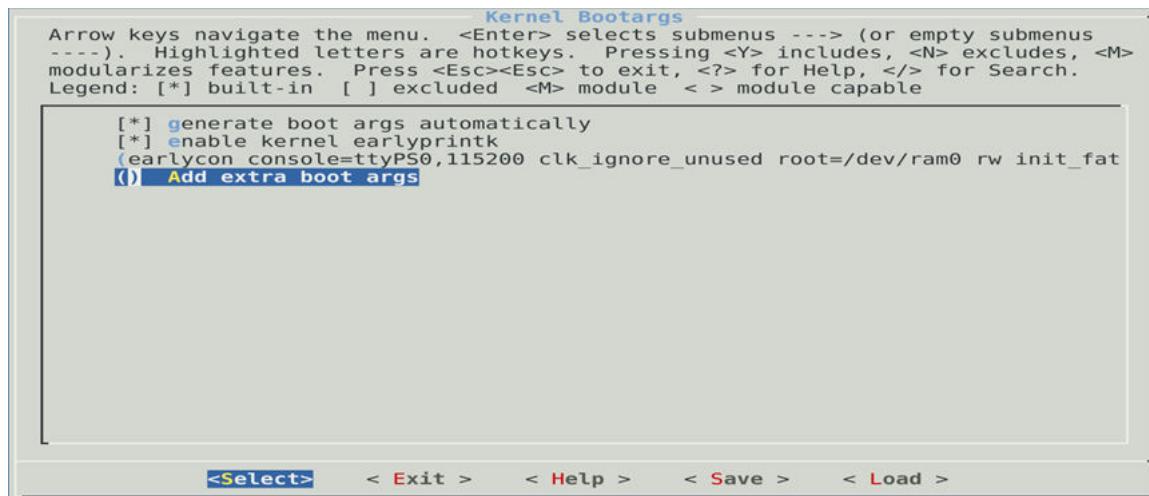


By Default, rtc is enabled to address the user-selected rtc for Zynq UltraScale+ MPSoC only.

For other platforms, you can enable using the `Enable rtc alias` configuration.

Add extra bootargs menu

Figure 22: Add extra bootargs menu

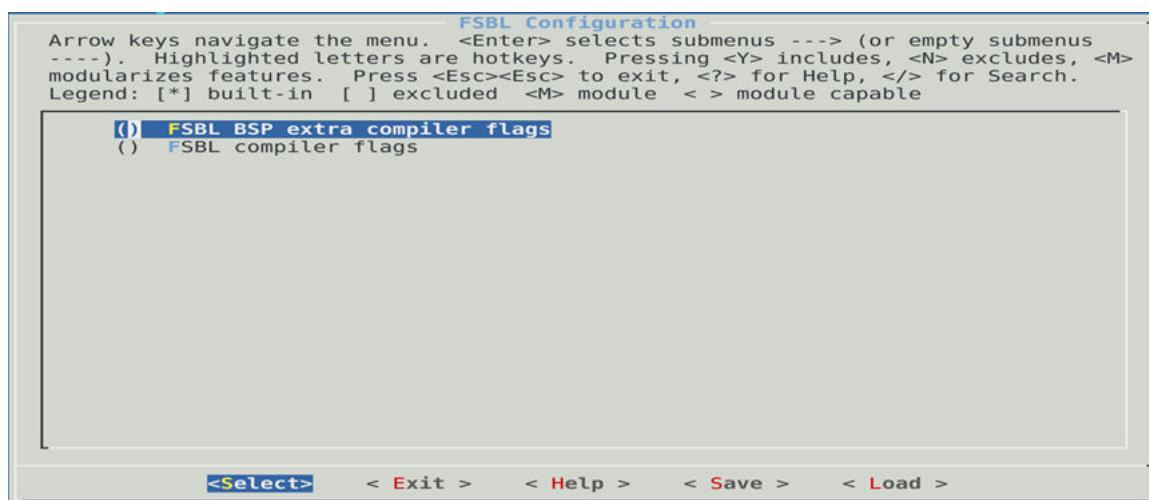


If you want to append any extra boot param other than the default example like setting cma, use the Add extra bootargs menu configuration. this gets appended to the existing boot args.

PMU Firmware Configuration for Zynq UltraScale+ MPSoC

You can provide compiler flags for the PMU firmware application. For example: -DENABLE_IPI_CRC to enable CRC check on IPI messages.

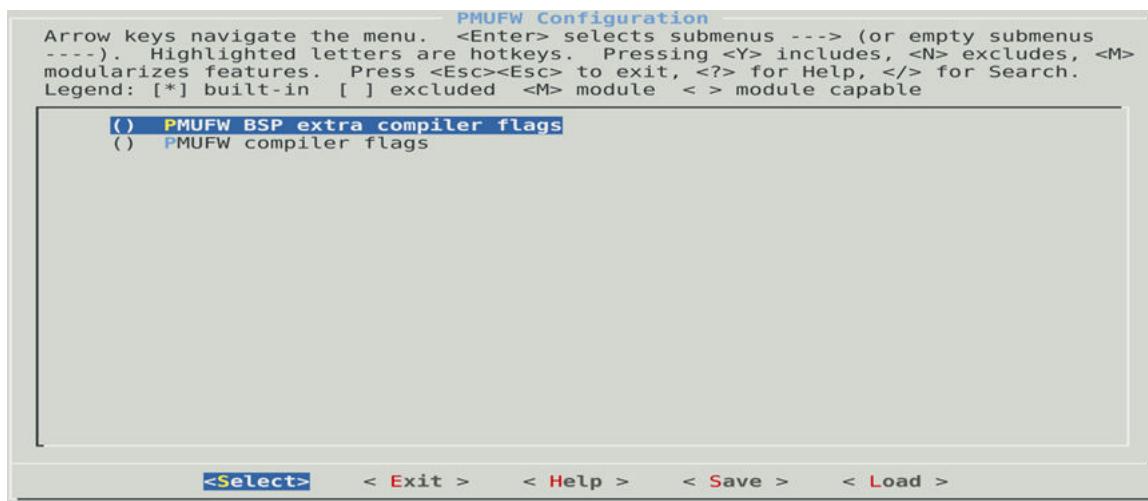
Figure 23: PMU Firmware Configuration



Added Extra Compiler Flags

From 2023.1, you are provided with extra pmufw compiler flag configuration in petalinux-config.

Figure 24: Extra Compiler Flag inPMU Firmware Configuration

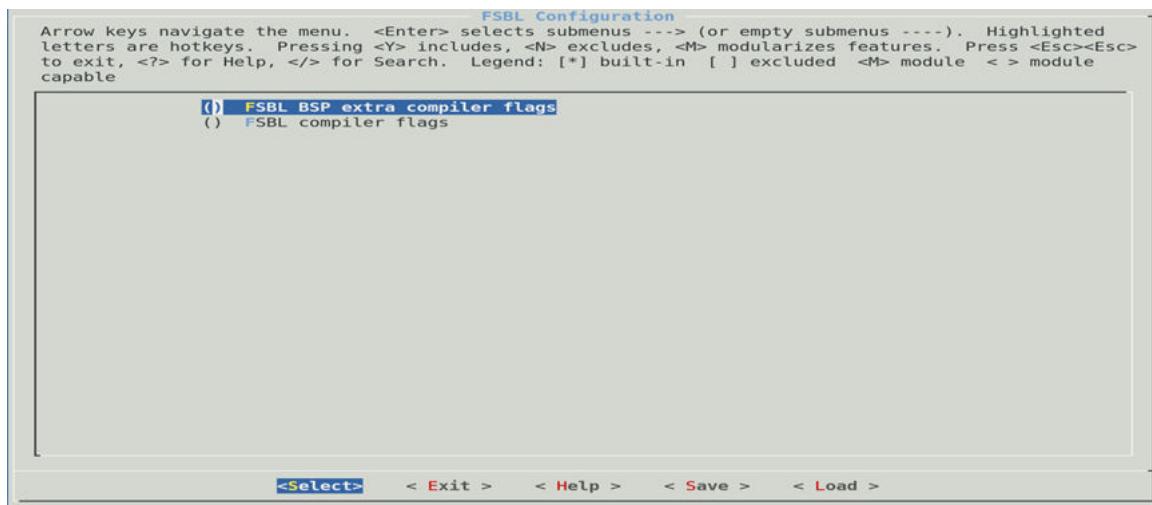


PMUFW BSP extra compiler flags option can specify multiple BSP flags separated with space. For example: DXSK_ACCESS_USER_EFUSE for enabling USER EFUSE access.

FSBL Configuration for Zynq UltraScale+ MPSoC

- **FSBL BSP extra compiler flags:** You can put multiple settings there, separated with space. For example: -DENABLE_IPI_CRC for enabling CRC check on IPI messages.
- **FSBL compilation Settings:** You can put multiple settings there, separated with space. For example: -DFSBL_PROT_BYPASS.

Figure 25: FSBL Configuration



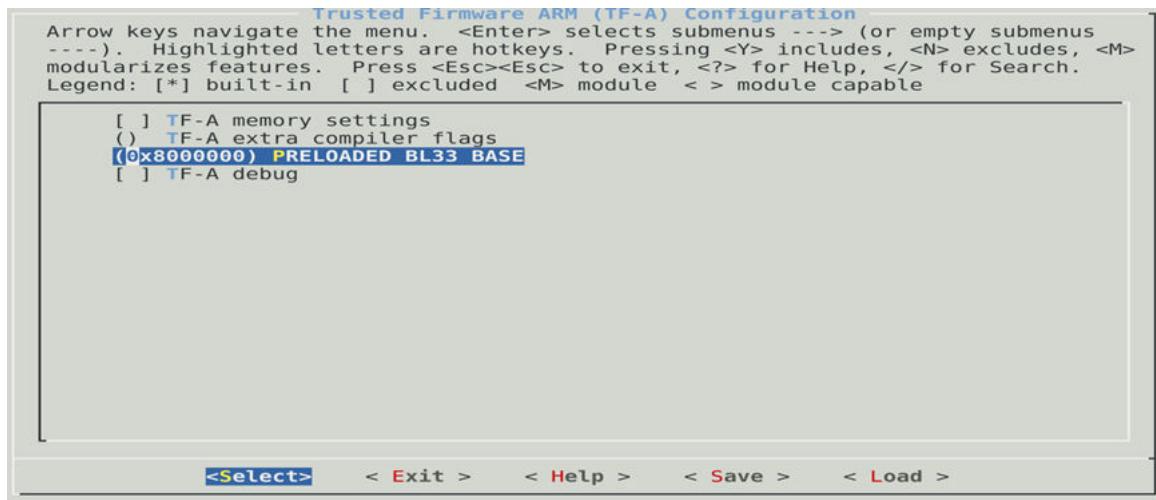
Trusted Firmware-A Configuration for Zynq UltraScale+ MPSoC and Versal Adaptive SoC

The Trusted Firmware-A (TF-A) Compilation configuration submenu allows you to set:

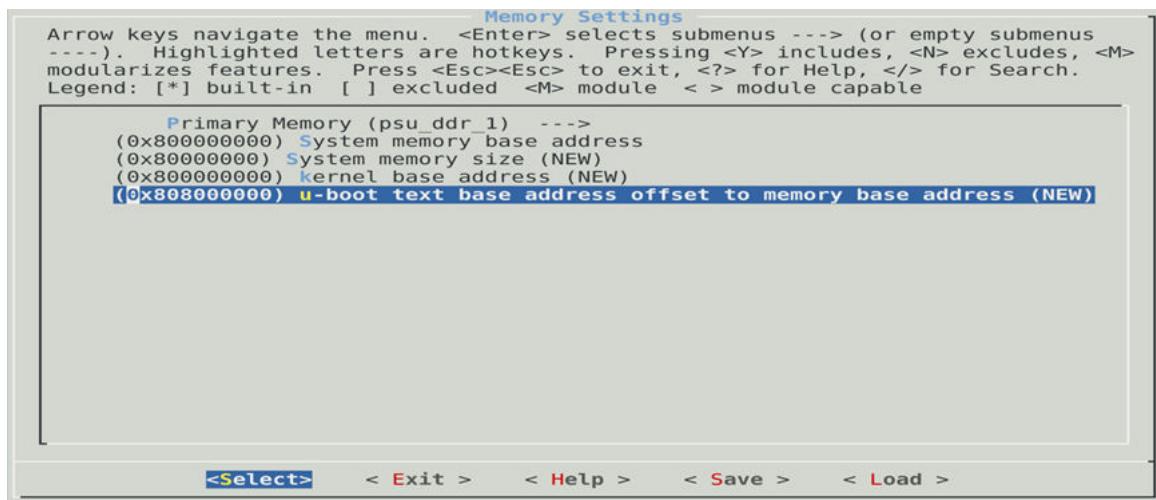
- Extra TF-A compilation settings
- Change the base address of bl31 binary
- Change the size of bl31 binary
- Enable debug in TF-A

Figure 26: Trusted Firmware-A Configuration

If you provide an offset for BL33_BASE address for Trusted-firmware arm through the following config petalinux-config → Trusted Firmware ARM (TF-A) Configuration → PRELOADED BL33 BASE.



Note: The same configuration changes should be taken care for U-Boot text base address petalinux-config → Subsystem Auto Hardware Settings → Memory Settings→ select Primary Memory → u-boot text base address.



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

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

After creating a PetaLinux project, follow these steps to build the FPGA manager support:

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

Note: The PetaLinux FPGA manager configuration performs the following:

1. Generates the `pl.dtsi` nodes as a dt overlay (`dtbo`).
2. Packs the `dtbos` and bitstreams in the `.bin` format or `pdi` into the `/lib/firmware/xilinx/base` directory in the root file system.
3. You can use the `petalinux-create` command to add the PL `xsa` files into the PetaLinux project. The following command creates the `fpgamanager_dtg` app with the `xsa` file to generate the `dtsi` and bit files.

```
petalinux-create -t apps --template fpgamanager_dtg -n can-interface --  
srcuri <path-to-xsa>/system.xsa --enable  
INFO: Create apps: can-interface  
INFO: Copying source urls into the project directory  
INFO: New apps successfully created in <plnx-proj-root>/project-spec/  
meta-user/recipes-apps/can-interface  
INFO: Enabling created component...  
INFO: Sourcing build environment  
INFO: Silentconfig rootfs  
INFO: can-interface has been enabled
```

Note: For each XSA, create a separate app using the previous command. FPGA manager should be enabled in `petalinux-config` for both `fpgamanager` and `fpgamanager_dtg` template apps.

4. Run `petalinux-build`.

Example loading full bitstream on target:

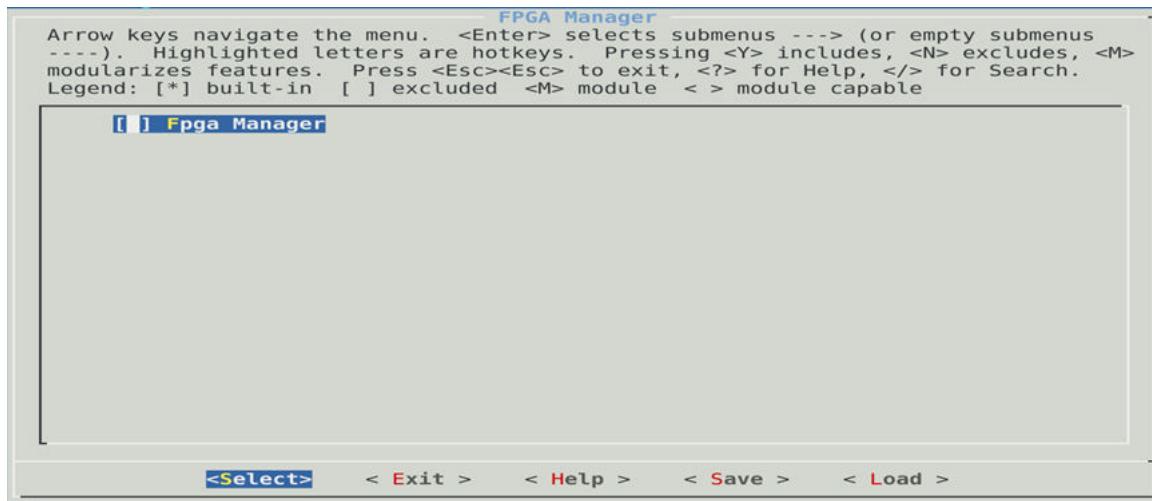
```
root@xilinx-zcu102:~# fpgautl -o /lib/firmware/xilinx/base/pl.dtbo -b  
/lib/firmware/xilinx/base/design_1_wrapper.bit.bin  
  
Time taken to load DTBO is 239.000000 milli seconds. DTBO loaded through  
ZynqMP FPGA manager successfully.
```

Loading a full bitstream through sysfs – loading bitstream only:

```
root@xilinx-zcu102-2023_1:~# fpgautl -b /mnt/design_1_wrapper.bit.bin  
  
Time taken to load BIN is 213.000000 milli seconds. BIN FILE loaded through  
zynqMP FPGA manager successfully.
```

See help section for more option: `root@xilinx-zcu102-2020_1:~# fpgautl -h`. For more information, see <https://www.wiki.xilinx.com/Solution+ZynqMP+PL+Programming>.

Figure 27: FPGA Manager



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

This section provides information on the mechanism and infrastructure required to work with custom IPs that are readily (hand-stitched) available dtsi files instead of relying on the XSA to generate them when the FPGA manager is enabled. This generates the dtbo and bin files and copies them into the rootfs /lib/firmware/xilinx directory.

1. Create the FPGA manager template:

```
petalinux-create -t apps --template fpgamanager -n can-interface --enable
INFO: Create apps: can-interface
INFO: New apps successfully created in <plnx-proj-root>/project-spec/
meta-user/recipes-apps/can-interface
INFO: Enabling created component...
INFO: sourcing build environment
INFO: silentconfig rootfs
INFO: can-interface has been enabled
```

2. Replace the default files with your own dtsi files. You can generate dtsi files using the device tree generator.

```
cp can.dtsi can.bit project-spec/meta-user/recipes-apps/can-interface/
files/
```

3. You can provide the dtsi and bit files when creating the fpgamanager template application. Use the following command to create the application and copy the provided .dtsi and .bit files to the application files directory.

```
petalinux-create -t apps --template fpgamanager -n can-interface --
srcuri "<path-to-dtsi>/pl.dtsi <path-to-bitfile>/system.bit" --enable
INFO: Create apps: can-interface
INFO: Copying source urls into the project directory
INFO: New apps successfully created in <plnx-proj-root>/project-spec/
```

```
meta-user/recipes-apps/can-interface1
INFO: Enabling created component...
INFO: Sourcing build environment
INFO: Silentconfig rootfs
INFO: can-interface has been enabled
```

4. Build the application:

```
petalinux-build
```

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

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

Note: You can modify the FPGA manager template recipe at <project-root-dir>/project-spec/meta-user/recipes-apps/can-interface/can-interface.bb.

DFX Support

This section helps to build and boot DFX (single slot) design for Versal platform.

Note: If DFX applications(rm.xsa) have any memory mapped PL IPs, only then use fpgamanager_dtg_dfx template. If not, you can use --template install to pack only pdi as part of rootfs.

1. Source the PetaLinux tool.
`source /opt/petalinux/petalinux-v<petalinux-version>/settings.sh`
2. Create a Versal template project or bsp project

```
petalinux-create -t project -n versal-dfx --template versal
```

```
petalinux-create -t project -s <bsp path> -n versal-dfx
```

3. Go to the project

```
cd versal-dfx
```

4. Configure the project with static.xsa/base.xsa

```
petalinux-config --get-hw-description <base.xsa/static.xsa>
```

5. Enable FPGA manager using the following command:

```
petalinux-config -> FPGA MANAGER
```

6. Create the static/base application using the following command from static/base xsa:

```
petalinux-create -t apps --template fpgamanager_dtg -n <static-app> --
enable --srcuri "<static xsa>"
```

The previous command creates and packages the static dtbo and pdi files into the rootfs (/lib/firmware/xilinx/) using fpgamanager_dtg template.

7. Create the partial application to configure the partial region using the rm xsa in the following command. You should point static pl app name as `--static-pn` command line option to define the relation between base and partial.

```
petalinux-create -t apps --template fpgamanager_dtg_dfx -n <rm-app> --enable --srcuri <rm.xsa>" --static-pn <static-app>
```

This command generates and packages the rm dtbo, pdi files into the rootfs (`/lib/firmware/xilinx/<static-app>/<rm-app>`).

8. Execute `petalinux-build`

The command generates the rootfs containing both static and rprm dtbos, and respective pdi files as mentioned in the step 5 and step 6.

Once the base boot images ready with the previous step, if you want to build only DFX apps, use the commands mentioned in step 8.

9. Execute the following command:

```
petalinux-build -c <static-app>
petalinux-build -c <rm-app>
in below path you will see with the below names.
in <TMPDIR>/deploy/rpm you will see <static-app>.rpm and <rm-app>.rpm
```

Note: The pdi's in the design should have `i*_partial.pdi` in the xsa files to avoid an error.

Note: In DFX use case, you can use the static xsa to create the Versal boot firmware images so static pdi is packaged as part of BOOT.BIN.

Boot Steps

Up the target with previous built images using any of the boot methods in the documentation. Once the target is up.

```
vck190-dfx:/home/petalinux# fpgautil -o /lib/firmware/xilinx/<static-app>/<static-app>.dtbo
[ 71.571728] OF: overlay: WARNING: memory leak will occur if overlay removed, property: /fpga/external-fpga-config
[ 71.582142] OF: overlay: WARNING: memory leak will occur if overlay removed, property: /__symbols__/overlay0
[ 71.592010] OF: overlay: WARNING: memory leak will occur if overlay removed, property: /__symbols__/fpga_PRO
[ 71.601854] OF: overlay: WARNING: memory leak will occur if overlay removed, property: /__symbols__/overlay2
[ 71.611688] OF: overlay: WARNING: memory leak will occur if overlay removed, property: /__symbols__/static_shell_SIHA_Manager_0
[ 71.623169] OF: overlay: WARNING: memory leak will occur if overlay removed, property: /__symbols__/static_shell_axi_gpio_0
[ 71.634302] OF: overlay: WARNING: memory leak will occur if overlay removed, property: /__symbols__/static_shell_axi_intc_0
[ 71.645435] OF: overlay: WARNING: memory leak will occur if overlay removed, property: /__symbols__/static_shell_clk_gen_clk_wizard_0
[ 71.660647] of-fpga-region fpga:fpga-PRO: FPGA Region probed
[ 71.667512] irq-xilinx: mismatch in kind-of-intr param
[ 71.672705] irq-xilinx: /axi/interrupt-controller@a4060000: num_irq=32, sw_irq=0, edge=0xffffffff
vck190-dfx:/home/petalinux#
```

```
vck190-dfx:/home/petalinux# fpgautil -b /lib/firmware/xilinx/<static-app>/rp0/<rprm-app>/<rprm-app>.pdi -o /lib/firmware/xilinx/<static-app>/rp0/<rprm-app>/<rprm-app>.dtbo -f Partial -n PRO
[ 142.795102] fpga_manager fpga0: writing opendfx-rp0-aes128.pdi to Xilinx Versal FPGA Manager
[152814.496]Loading PDI from DDR
[152814.609]Monolithic/Master Device
[152817.904]3.383 ms: PDI initialization time
[152821.876]+++Loading Image#: 0x0, Name: pl_cfi, Id: 0x18700000
[152827.520]---Loading Partition#: 0x0, Id: 0x103
[152831.949] 0.046 ms for Partition#: 0x0, Size: 28560 Bytes
[152837.180]---Loading Partition#: 0x1, Id: 0x105
[152841.567] 0.005 ms for Partition#: 0x1, Size: 32 Bytes
[152846.582]---Loading Partition#: 0x2, Id: 0x205
[152851.076] 0.112 ms for Partition#: 0x2, Size: 2064 Bytes
[152856.156]---Loading Partition#: 0x3, Id: 0x203
[152860.565] 0.027 ms for Partition#: 0x3, Size: 544 Bytes
[152865.643]---Loading Partition#: 0x4, Id: 0x303
[152875.331] 5.304 ms for Partition#: 0x4, Size: 3238160 Bytes
[152878.011]---Loading Partition#: 0x5, Id: 0x305
[152882.801] 0.406 ms for Partition#: 0x5, Size: 7296 Bytes
[152887.586]---Loading Partition#: 0x6, Id: 0x403
[152892.020] 0.051 ms for Partition#: 0x6, Size: 49312 Bytes
[152897.246]---Loading Partition#: 0x7, Id: 0x405
[152901.633] 0.005 ms for Partition#: 0x7, Size: 32 Bytes
[152906.697]Subsystem PDI Load: Done
[ 142.904337] OF: overlay: WARNING: memory leak will occur if overlay removed, property: /__symbols__/overlay0RP_0_AES128_inst_0
[ 142.918020] OF: overlay: WARNING: memory leak will occur if overlay removed, property: /__symbols__/overlay2_RP_0_AES128_inst_0
[ 142.929504] OF: overlay: WARNING: memory leak will occur if overlay removed, property: /__symbols__/RP_0_AccelConfig_0
[ 142.940202] OF: overlay: WARNING: memory leak will occur if overlay removed, property: /__symbols__/misc_clk_RP_0_AES128_inst_00
[ 142.951768] OF: overlay: WARNING: memory leak will occur if overlay removed, property: /__symbols__/RP_0_axi_gpio_0
[ 142.962208] OF: overlay: WARNING: memory leak will occur if overlay removed, property: /__symbols__/RP_0_rm_comm_box_0
[ 142.975851] gpio-xilinx 20100010000 gpio: #gpio-cells mismatch
[ 142.981709] gpio-xilinx: probe of 20100010000 gpio failed with error -22
Time taken to load BIN is 196.000000 Milli Seconds
BIN FILE loaded through FPGA manager successfully
vck190-dfx:/home/petalinux#
```

Custom dtsi for DFX

1. Source the PetaLinux tool

```
source /opt/petalinux/tool/petalinux-v2023.1-final/settings.sh
```

2. Create the project

```
petalinux-create -t project -n vck190 --template versal
```

3. Go to the project

```
cd vck190/
```

4. Configure the project with the base/static dfx design

```
petalinux-config --get-hw-description <base.xsa/static.xsa>
```

5. Create the static firmware application

```
petalinux-create -t apps --template fpgamanager_dtg --enable -n static-app
```

6. Create the rm firmware application

```
petalinux-create -t apps --template fpgamanager_dtg_dfx -n rprm-app --enable --srcuri "/rm.xsa pl-partial-custom.dtsi" --static-pn static-app
```

7. open the custom dtsi file and modify/add the custom partial dtsi changes

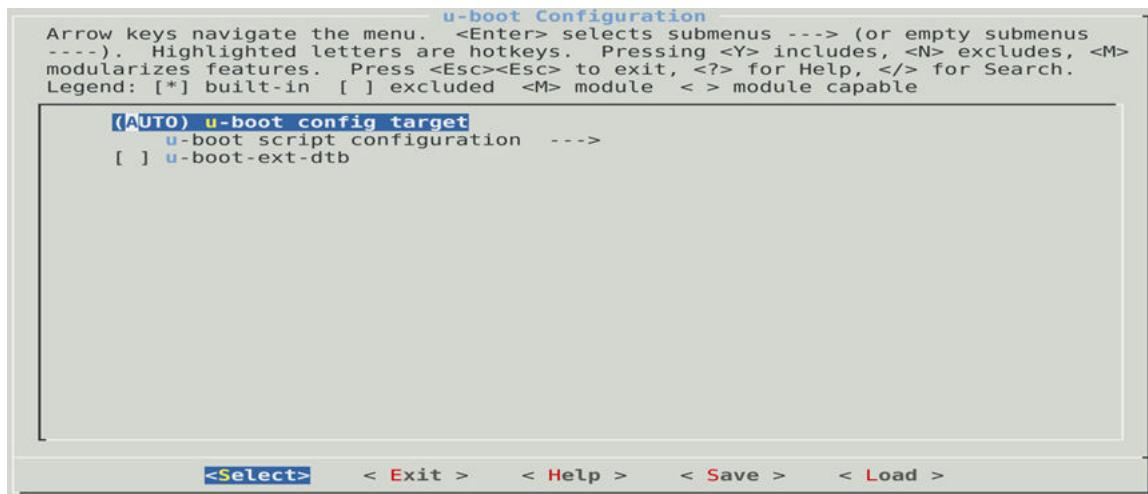
8. Build the project or partial firmware application

```
petalinux-build or petalinux-build -c rprm-app
```

9. Verify the partial dtbo changes

U-Boot Configuration

Figure 28: U-Boot Configuration



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

Building a Separate U-Boot DTB

As shown in the following figure, if you enable the `u-boot dtb` configuration, you can build separate U-Boot DTB with the provided design and you can find the binary in `<plnx-proj-root>/images/linux` with the name `u-boot.dtb`. You can change the name with the `uboot dtb package name` option. If you have a dts file to build a separate U-Boot DTB, you can use the `uboot dts` file path to provide the dts/dtsi files with space separation.

Figure 29: Separate U-Boot DTB



Image Packaging Configuration

The Image Packaging Configuration submenu allows you to set the following image packaging configurations:

- Adding required root file system types.

- File name of the generated bootable kernel image.
- Linux kernel image hash function.
- DTB padding size.
- Whether to copy the bootable images to host TFTP server directory.



TIP: The `petalinux-build` tool always generates a FIT image as the kernel image.

Figure 30: Image Packaging Configuration

```

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

Root filesystem type (INITRD) --->
(0x0) RAMDISK loadaddr
(petalinux-initramfs-image) INITRAMFS/INITRD Image name
(image.ub) name for bootable kernel image
(cpio cpio.gz cpio.gz.u-boot ext4 tar.gz jffs2) Root filesystem formats
(0x1000) DTB padding size
[*] Copy final images to tftpboot
(/tftpboot) tftpboot directory

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

```

Note: You can add extra spaces in the root file system by adding value to this variable <project>/
project-spec/meta-user/conf/petalinuxbsp.conf IMAGE_ROOTFS_EXTRA_SPACE.

Meta-user Layer Changes

In earlier releases, `u-boot-zynq-scr.bbappend` and `bootcmd` files were available in meta-user layers. From 2023.1 release, they are removed from the meta-user layer and moved to Yocto eSDKs at <plnx-proj-root>/components/yocto/layers/meta-xilinx/meta-xilinx-bsp/recipes-bsp/u-boot/u-boot-zynq-scr. You can find these files on running the `petalinux-config/petalinux-build` command.

buildtools-extended

Added `buildtools-extended` as an option in `petalinux-configuration`. Enable this option to use gcc and other binaries as part of the PetaLinux tool. If you are using the newer Yocto version, Langdale, to build the project, the gcc version should be higher than version 6 which is not available by default in most of the supported operating systems. Enable this option to unblock this issue.

serial-autologin-root

serial-autologin-root option in PetaLinux tool is disabled by default. You can enable this with petalinux-config -c rootfs -> Image Features -> [] serial-autologin-root.

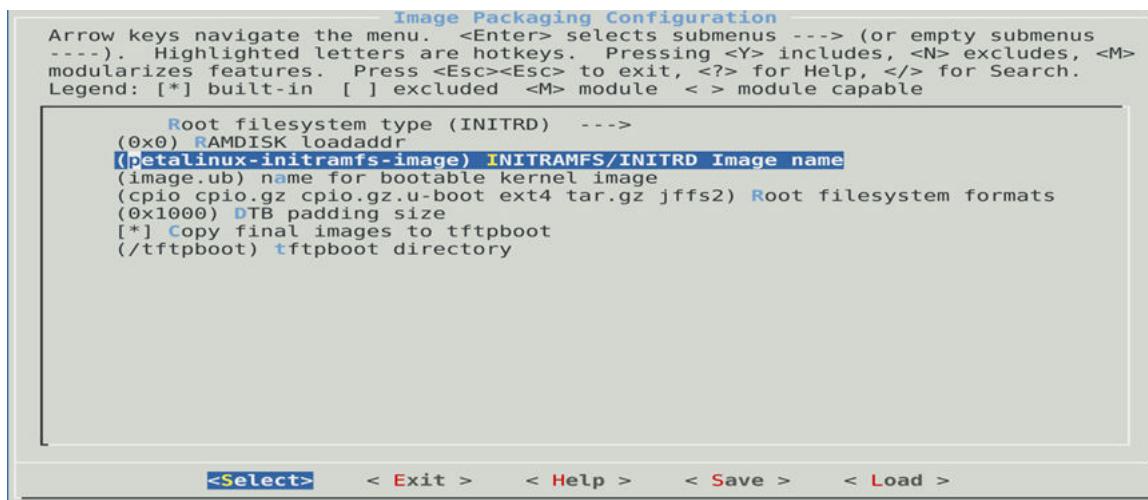
Linux Components Selection

Added Image selector menu configuration option. It is disabled by default.

Option to Change RAM-based Filesystem

There is a menu configuration option to build the initial RAM-based root file system as highlighted in the following figure.

Figure 31: RAM-based Root File System



You can provide the name of the Yocto image recipe that is used to build an initial RAM filesystem in the menu configuration option, INITRAMFS/INITRD Image name. By default, it is set to petalinux-image-minimal for Zynq 7000 devices and MicroBlaze processors and to petalinux-initramfs-image for Zynq UltraScale+ MPSoCs and Versal adaptive SoCs. The petalinux-initramfs-image is a small file system which is deployed into images/linux as ramdisk.cpio.gz.u-boot. It is also packed into Fit image as image.ub. This allows the specified image to be bundled inside the kernel image. An INITRAMFS/INITRD image provides a temporary root file system used for early system initialization, for example, when loading modules that are needed to locate and mount the "real" root filesystem.

After configuration, you can find the following bootargs changes in <plnx-proj-root>/configs/config:

```
init_fatal_sh=1:  
Launch the initramfs prompt if the specified device not available.
```

You can see the two root file systems in <plnx-proj-root>/images/linux:

```
ramdisk.cpio, ramdisk.cpio.gz.u-boot etc..
rootfs.cpio, rootfs.cpio.gz.u-boot etc..
```

Using the petalinux-initramfs-image as the INITRD/INITRAMFS image, boot up the system with tiny rootfs, search for the ext2, ext3, and ext4 SD partitions, and check for the /dev,/etc and /sbin/init file/ folder in it to validate whether the rootfs is valid and can be used as a real file system using the command switch_root. If you want to specify the SD device which has the ext rootfs, add ext4=/dev/mmcblk0p2:/rootfs in system bootargs as follows:

```
petalinux-config ---> DTG Settings ---> Kernel Bootargs ---> [ ] generate
boot args automatically
```

```
petalinux-config ---> DTG Settings ---> Kernel Bootargs ---> (earlycon
console=ttyPS0,115200 clk_ignore_unused init_fatal_sh=1 ext4=/dev/
mmcblk0p2:/rootfs) user set kernel bootargs
```

Disabling switch_root

If you do not want to use the switch_root command which is the default command to mount and use the ext rootfs in Zynq UltraScale+ MPSoC and Versal devices, you can disable it in build time using the following command:

```
petalinux-config ---> Image Packaging Configuration --->
```

Update the name petalinux-initramfs-image to petalinux-image-minimal in INITRAMFS/INITRD image name and run the petalinux-build command.

Note: If you have previously built using the petalinux-initramfs-image command, you continue to see the ramdisk.* images in the images/linux folder. To avoid confusion, check the time stamp.

Note: If you enable Xen when switch_root is enabled, you see build failures because Xen only supports the ramfs boot. Enabling switch_root enables the ext4-based boot. To resolve the issue, change the previous configuration to petalinux-image-minimal from petalinux-initramfs-image.

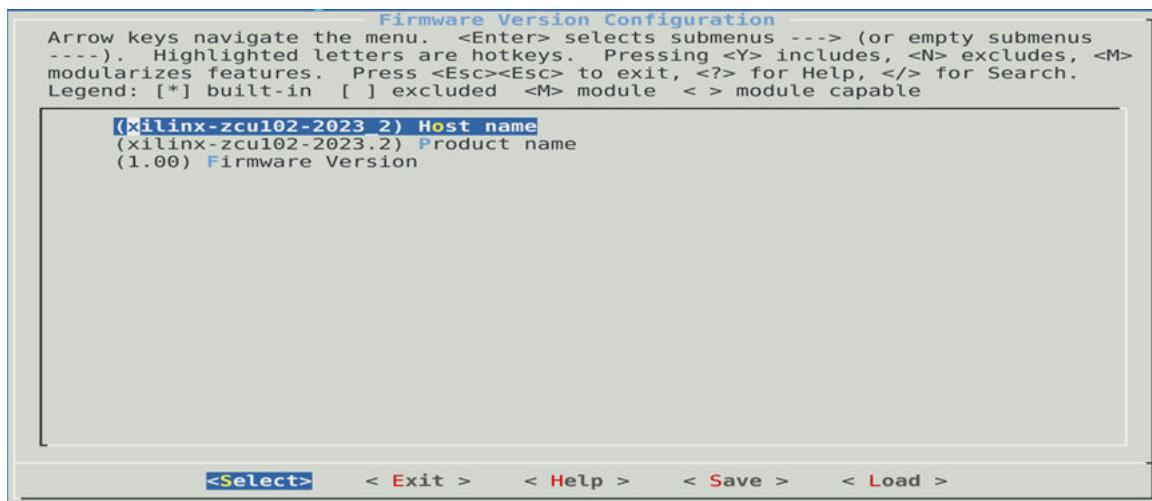
Firmware Version Configuration

The Firmware Version Configuration submenu allows you to set the firmware version information:

Table 31: Firmware Version Options

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

Figure 32: Firmware Version Configuration



TIP: The host name does not get updated. Refer to answer [69122](#) for more details.

Yocto Settings

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

Figure 33: Yocto Settings

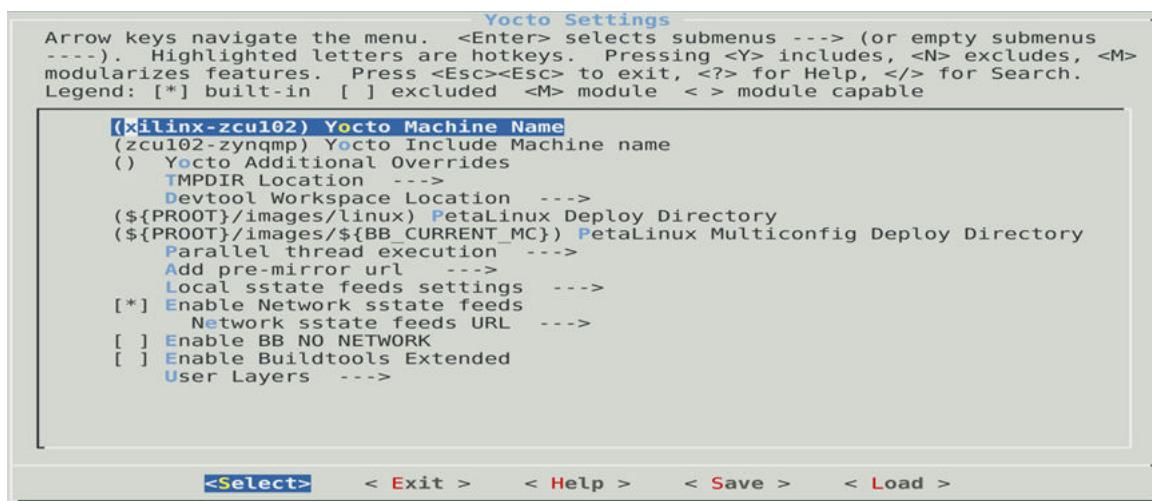


Table 32: Yocto Settings

Parameter	Description
TMPPDIR Location	This directory is used by BitBake to store logs and build artifacts
Yocto Machine Name	Specifies the Yocto machine name for the project. By default PetaLinux uses generic machine names for each platform. For details, see Machine Support

Table 32: Yocto Settings (cont'd)

Parameter	Description
Yocto Include Machine Name	You can specify your own Yocto conf file to include into AUTO generated YOCTO_MACHINE_NAME conf file. By default, it includes generic machine conf files.
Yocto Additional Overrides	You can specify your own Yocto machine overrides to include into AUTO generate YOCTO_MACHINE_NAME conf file. This helps to customize the recipes or apps specific to machine overrides.
PetaLinux Deploy Directory	Configure the path to deploy the generated images. Default is <plnx-project-root>/images/linux
Parallel thread execution	To limit the number of threads of BitBake instances
Add pre-mirror url	Adds mirror sites for downloading source code of components
Local sstate feeds settings	To use local sstate cache at a specific location
Enable Network sstate feeds	Enabled NW sstate feeds
Enable Buildtools extended	Use this option on a machine that does not meet the minimal gcc, Git, tar, and Python requirements
User layers	Adds user layers into projects
BB_NO_NETWORK	When enabled, internet access is disabled on the build machine

Wolfssl config

From 2023.1 release, a new menu configuration is introduced.

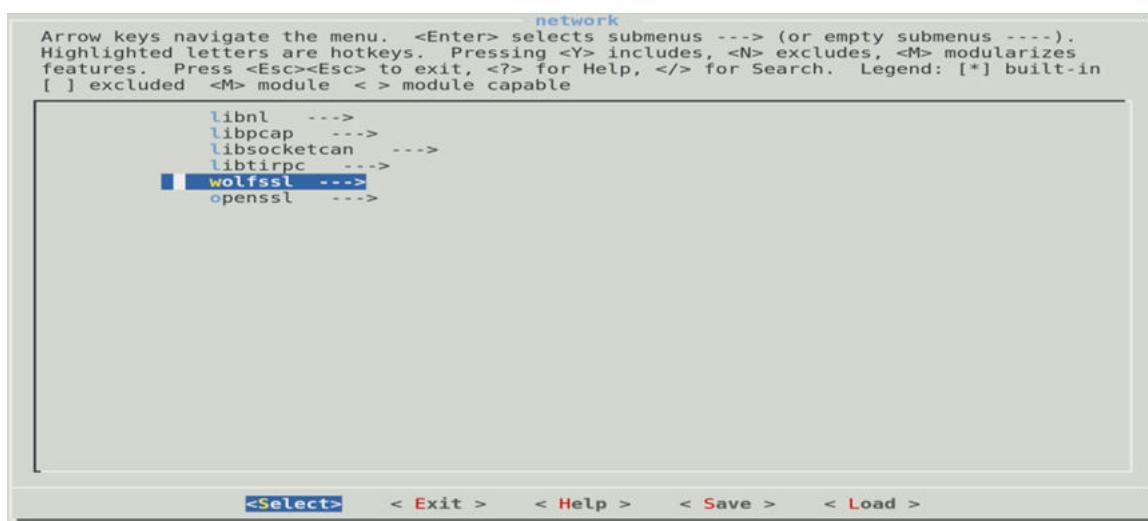
Wolf ssl is a lightweight version of SSL.

Note: You should disable openssl to work with wolfssl.

Select the menu configuration in `petalinux-config -c rootfs`

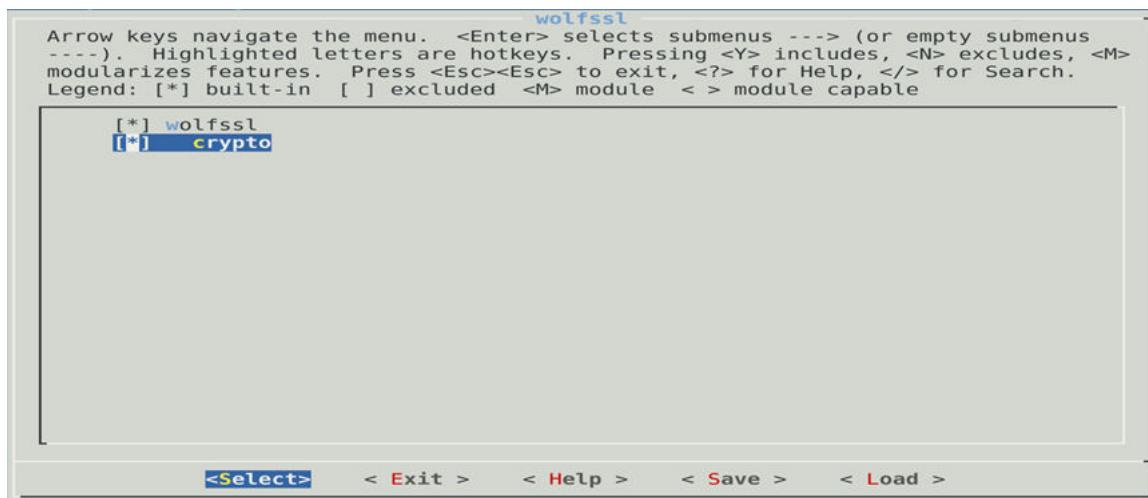
`petalinux-config -c rootfs → File System Packages → libs → network → wolfssl`

Figure 34: Wolfssl



You can enable armv8 crypto assembly instructions also using the menu configuration.

Figure 35: Crypto



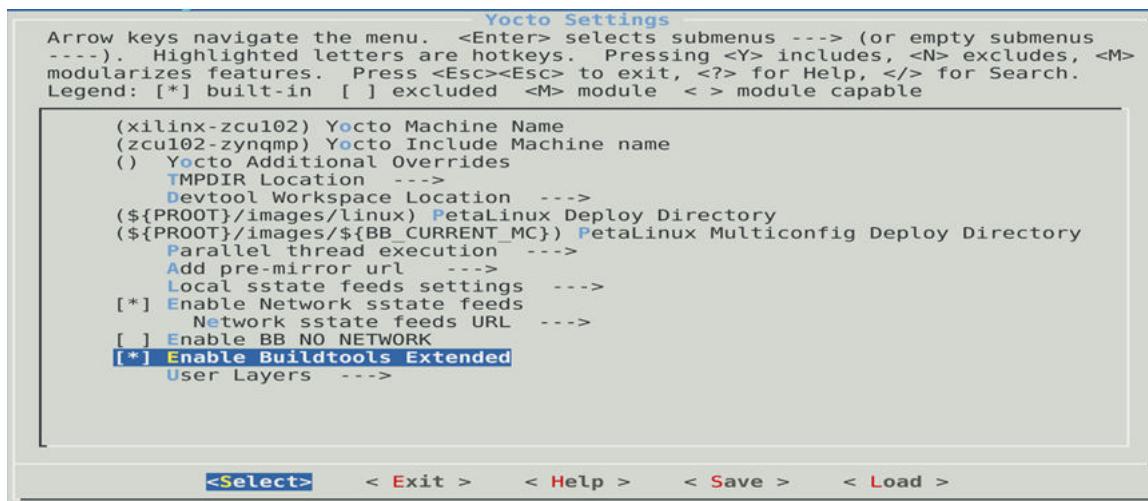
Note: The Wolfssl configuration is supported only for the aarch64 family. Enabling this rebuilds all the components.

Integrated buildtools-extended into PetaLinux

The latest Yocto SDK version requires host development machines to run gcc version 6 or higher. This is currently not available by default in some UBUNTU versions . If this requirement is not met, the following error is displayed when petalinux-config or petalinux-build is run:

```
Error: Incompatible SDK installer! Your host gcc version is 4.8 and this
SDK was built by gcc higher version.
```

To fix this issue, a config option is provided to allow you to use the gcc and other binaries/libraries as a part of the PetaLinux tool (\$PETALINUX/components/yocto/buildtools_extended). Enable the buildtools-extended option in **petalinux-config → Yocto Settings → Enable Buildtools Extended**, as shown in the following figure, to use this option:

Figure 36: Integrated buildtools-extended

Open Source Bootgen for On-target Use for Zynq Devices, Versal Adaptive SoC, and Zynq UltraScale+ MPSoC

If you want to build an open source Bootgen as part of the root file system, follow these steps.

1. Go to the PetaLinux project: `cd <plnx-proj-root>`
2. Run `petalinux-config -c rootfs` and select **Filesystem Packages → Bootgen**
3. Run `petalinux-build`

Once the target is up, you can find the Bootgen binary in `/usr/bin`

Configuring Out-of-tree Build

PetaLinux has the ability to automatically download up-to-date kernel/U-Boot source code from a git repository. This section describes how this feature 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 are 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.

Configuring 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** submenu.

- 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.

Set any of the following values to this setting, otherwise an error message appears.

- To point to HEAD of repository of the currently checked out branch:

```
 ${AUTOREV}
```

- To point to any tag:

```
tag/mytag
```

- To point to any commit id:

```
commit id sha key
```

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



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

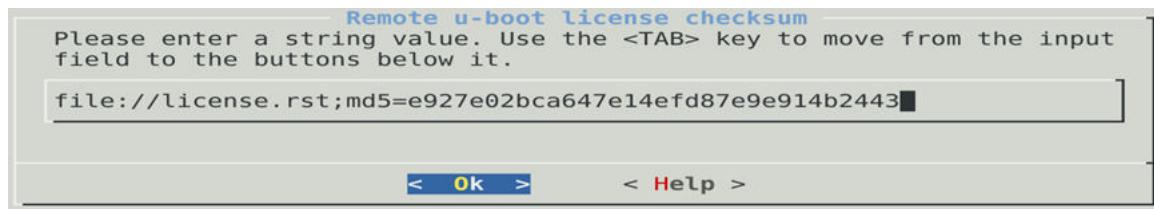
For example, to specify the master branch, type master as shown in the following figure:



6. To specify LICENSE checksum to kernel/u-boot/arm-trusted-firmware/plm/psm-firmware, select **Remote settings (Optional)**.

For example, to specify file://

license.rst;md5=e927e02bca647e14efd87e9e914b2443, enter the string value as shown in the following figure:



7. Exit the menu, and save your settings.

Using External Kernel and U-Boot with PetaLinux

PetaLinux includes kernel source and U-Boot source. However, you can build your own kernel and U-Boot with PetaLinux.

PetaLinux supports local sources for kernel, U-Boot, TF-A, PLM, and PSM-FIRMWARE.

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 submenu

- For kernel, select **linux-kernel ()** ---> and select **ext-local-src**.

() linux-xlnx

() remote

(X) ext-local-src

- For U-Boot, select **u-boot ()** ---> and select **ext-local-src**.

() u-boot-xlnx

() remote

(X) ext-local-src

4. Add external source path.

- For trusted-firmware-arm, select **trusted-firmware-arm()** -> **ext-local-src ()trusted-firmware-arm()remote(x)ext-local-src**

- For PLM, select **plm()-> ext-local-src()plm()remote(*)ext-local-src**

- For PSM-FIRMWARE, select **psm-firmware()-> ext-local-src()psm-firmware()remote(*)ext-local-src**

- For kernel, select **External linux-kernel local source settings** --->. Enter the path:

```
 ${PROOT}/components/ext_sources/<MY-KERNEL>
```

- For U-Boot, select **External u-boot local source settings** --->. Enter the path:

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

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

Note: If after setting ext-local-src, you try to change it to linux-xlnx/u-boot-xlnx in petalinux-config, it gives the following warning.

```
WARNING: Workspace already setup to use from <ext-local-src path>,  
Use 'petalinux-devtool reset linux-xlnx' To remove this (or) Use this  
for your development.
```

Note: When creating a BSP with external sources in project, it is your responsibility to copy the sources into the project and do the packing. For more information, see [BSP Packaging](#).



IMPORTANT! It is not mandatory to have external sources under `components/`. You can specify any location outside the project as well. However, while packaging the BSP, you are responsible for copying the external sources into `components/` and setting relative path.

Note: If the external source is a git repository, its checked out state must be appropriate for the project that is being built.

Troubleshooting

This section describes some common issues you can experience while configuring out-of-tree build.

Table 33: Configuring Out-of-Tree Build Troubleshooting

Problem / Error Message	Description
fatal: The remote end hung up unexpectedly ERROR: Failed to get linux-kernel	Problem Description: This error message indicates that system is unable to download the source code (Kernel/UBOOT) using remote git URL and hence can not proceed with <code>petalinux-build</code> . Solution: Check whether entered remote git URL is proper or not. If the previous solution does not solve the problem, cleanup the build with the following command: <code>petalinux-build -x mrproper</code> Above command removes following directories. <code><plnx-proj-root>/images/</code> <code><plnx-proj-root>/build/</code> Re-build the system image. For more information, see the Building a 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 AMD drivers or optimizations in the Linux kernel configuration are supported by AMD technical support. For more information on AMD drivers for Linux, see <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841873/Linux+Drivers>.

The following examples demonstrate how to use `petalinux-config` to review or modify your PetaLinux project configuration:

1. Change into the root directory of your PetaLinux project

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

2. Launch the top level system configuration menu and configure it to meet your requirements:

```
petalinux-config
```

3. Launch the Linux kernel configuration menu and configure it to meet your requirements:

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

4. Launch the root file system configuration menu and configure it to meet your requirements:

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

5. Use `--silentconfig` for the components when you do not have Kconfig/Menuconfig support or to skip the launching of configuration menu

```
petalinux-config -c <COMPONENT> --silentconfig
```

Warning Message for `petalinux-config` or `petalinux-build` Commands

The following warning message appears when you run the `petalinux-config` or `petalinux-build` for components (for example, `petalinux-build -c u-boot`) and this can be ignored.

```
WARNING: SRC_URI is conditionally overridden in this recipe, thus several devtool-override-* branches have been created, one for each override that makes changes to SRC_URI. It is recommended that you make changes to the devtool branch first, then checkout and rebase each devtool-override-* branch and update any unique patches there (duplicates on those branches will be ignored by devtool finish/update-recipe).
```



TIP: Set U-Boot target in `petalinux-config` menuconfig as required, for your custom board. Set \$ `petalinux-config` → **U-Boot Configuration** → **u-boot config target** as required. Default values for AMD evaluation boards are as follows:

- For Zynq 7000 devices, `xilinx_zynq_virt_defconfig`
- For Zynq UltraScale+ MPSoC, `xilinx_zynqmp_virt_defconfig`
- For MicroBlaze processors, `microblaze-generic_defconfig`
- For Versal devices, `xilinx_versal_virt_defconfig`

Note: Make sure board and user specific dtsi entries are added to `project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi`.

Using template flow, for zcu102 and zcu106 boards, add the following line to <plnx-proj-root>/project-spec/meta-user/recipes-bsp/embeddedsw/fsbl-firmware%.bbappend for FSBL initializations.

```
YAML_COMPILER_FLAGS:append = " -DXPS_BOARD_ZCU102" # for zcu102  
YAML_COMPILER_FLAGS:append = " -DXPS_BOARD_ZCU106" # for zcu106
```

Device Tree Configuration

This section describes which files are safe to modify for the device tree configuration and how to add new information into the device tree.

Prerequisites

This section assumes that you have PetaLinux tools software platform ready for building a Linux system customized to your hardware platform. For more information, see [Importing Hardware Configuration](#). Knowledge of DTS syntax is required to customize the default DTS.

Configuring Device Tree

User-modifiable PetaLinux device tree configuration is associated with following config files, that are located at <plnx-proj-root>/project-spec/meta-user/recipes-bsp/device-tree/files/:

- system-user.dtsi

The generated files are in the <plnx-proj-root>/components/plnx_workspace/device-tree/device-tree/ directory.



CAUTION! These DTSI files are auto-generated. Do not edit these files

For more details on device tree files, see [Appendix C: PetaLinux Project Structure](#).

To add information like the Ethernet PHY, include the information in the `system-conf.dtsi` file. In this case, device tree should include the information relevant for your specific platform as information (here, Ethernet PHY information) is board-level and board-specific.

The `system-user.dtsi` is automatically created when you configure your PetaLinux project. Once created, the tools do not update it automatically.

Note: The need for this manual interaction is because some information is "board level," and the tools do not have a way of predicting what should be here. Refer to the Linux kernel Device Tree bindings documents (`Documentation/devicetree/bindings` from the root of the kernel source) for the details of the bindings of each device.

An example of a well-formed device tree node for the `system-user.dtsi` is as follows:

```
/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 = <'>;
        };
    };
};
```

 **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"
```



RECOMMENDED: It is not recommended to change anything in `<p1nx-proj-root>/components/p1nx_workspace/device-tree/device-tree/`.



RECOMMENDED: 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 a higher priority.

You can overwrite any existing value in other DTSIs by defining in system user DTSIs.

U-Boot Configuration

This section describes which files are safe to modify for the U-Boot configuration and discusses about the U-Boot `CONFIG_` options/settings.

Prerequisites

This section assumes that you have PetaLinux tools software platform ready for building a Linux system customized to your hardware platform. Refer to section [Importing Hardware Configuration](#) for more information.

Configuring U-Boot

Universal boot loader (U-Boot) configuration is usually done using C pre-processor. It defines:

- **Configuration _OPTIONS_:** You can select the configuration options. They have names beginning with CONFIG_.
- **Configuration _SETTINGS_:** These depend on the hardware and other factors. They have names beginning with CONFIG_SYS_.



TIP: Detailed explanation on *CONFIG_* options/settings documentation and README on U-Boot can be found at [Denx U-Boot Guide](#).

PetaLinux U-Boot configuration is associated with config.cfg configuration file located at <plnx-proj-root>/project-spec/configs/u-boot-xlnx/ .

Note: config.cfg is generated only when **petalinux-config → Auto Config Settings → u-boot autoconfig** is enabled.

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



IMPORTANT! If **petalinux-config → Auto Config Settings → u-boot autoconfig** is disabled, the *platform-top.h* changes are not reflected directly. To resolve this, see [Steps for Managing Image Size](#).

Steps to add CONFIG_ option (For example, CONFIG_CMD_MEMTEST) to platform-top.h:

- Change into the root directory of your PetaLinux project.

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

- Open the file platform-top.h

```
vi project-spec/meta-user/recipes-bsp/u-boot/files/platform-top.h
```

- If you want to add CONFIG_CMD_MEMTEST option, add the following line to the file. Save the changes.

```
#define CONFIG_CMD_MEMTEST
```



TIP: Defining CONFIG_CMD_MEMTEST enables the Monitor Command "mtest", which is used for simple RAM test.

- Build the U-Boot image.

```
petalinux-build -c u-boot
```

- Generate **BOOT.BIN** using the following command.

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

- Boot the image either on hardware or QEMU and stop at U-Boot stage.

- Enter the `mtest` command in the U-Boot console as follows:

```
ZynqMP mtest
```

- Output on the U-Boot console should be similar to the following:

```
Testing 00000000 ... 00001000:  
Pattern 00000000 Writing... Reading... Iteration:  
20369
```



IMPORTANT! If `CONFIG_CMD_MEMTEST` is not defined, output on U-Boot console is as follows:

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

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

Yocto Features

SDK Generation (Target Sysroot Generation)

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

Building SDK

The following command builds SDK and copies it at <proj_root>/images/linux/sdk.sh.

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

The following is the equivalent BitBake command.

```
bitbake petalinux-image-minimal -c do_populate_sdk
```

Installing SDK

The generated SDK has to be installed/extracted to a directory. The following command extracts the SDK to a specified directory. The default SDK is <proj_proot>/images/linux/sdk.sh and default installation directory is <proj_proot>/images/linux/sdk/.

```
petalinux-package --sysroot -s|--sdk <custom sdk path> -d|--dir <custom directory path>
```

Examples

Few examples for building the SDK:

1. Adding a cross compiling qt toolchain

To build SDK with qt toolchain:

- a. Create the <plnx-proj-root>/project-spec/meta-user/recipes-core/images/petalinux-image-minimal.bbappend file.

- b. Add `inherit populate_sdk_qt5` in the newly created file.
- c. Run `petalinux-config -c rootfs` and select **packagegroup-petalinux-qt**.
- d. Run `petalinux-build -s`.
- e. Run `petalinux-package --sysroot`.

To verify:

- a. Open a new terminal.
- b. Go to `<plnx-proj-root>/image/linux/sdk`.
- c. Run `source environment-setup-aarch64-xilinx-linux`.
- d. Run `which qmake`. This confirms that the qmake is coming from the SDK.

2. Building OpenCV applications

- a. Create a PetaLinux project.
- b. Add `packagegroup-petalinux-opencv` in the RootFS menu config.
- c. Build SDK.

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

This command builds SDK and deploys it at `<plnx-proj-root>/images/linux/sdk.sh`.

- d. Install SDK.

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

This command installs SDK at `<plnx-proj-root>/images/linux/sdk`.

- e. Use the `images/linux/sdk` directory as sysroot for building the OpenCV applications.

Building an eSDK

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

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

Packaging Sources and Licenses

- To pack all the components of `petalinux-build`, issue the following commands.

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

- To pack only the sysroot components, use the following command.

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

Note: You can find the archiver tar in <plnx-proj-root>/images/linux.

Accessing BitBake/Devtool in a Project

BitBake is available only in the bash shell.

Steps to Access the BitBake Utility

To access the BitBake utility:

1. Source the PetaLinux tools script:

```
source /opt/pkg/petalinux/settings.sh
```

2. Run petalinux-create command to create project

```
petalinux-create -t project -s <PATH_TO_PETALINUX_PROJECT_BSP>
```

3. Run petalinux-config or petalinux-config --silentconfig at least once after creating the project to setup the required environment.

4. Run unset LD_LIBRARY_PATH by using below command

```
unset LD_LIBRARY_PATH
```

5. Source the Yocto e-SDK:

```
source <plnx-proj-root>/components/yocto/environment-setup-cortexa72-cortexa53-xilinx-linux
```

6. Source the environment setup script to be redirected to the build directory:

```
source <plnx-proj-root>/components/yocto/layers/poky/oe-init-build-env
```

Stay in the build directory to run BitBake

7. Export PROOT:

Note: PROOT PATH should not end with "/" .

```
export PROOT=<path of your petalinux project directory>"
```

8. Parse the PetaLinux variable to recipes:

```
export BB_ENV_PASSTHROUGH_ADDITIONS="$BB_ENV_PASSTHROUGH_ADDITIONS  
PETALINUX PROOT"
```

9. To test if the BitBake is available, run:

```
bitbake strace
```

The generated images are placed in the deploy directory. You have to copy the generated images into <plnx-proj-root>/images/linux directory to work with the other commands

Steps to Access the Devtool Utility

To access the Devtool utility:

1. Follow steps 3-7 as described in [Steps to Access the BitBake Utility](#)
2. Create a workspace for devtool:

```
devtool create-workspace ../components/pinx_workspace/
```

3. Add the recipe to workspace directory:

```
devtool add --version 1.0 gpio-demo ../project-spec/meta-user/recipes-apps/gpio-demo
```

4. Build the recipe:

```
devtool build gpio-demo
```

For more devtool commands, type `devtool --help`.

Shared State Cache

Yocto e-SDK contains minimal shared state (sstate) cache. AMD hosts the full petalinux-image-minimal sstate cache at <https://petalinux.xilinx.com/sswreleases/rel-v2023>.

During petalinux-build, BitBake searches for the sstate cache in <https://petalinux.xilinx.com/sswreleases/rel-v2023>. If it fails to find the sstate cache, BitBake builds it from scratch. sstate is signature locked.

For a `.bbappend` file which you create for any root file system component, you must add `SIGGEN_UNLOCKED_RECIPES += "<recipe-name>"` (for example, `example`, `SIGGEN_UNLOCKED_RECIPES += "u-boot-xlnx"` in `<plnx-proj-root>/project-spec/meta_user/conf/petalinuxbsp.conf`.

Sharing your State Cache

If you want to share/use your previously build sstate cache, you can follow either of the following approaches.

As an optimization, the Yocto Project optimizes downloads of the sstate cache items to only the minimal items required for the current build. This must be factored in when sharing your sstate cache with another user. The configuration of the second user can be different, causing a different set of sstate cache items to be required. There are two approaches to optimizing your downstream user and their usage of the sstate cache. The first approach is that the second user should include both the sstate cache directory you provided and the original AMD sstate cache directory in <plnx-proj-root>/build/conf/plnxtool.conf.

```
SSTATE_MIRRORS = " \
file://.* file://<your-sstate-cache>/PATH \n \
file://.* http://petalinux.xilinx.com/sswreleases/rel-v2022/aarch64/sstate-
cache/PATH;downloadfilename=PATH \n \"
```

The second approach is to fetch all of the sstate cache items that can be required for a particular build. This is required if you want to share your build sstate with the downstream user. There is an option called `--setscene-only` that fetches all of the sstate objects that might be needed for a particular target recipe. For example, if you used `petalinux-build` (`bitbake petalinux-image-minimal`), you should run the following command first to fetch all the required sstate from AMD provided sstate.

```
petalinux-build -c "petalinux-image-minimal --setscene-only" (bitbake
petalinux-image-minimal --setsecene-only)
```

Downloading Mirrors

AMD hosts all source download tar files for each release at <https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools.html>. By default, PetaLinux points to pre-mirrors using `petalinux-config` command.

If any component is rebuilt from scratch, BitBake, or devtool searches for its source in pre-mirrors and downloads the mirror URL. Later, it searches in SRC_URI of recipes for downloading the source of that component. If you configure any value through **petalinux-config → yocto settings → premirrors**, it first searches in the configured pre-mirrors, petalinux.xilinx.com, and finally in the SRC_URI in recipes.

You can add more mirrors by adding `SOURCE_MIRROR_URL += file:///home/you/your-download-dir/` in <plnx-proj-root>/project-spec/meta-user/conf/petalinuxbsp.conf.

For more information on how to set SSTATE and DL_DIR, see [How to Reduce Build Time using SSTATE Cache](#).

Machine Support

The Yocto machine specifies the target device for which the image is built. The variable corresponds to a machine configuration file of the same name, which sets machine-specific configurations. Currently, PetaLinux supports the user machine configuration file.

You can add your machine configuration file under <plnx-proj-root>/project-spec/meta-user/conf/machine/ or add your machine configuration file in any additional layers and add it to the project through petalinux-config.

→ () Yocto Include Machine name specifies the machine name included in the () Yocto Machine Name Specified. The default uses the generic config files based on the platform.

Follow these steps to specify the user machine configuration file name in the PetaLinux project:

1. Go to the PetaLinux project.

2. Select petalinux-config → Yocto Settings → () Yocto Machine Name.

3. Specify your machine configuration file name.

Based on the design, the machine configuration files are generated dynamically.

4. Optionally add/modify petalinux-config → Yocto Settings → () Yocto Include Machine name

Table 34: Default Machine Names in Template

Template	Machine
zynq	zynq-generic
zynqmp	zynqmp-generic
microblaze	microblazeel-generic
versal	versal-generic

Table 35: Default Machine Names and Corresponding Include Machine in BSP

BSP name	Machine	Include Machine
xilinx-ac701-v\$version-final.bsp	xilinx-ac701	ac701-microblazeel
xilinx-kc705-v\$version-08071442.bsp	xilinx-kc705	kc705-microblazeel
xilinx-kcu105-v\$version-final.bsp	xilinx-kcu105	kcu105-microblazeel
xilinx-vcu118-v\$version-final.bsp	xilinx-vcu118	vcu118-microblazeel
xilinx-sp701-v\$version-final.bsp	xilinx-sp701	kcu105-tmr-microblazeel
xilinx-zc702-v\$version-final.bsp	xilinx-zc702	zc702-zynq7
xilinx-zc706-v\$version-final.bsp	xilinx-zc706	zc706-zynq7
xilinx-zcu102-v\$version.bsp	xilinx-zcu102	zcu102-zynqmp
xilinx-zcu104-v\$version.bsp	xilinx-zcu104	zcu104-zynqmp
xilinx-zcu106-v\$version.bsp	xilinx-zcu106	zcu106-zynqmp

Table 35: Default Machine Names and Corresponding Include Machine in BSP (cont'd)

BSP name	Machine	Include Machine
xilinx-zcu111-v\$version.bsp	xilinx-zcu111	zcu111-zynqmp
xilinx-zcu208-v\$version.bsp	xilinx-zcu208	zcu208-zynqmp
xilinx-zcu216-v\$version.bsp	xilinx-zcu216	zcu216-zynqmp
xilinx-zcu670-v\$version.bsp	xilinx-zcu670	zcu670-zynqmp
xilinx-k26-som-v\$version.bsp	xilinx-k26-som	k26-sm
xilinx-vck190-SC-v\$version-final.bsp	xilinx-vck190-sc	vck-sc-zynqmp
xilinx-vck190-v\$version-final.bsp	xilinx-vck190	vck190-versal
xilinx-vc-p-a2197-00-reva-x-prc-02-reva-v\$vesion-final.bsp	xilinx-vc-p-a2197-00	vc-p-a2197-00-versal
xilinx-vck5000-v\$version-final.bsp	xilinx-vck5000	vck5000-versal
xilinx-vek280-v\$version-final.bsp	xilinx-vek280	vek280-versal
xilinx-vhk158-v\$version-final.bsp	xilinx-vhk158	vhk158-versal
xilinx-vmk180-v\$version-final.bsp	xilinx-vmk180	vmk180-versal
xilinx-vpk120-v\$version-final.bsp	xilinx-vpk120	vpk120-versal
xilinx-vpk180-v\$version-final.bsp	xilinx-vpk180	vpk180-versal

Note: In the previous table, \$version indicates PETALINUX_VERSION

Note: For TEMPLATE, if you specify any of the mentioned Machine names, the corresponding included machine name is auto-updated. You can update it based on your needs.

SoC Variant Support

AMD delivers multiple devices for each SoC product. AMD 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). AMD distributes SoCs with multiple variants as follows:

1. Zynq 7000 devices are distributed as Zynq7000zs and Zynq7000z. The available SOC_VARIANTs are:
 - "7zs" - Zynq 7000 Single A9 Core
 - "7z" - Zynq 7000 Dual A9 Core
 - Default SOC_VARIANT for Zynq 7000 devices is "7z". For 7000zs devices, add the SOC_VARIANT = "7zs" in petalinuxbsp.conf

There are no additional overrides for Zynq 7000 devices.

2. Zynq UltraScale+ MPSoC is shipped in four device variants. The available SOC_VARIANTs are:
 - "cg" - Zynq UltraScale+ MPSoC CG Devices
 - "eg" - Zynq UltraScale+ MPSoC EG Devices
 - "ev" - Zynq UltraScale+ MPSoC EV Devices
 - "dr" - Zynq UltraScale+ MPSoC RFSoC devices

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

Note: You have to explicitly set SOC_VARIANT = "cg" in `petalinuxbsp.conf` for "CG" devices.

3. Versal adaptive SoC is shipped only in one design variant currently. The available SOC_VARIANTs are:
 - "prime" - Versal Prime devices
 - "ai-core" - Versal ai-core devices
 - "ai-edge" - Versal ai-edge devices
 - "premium" - Versal premium devices
 - "hbm" - Versal H50 hbm enabled devices

Image Features

The contents of images generated by the OpenEmbedded build system can be controlled by the IMAGE_FEATURES and EXTRA_IMAGE_FEATURES variables that you typically configure in your image recipes. Through these variables, you can add several different predefined packages such as development utilities or packages with debug information needed to investigate application problems or profile applications.

To remove any default feature, add the following code in the `petalinuxbsp.conf`:

```
IMAGE_FEATURES:remove = "ssh-server-dropbear"
```

To add any new feature, add the following command in the `petalinuxbsp.conf`:

```
IMAGE_FEATURES:append = " myfeature "
```

Filtering RootFS Packages Based on License

The INCOMPATIBLE_LICENSE flag is used to control which packages are included in the final root file system configuration based on the license.

If you want to exclude packages based on license, you can edit the <plnx-proj-root>/project-spec/conf/petalinuxbsp.conf file. For example, set INCOMPATIBLE_LICENSE = "GPLv3", run the petalinux-build command.

Creating and Adding Patches For Software Components within a PetaLinux Project

To create and add patches for software components within a PetaLinux project, follow these steps:

1. Get the source code from git URL specified in meta-layers:

```
petalinux-devtool modify <recipe-name>
```

For example:

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

The previous command fetches the sources for the recipe and unpack them to a <plnx-proj-root>/components/yocto/workspace/sources/<recipe-name> directory and initialize it as a git repository if it is not already one.

2. Make the changes you want to make to the source.
3. Run a build to test your changes. You can run petalinux-build -c <recipename> or even build an entire image using petalinux-build incorporating the changes assuming a package produced by the recipe is part of an image. There is no need to force anything, the build system detects changes to the source and recompile as necessary.
4. Optional: Test your changes on the target.
5. Place your changes in the form of a patch to the PetaLinux project. To commit your changes, use the following commands.

```
git add <filename>
git commit -s
```

6. petalinux-devtool finish <recipe-name> <destination layer path> creates a patch for the committed changes in recipe sources directory.

For example:

```
petalinux-devtool finish linux-xlnx <plnx-proj-dir>/project-spec/meta-user
```

petalinux-devtool update-recipe linux-xlnx -a <destination layer path> creates a patch for the committed changes in recipe sources directory without removing the bbappend and source directory from the workspace directory.

For example:

```
petalinux-devtool update-recipe linux-xlnx -a <plnx-proj-dir>/project-spec/meta-user
```

7. Once you have finished working on the recipe, run `petalinux-devtool reset <recipe-name>` to remove the source directory for the recipe from workspace.

Known Issues for the Devtool Flow

When using `ext-local-src` for `linux-xlnx` and executing `petalinux-devtool finish linux-xlnx <layer path>` after making menuconfig changes, the file `bsp.cfg` is deleted from the `<plnx-proj-dir>/project/-spec/meta-user/recipes-kernel/linux/linux-xlnx` directory. However, the `<plnx-proj-dir>/project/-spec/meta-user/recipes-kernel/linux/linux-xlnx_%.bbappend` file still references this file in the 'KERNEL_FEATURES' variable. This causes an error when executing `petalinux-build/petalinux-build -c kernel`. To overcome this issue, create an empty `bsp.cfg` file in the `<plnx-proj-dir>/project/-spec/meta-user/recipes-kernel/linux/linux-xlnx` directory before building the project/kernel.

Adding Extra Users to the PetaLinux System

You can make the changes using the following steps:

1. Go to project: `petalinux-config -c rootfs` → **PetaLinux Rootfs Settings** → Add extra users.
2. Provide the users. To add extra users to the PetaLinux system, provide the user ID (userid) and password (passwd) separated by `:`; for multiple users, separate sets of user IDs and passwords using `;`.

Examples:

To add a passwd1 for user1:

```
user1:passwd1; or user1:passwd1
```

To add an empty passwd for the user1:

```
user1:
```

To add user1 and user2 with passwd1 and passwd2, respectively:

```
user1:passwd1;user2:passwd2;
```

To add an empty passwd for user1 and passwd2 for user2

```
user1:;user2:passwd2
```

Technical FAQs

Troubleshooting

This section details the common errors that appear, while working with the PetaLinux commands, and lists their recovery steps in detail.

For Yocto related information, see <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842475/Petalinux+Yocto+Tips>.

```
versal-net support:  
WARNING: versal-net: Versal-net is not supported in 2023.2
```

Note: We are not supporting versal-net builds in PetaLinux tool for this release.

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-<petalinux-version>/
build/tmp cannot be located on NFS.

When TMPDIR is on NFS, BitBake throws an error at the time of parsing. You can change it to local storage while creating the PetaLinux project. To do so, follow these steps:

1. Either run `petalinux-create -t project -s <PATH_TO_PETALINUX_PROJECT_BSP> --tmpdir <TMPDIR PATH>` or `petalinux-config`.
2. Provide any local storage by selecting **Yocto-settings → TMPDIR**.



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

Recipe Name has ' _ ' or Uppercase Letters or Starts with an Uppercase Letter

If the application name is `plnx_myapp`, BitBake throws an error. A version number has to be entered after `' _ '`. For example, `myapp_1` is an accurate application and module name.

To recover, `sstateclean` the application and delete it. Also, delete the following line in `<plnx-proj-root>/project-spec/meta-user/conf/user-rootfsconfig`.

```
CONFIG_plnx_myapp
```

If the application or library or module name has all uppercase letters or starting with an uppercase letter `MYAPP/Myapp`, BitBake throws a `do_package_qa` error.

To recover, `sstateclean` the application and delete it. Also, delete the line in `<plnx-proj-root>/project-spec/meta-user/conf/user-rootfsconfig`.



CAUTION! If the project path has special characters like `+, *, !` etc., the `petalinux-config` command fails to execute. For example: `/opt/petalinux+/xilinx-zc702-2`. To recover, do not use any special characters in the path.

Recover from Corrupted Terminal

When PetaLinux is exited forcefully by pressing `Ctrl+C` twice, the following error appears:

```
NOTE: Sending SIGTERM to remaining 1 tasks
Error in atexit._run_exitfuncs:
Traceback (most recent call last):
  File
  "<plnx-proj-root>/components/yocto/layers/core/bitbake/lib/bb/ui/k
notty.py", line 313, in finish
    self.termios.tcsetattr(fd, self.termios.TCSADRAIN, self.stdinbackup)
termios.error: (5, 'Input/output error')
```

After this error, the console is broken and you cannot see the text that you typed. To restore the console, enter `stty sane` and press `Ctrl+J` twice.

Python Language Settings

The following errors appear when the language settings are missing:

- Could not find the `/log/cooker/plnx_microblaze` in the `/tmp` directory during `petalinux-config`
- Please use a locale setting which supports UTF-8 (such as `LANG=en_US.UTF-8`).

Python cannot change the file system locale after loading. Therefore, you need a UTF-8 when Python starts, else it does not work.

```
ERROR: Failed to build project
```

To resolve the previous errors, set the following:

```
export LC_ALL=en_US.UTF-8  
export LANG=en_US.UTF-8  
export LANGUAGE=en_US.UTF-8
```

Menuconfig Hang for Kernel and U-Boot

For `petalinux-config -c`, sometimes when the kernel and U-Boot BitBake try to open a new terminal inside, they fail. The following are the possible error messages:

- ERROR: Unable to spawn new terminal
- ERROR: Continuing the execution without opening the terminal

The solutions can be:

- Use `ssh -X <hostname>`.
- Uncomment the `OE_TERMINAL` line in `<plnx-proj-root>/project-spec/meta-user/conf/petalinuxbsp.conf`. You can set any terminal which suits you (possible values could be auto, screen, tmux, xterm, and konsole). You have to change the `OE_TERMINAL` as it cannot get through default. For this, you must have the corresponding utility installed in your machine.

Menuconfig Not Seen for Kernel and U-Boot

Set `SHELL=/bin/bash` before issuing `petalinux-config -c kernel/ petalinux-config -c u-boot`.

Menuconfig Corruption for Kernel and U-Boot

When you issue `petalinux-config -c kernel/u-boot`, you might see a corrupted menu config. This is due to the terminal you are using.

To resolve, export other terminals like screen, xterm, konsole, putty, or gnome, and retry. For example:

```
export TERM=screen
```

Note: You do not see this issue on Ubuntu 18.x host machines.

External Source Configurations

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

Xen and openamp are handled through distro features. Adding distro features does not enable their corresponding configurations in kernel as they are handled in scc file. The solution is to edit <plnx-proj-root>/project-spec/meta-user/recipes-kernel/linux/linux-xlnx_%.bbappend.

Add the following lines:

```
SRC_URI += "file:///xilinx-kmeta/bsp/xilinx/xen.cfg"
```

To work with the scc files, replace their respective cfg files using external source methodology.

do_image_cpio: Function Failed

CPIO format does not support sizes greater than 2 GB. Therefore, you cannot use INITRAMFS for larger sizes. The following steps describes the process for larger image sizes (greater than 2 GB).

1. Change the root file system type to EXT4 (SD/eMMC/SATA/USB).

```
petalinux-config
```

Select **Image Packaging Configuration** → **Root filesystem type** → **EXT4 (SD/eMMC/SATA/USB)**.

2. Add the following lines in the <plnx-proj-root>/project-spec/meta-user/conf/petalinuxbsp.conf.

```
IMAGE_FSTYPES:remove = "cpio cpio.gz cpio.bz2 cpio.xz cpio.lzma cpio.lz4  
cpio.gz.u-boot"  
IMAGE_FSTYPES_DEBUGFS:remove = "cpio cpio.gz cpio.bz2 cpio.xz cpio.lzma  
cpio.lz4  
cpio.gz.u-boot"
```

3. Build the project.

```
petalinux-build
```

Note: Unlike earlier, currently PetaLinux does not generate the global DTS file. Use the following command to generate the global DTS file:

```
dts -I dtb -O dts -o system.dts system.dtb
```

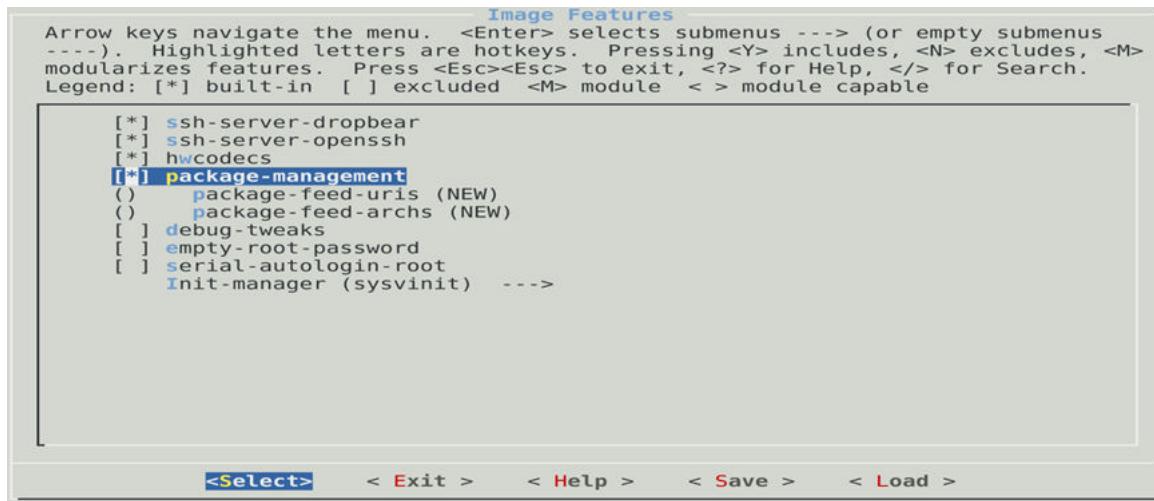


CAUTION! Do not use the symlinked path to the project directories for any build operations, including simply "cd"ing into the directory.

Package Management

PetaLinux supports package management system for Zynq 7000, Zynq UltraScale+ MPSoC, and AMD Versal™ devices. Use the following steps to configure and use the package management system:

Figure 37: Package Management



1. Enable DNF through petalinux-config -c rootfs. Enable the following configs to use DNF.
 - **Image Features → [*] package management**
 - No need to set the base package feed url in **Image features → package-management → package-feed-uris**.
 - No need to set the package feed architecture in **Image features → package management → package-feed-archs**.

From 2021.1, specifying package-feed-uris and package-feed-archs is optional. PetaLinux or Yocto set the uris and set the archs based on the project.

The possible archs are available in <https://petalinux.xilinx.com/sswreleases/rel-v2021.2/generic/rpm>

2. Build the project.

```
#petalinux-build
```

3. Boot Linux in SD or in JTAG boot mode.

4. Check for .repo file on target in /etc/yum.repos.d/ as follows:

```
[oe-remote-repo-sswreleases-rel-v2022-generic-rpm-noarch]
name=OE Remote Repo: sswreleases rel-v<PETALINUX_VER> generic rpm noarch
baseurl=http://petalinux.xilinx.com/sswreleases/rel-v<PETALINUX_VER>/
generic/rpm/noarch
gpgcheck=0

[oe-remote-repo-sswreleases-rel-v<PETALINUX_VER>-generic-rpm-aarch64]
name=OE Remote Repo: sswreleases rel-v<PETALINUX_VER> generic rpm aarch64
baseurl=http://petalinux.xilinx.com/sswreleases/rel-v<PETALINUX_VER>/
generic/rpm/aarch64
gpgcheck=0

[oe-remote-repo-sswreleases-rel-v<PETALINUX_VER>-generic-rpm-zynqmp]
name=OE Remote Repo: sswreleases rel-v<PETALINUX_VER> generic rpm zynqmp
baseurl=http://petalinux.xilinx.com/sswreleases/rel-v<PETALINUX_VER>/
generic/rpm/zynqmp
gpgcheck=0

[oe-remote-repo-sswreleases-rel-v<PETALINUX_VER>-generic-rpm-zynqmpeg]
name=OE Remote Repo: sswreleases rel-v<PETALINUX_VER> generic rpm
zynqmpeg
baseurl=http://petalinux.xilinx.com/sswreleases/rel-v<PETALINUX_VER>/
generic/rpm/zynqmpeg
gpgcheck=0

[oe-remote-repo-sswreleases-rel-v<PETALINUX_VER>-generic-rpm-
zynqmp-generic]
name=OE Remote Repo: sswreleases rel-v<PETALINUX_VER> generic rpm
zynqmp-generic
baseurl=http://petalinux.xilinx.com/sswreleases/rel-v2022/generic/rpm/
zynqmp-generic
gpgcheck=0
```

5. List all available packages.

```
#dnf repoquery
```

6. Install a specific package.

```
#dnf install <pkg name>
```

Example: #dnf install packagegroup-petalinux-matchbox

Once the matchbox package is installed, reboot the target and you should get the desktop environment.

Common Tar Usage

Get the BOOT.BIN from the Vitis software platform or the PetaLinux BSP prebuilt to boot Linux with common tar images. You can see runtime errors when hardware is not supported for the utilities/packages enabled in the common rootfs with respect to each platform. Example cases are as follows:

- An error occurs when the AI EngineAI Engine is accessed on the VMK180 board using the common rootfs because the VMK180 board does not support AI Engine.
- An error occurs when the VCU is accessed on the ZCU102 board using the common rootfs because the VCU is not supported on the ZCU102.
- Use the rootfs utilities/packages based on the hardware supported on the board.

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

When the `petalinux-boot` command is issued, the following warning message is displayed:

```
"Linux image size is large (${imgsize}). It can cause boot issues. Please refer to Technical FAQs. Storage based Root Filesystem is recommended for large images."
```

Note: If your INITRAMFS image size is 128 MB, use storage-based boot only. It is recommended to use only the INITD file system.

Detailed Component-based Logs

The `<plnx-proj-root>/build/build.log` log provides information on all the tasks that are running and the tasks that have failed.

For each component and for each task, there are detailed logs that you can find in the build or in the TMPDIR(NFS builds) directory.

- If you build on local space, you can find the component task wise logs in the build directory. For example: For the device-tree component, `do_configure` task, the log is located at `<plnx-proj-root>/build/tmp/work/versal_generic-xilinx-linux/device-tree/xilinx-v<petalinux-version>+gitAUTOINC+f725aaecff-r0/temp/log.do_configure.32730`.
- If you build on the NFS, you can find the log in `<TMPDIR>/work/versal_generic-xilinx-linux/device-tree/xilinx-<PETALINUX_VERSION>+gitAUTOINC+f725aaecff-r0/temp/log.do_configure.32730`. You can check the TMPDIR location in the `<plnx-proj-root>/project-spec/configs/config` file.

PetaLinux Environment Issues

Note: Remove environment variables in `.bashrc` file while sourcing the PetaLinux tool, otherwise you might see the following issue:

```
$ source /opt/pkg/petalinux/settings.sh
PetaLinux environment set to /opt/pkg/autolist'
bash: /opt/pkg/autolist/tools/common/petalinux//utils/petalinux-env-check:
No such file or directory
```

From the previous example 'autolist' variable in .bashrc file causing the issue, this issue is resolved by removing 'set autolist' variable from .bashrc file.

If you are facing such kind of environment issue in sourcing the PetaLinux tool, clean up .bashrc by removing the specific environment variables.

Adding User Layer Failed with BitBake Timeout

If PetaLinux project configure to fail user layer with the following BitBake error, your user layer PATH is exceeding limitation.

```
[ INFO] Adding user layers
NOTE: Starting bitbake server...
NOTE: No reply from server in 30s
NOTE: No reply from server in 30s
Timeout while waiting for a reply from the bitbake server (60s)
```

To overcome the timeout issue, use the path within 250 characters.

Password Recovery

Use the following steps to recover password:

1. Set "launch_ramdisk_init" U-Boot environment variable in uEnv.txt file

```
launch_ramdisk_init=1
```

2. Add the file into fat partition of sd/eMMC/usb boot devices

Note: Instead of previous variable, you can also set this `bootargs setenv bootargs 'ext4=/dev/null:/rootfs init_fatal_sh=1'`; in uEnv.txt or from U-Boot prompt before kernel handoff.

3. U-Boot reads the file and boot with tiny rootfs

4. On tiny rootfs, mount rootfs partition to directory

```
mount /dev/mmcblk0p2 /mnt/
```

5. Change the root directory for the current running process to the specified directory

```
chroot /mnt
```

6. Change the password

```
passwd petalinux
```

7. Sync and unmount partition

```
sync
exit
umount /dev/mmcblk0p2
```

8. Reset the board and try to login using updated password

Migration

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

PetaLinux Supported Dynamic Configuration

The fpgamanager template names got renamed, and the exact mapping is given in the following table.

Outdated templates remain functional, yet it is strongly recommended to adopt new template names.

Sr No	Design Type	Input Files	Yocto Class Name	PetaLinux Template Name	old Petalinux template name	platform
1	NA	dts/dtsi/dtbo, bit.bin or pdi	dfx_user_dts	dfx_user_dts	fpgamanager	zynq zynqmp versal
2	Flat	xsa or dtsi	dfx_dtg_zynq_full	dfx_dtg_zynq_full	fpgamanager_d_tg	zynq
3	Flat	xsa or dtsi	dfx_dtg_zynqmp_full	dfx_dtg_zynqmp_full	fpgamanager_d_tg	zynqmp
4	DFX static	xsa or dtsi	dfx_dtg_zynqmp_static	dfx_dtg_zynqmp_static	fpgamanager_d_tg	zynqmp
5	DFX RP	xsa or dtsi	dfx_dtg_zynqmp_partial	dfx_dtg_zynqmp_partial	fpgamanager_d_tg_dfx	zynqmp
6	Flat	xsa or dtsi	dfx_dtg_versal_full	dfx_dtg_versal_full	fpgamanager_d_tg_csoc2	versal
7	DFX static	xsa or dtsi	dfx_dtg_versal_static	dfx_dtg_versal_static	fpgamanager_d_tg	versal
8	DFX RP	xsa or dtsi	dfx_dtg_versal_partial	dfx_dtg_versal_partial	fpgamanager_d_tg_dfx	versal

Dynamic Configuration Support in PetaLinux

The Programmable Logic(PL) can be programmed using FSBL/PLM or from U-Boot or from Linux.

This section provides the details about programming/reprogramming the PL from Linux world using Linux FPGA Manager framework..

FPGA Manager Full PL Programming

Zynq 7000

The Zynq flat design has programmable logic (PL), and you want to defer the PL loading till Linux comes up, use the following steps to extract the PL bitstream and pack the dtbo and bitstream files into the `/lib/firmware/xilinx` directory in the root file system.

Prerequisites

Zynq 7000 flat design(xsa) with PL IPs in bitstream

Build Steps

Follow the steps to build and pack the full bitstream and its corresponding dtbo:

1. Source the PetaLinux tool

```
source /opt/petalinux/petalinux-v<petalinux-version>-final/settings.sh
```

2. Create a Zynq template project or bsp project

```
petalinux-create -t project -n zynq --template zynq  
petalinux-create -t project -s <bsp path> -n zynq
```

3. Go to the project

```
cd zynq
```

4. Configure the project with flat xsa if you create a project in template flow

```
petalinux-config --get-hw-description <flat.xsa>
```

5. In the petalinux-config command, select **FPGA Manager** → **[*] Fpga Manager**.

Note: The PetaLinux FPGA manager configuration performs the following:

- fpga-overlay Machine features
- Enables the required kernel configs to load the fpgamanager driver

6. Use the `petalinux-create` command to extract the full bitstream and its corresponding pl dtbo into the PetaLinux rootfs.

```
petalinux-create -t apps --template dfx_dtg_zynq_full -n can-interface --  
srcuri "<path-to-xsa>/system.xsa" --enable
```

Note: If you do not specify `--enable` the dtbo and its bitstream is generated in `<project>/build/tmp/deploy` directory.

7. To build as part of rootfs you can use the following commands:

```
petalinux-build  
petalinux-build -c rootfs
```

To build only the application:

```
petalinux-build -c
```

Boot Steps

Once the base target is up, run the following command:

```
fpgautil o /lib/firmware/can_interface/pl.dtbo b /lib/firmware/xilinx/can_interface/design_1_wrapper.bit.bin
```

ZynqMP

The ZynqMP design has programmable logic (PL), and you want to defer the PL loading till Linux comes up, use the following steps to extract the PL bitstream and pack the dtbo and bitstream files into the `/lib/firmware/xilinx` directory in the root file system.

Prerequisites

ZynqMP flat design(xsa) with PL IPs in bitstream

Build Steps

After creating a PetaLinux project, follow the steps to build and pack the full bitstream and its corresponding dtbo:

1. Source the PetaLinux tool

```
source /opt/petalinux/petalinux-v<petalinux-version>-final/settings.sh
```

2. Create a ZynqMP template project or bsp project

```
petalinux-create -t project -n zynqmp --template zynqmp  
petalinux-create -t project -s <bsp path> -n zynqmp
```

3. Go to the project

```
cd zynqmp
```

4. Configure the project with flat xsa if you create project in template flow

```
petalinux-config --get-hw-description <flat.xsa>
```

5. In the `petalinux-config` command, select **FPGA Manager** → **[*] Fpga Manager**.

Note: The PetaLinux FPGA manager configuration performs the following:

- fpga-overlay Machine features

- Enables the required kernel configs to load the fpgamanager driver
6. Use the `petalinux-create` command to extract the full bitstream and its corresponding pl dtbo into the PetaLinux rootfs.

```
petalinux-create -t apps --template dfx_dtg_zynqmp_full -n can-interface --srcuri "<path-to-xsa>/system.xsa" --enable
```

Note: If you do not specify `--enable` the dtbo and its bitstream is generated in `<project>/build/tmp/deploy` directory.

7. To build as part of rootfs you can use the following commands:

```
petalinux-build  
petalinux-build -c rootfs
```

To build only the application:

```
petalinux-build -c can-interface
```

Boot Steps

Once the base target is up, run the following command:

```
fpgautil -o /lib/firmware/can_interface/pl.dtbo -b /lib/firmware/xilinx/can_interface/design_1_wrapper.bit.bin
```

Versal (*Segmented Configuration Flow*)

Segmented Configuration flow is quickly booted on an operating system prior to PL loading; NoC + DDR is required in the first programming image. Defer PL loading indefinitely, with the ability to load PL PDI from any primary or secondary boot interface

Prerequisites

Versal flat design(xsa) contains boot pdi which includes Versal cips + NOC + DDR, and pld pdi, which has PL IPs

Build Steps

This section helps to build and boot segmented configuration design flow for Versal platforms.

1. Source the PetaLinux tool

```
source /opt/petalinux/petalinux-v<petalinux-version>-final/settings.sh
```

2. Create a Versal template project or BSP project

```
petalinux-create -t project -n versal-seg-flow --template versal  
petalinux-create -t project -s <bsp path> -n versal-seg-flow
```

3. Go to the project

```
cd versal-seg-flow
```

4. Configure the project with flat xsa if you create project in template flow

```
petalinux-config --get-hw-description <flat.xsa>
```

5. Enable FPGA manager, select **FPGA Manager** → **[*] Fpga Manager**.

Note: The PetaLinux FPGA manager configuration performs the following:

- fpga-overlay Machine features
- Enables the required kernel configs to load the fpgamanager driver

6. Create the pl application to pack the pl pdi and the pl ips dtbo into rootfs command:

```
petalinux-create -t apps --template dfx_dtg_versal_full -n pl-app --enable --srcuri "<path>/rprm.xsa"
```

The previous command generates and packages the rm dtbo, pl pdi files into the rootfs(/lib/firmware/xilinx/<pl-app>)

7. To build the application, use the following command. The following command generates the rootfs containing the rprm, dtbo, and respective pdi files:

```
petalinux-build  
petalinux-build -c rootfs
```

To build only the application:

```
petalinux-build -c <pl-app>
```

8. To boot target with PS generate the boot.bin using the following command:

```
petalinux-package --boot --u-boot (It will pick *_soc.pdi) and packs as part of the boot.bin
```

Boot Steps

Once the base target is up, run the following command:

```
fpgautil -o /lib/firmware/can_interface/pl.dtbo -b /lib/firmware/xilinx/pl-app/hw_description_pld.pdi
```

Dynamic Functional Exchange (DFX) Flow

Versal

The FPGA manager provides an interface to Linux for reconfiguring the programmable region. It packs the dtbos and bitstreams/pdi files into the /lib/firmware/xilinx directory in the root file system.

This section helps to build and boot PL applications with DFX design for the Versal platform.

Note: If DFX applications (rm.xsa) have any memory-mapped PL IPs, only then use fpgamanager_dtg_dfx template. If not, you can use --template install to pack only pdi as part of rootfs.

Prerequisites

- Versal static design(static xsa) contains static pdi which includes Versal cips + NOC + DDR + PL(static PL + empty RP regions)
- Versal reconfigurable module design(rm.xsa) which includes pdi with ips that we want to program
- You can have more than one rm xsa file

Build Steps

1. Source the PetaLinux tool.

```
source /opt/petalinux/petalinux-v<petalinuxversion>/settings.sh
```

2. Create a Versal template project or BSP project

```
petalinux-create -t project -n versal-dfx --template versal  
petalinux-create -t project -s <bsp path> -n versal-dfx
```

3. Go to the project

```
cd versal-dfx
```

4. Configure the project with static.xsa/base.xsa if you have created the project using the template flow

```
petalinux-config --get-hw-description <base.xsa/static.xsa>
```

5. Enable FPGA manager using the following command:

```
petalinux-config -> FPGA MANAGER
```

Note: The PetaLinux FPGA manager configuration performs the following:

- fpga-overlay Machine features
- Enables the required kernel configs to load the fpgamanager driver

6. Create the static application using the following command from static xsa

```
petalinux-create -t apps --template dfx_dtg_versal_static -n <static-app> --enable --srcuri "<static xsa>"
```

The previous command creates and packages the static dtbo and pdi files into the rootfs (/lib/firmware/xilinx/) using dfx_dtg_versal_static template.

7. Create the partial application to configure the partial region using the rm xsa in the following command. You should point static pl app name as `--static-pn` command line option to define the relation between base and partial.

```
petalinux-create -t apps --template dfx_dtg_versal_partial -n <rm-app>
--enable --srcuri <rm.xsa>" --static-pn <static-app>
```

This command generates and packages the rm dtbo, pdi files into the rootfs (`/lib/firmware/xilinx/<static-app>/<rm-app>`).

8. To build the application, use the following command. The following command generates the rootfs containing both static and rprm dtbos, and respective pdi files.

```
petalinux-build or petalinux-build -c rootfs
```

To build only the application

```
petalinux-build -c <static-app>
petalinux-build -c <rm-app>
```

In `<TMPDIR>/deploy/rpm` you can see `<static-app>.rpm` and `<rm-app>.rpm`

Note: The pdi's in the design should have `i*_partial.pdi` in the xsa files to avoid an error.

Note: In the DFX use case, you can use the static xsa to create the Versal boot firmware images so static pdi is packaged as a part of `BOOT.BIN`.

Boot Steps

Up the target with previously built images using any of the boot methods in the documentation. Once the target is up, run the following commands.

```
fpgautil -o /lib/firmware/xilinx/<static-app>/<static-app>.dtbo
fpgautil -b /lib/firmware/xilinx/<staticapp>/rp0/<rprm-app>/<rprm-app>.pdi
-o /lib/firmware/xilinx/<staticapp>/rp0/<rprm-app>/<rprm-app>.dtbo -f
Partial -n PRO
```

ZynqMP

The FPGA manager provides an interface to Linux for reconfiguring the programmable region later. It packs the dtbos and bitstreams files into the `/lib/firmware/xilinx` directory in the root file system.

This section helps to build and boot DFX design for the ZynqMP platform.

Note: If DFX applications(rm.xsa) have any memory mapped PL IPs only then use `fpgamanager_dtg_dfx` template. If not you can use `--template install` to pack only pdi as part of rootfs.

Prerequisites

- ZynqMP static design(static xsa) contains static bitream which includes static PL IPs and reprogrammable regions

- ZynqMP reconfigurable module design(rm.xsa) which includes bitstream with ips that you want to program
- You can have more than one rm xsa files

Build Steps

1. Source the PetaLinux tool.

```
source /opt/petalinux/petalinux-v<petalinuxversion>/settings.sh
```

2. Create a ZynqMP template project or BSP project

```
petalinux-create -t project -n zynqmp-dfx --template zynqmp  
petalinux-create -t project -s <bsp path> -n zynqmp-dfx
```

3. Go to the project

```
cd zynqmp-dfx
```

4. Configure the project with static.xsa/base.xsa

```
petalinux-config --get-hw-description <base.xsa/static.xsa>
```

5. Enable FPGA manager using the following command:

```
petalinux-config -> FPGA MANAGER
```

Note: The PetaLinux FPGA manager configuration performs the following:

- fpga-overlay Machine features
- Enables the required kernel configs to load the fpgamanager driver

6. Create the static application using the following command from static xsa

```
petalinux-create -t apps --template dfx_dtg_zynqmp_static -n <static-app> --enable --srcuri "<static xsa>"
```

The previous command creates and packages the static dtbo and pdi files into the rootfs (/lib/firmware/xilinx/) using dfx_dtg_zynqmp_static template.

7. Create the partial application to configure the partial region using the rm xsa in the following command. You should point static pl app name as --static-pn command line option to define the relation between base and partial.

```
petalinux-create -t apps --template dfx_dtg_zynqmp_partial -n <rm-app>  
--enable --srcuri <rm.xsa> --static-pn <static-app>
```

This command generates and packages the rm dtbo, pdi files into the rootfs (/lib/firmware/xilinx/<static-app>/<rm-app>).

8. To build the application, use the following command. The following command generates the rootfs containing both static and rpm dtbos, and respective pdi files.

```
petalinux-build or petalinux-build -c rootfs
```

To build only the application

```
petalinux-build -c <static-app>
petalinux-build -c <rm-app>
```

In <TMPDIR>/deploy/rpm you can see <static-app>.rpm and <rm-app>.rpm

Note: The pdi's in the design should have i*_partial.pdi in the xsa files to avoid an error.

Boot Steps

Up the target with previously built images using any of the boot methods in the documentation. Once the target is up, run the following commands.

```
fpgautil -o /lib/firmware/xilinx/<static-app>/<static-app>.dtbo
fpgautil -b /lib/firmware/xilinx/<staticapp>/rp0/<rprm-app>/<rprm-app>.pdi
-o /lib/firmware/xilinx/<staticapp>/rp0/<rprm-app>/<rprm-app>.dtbo -f
Partial -n PRO
```

Dynamic Configuration with User Specified dtci/dtbo Flow for Zynq/ZynqMP/Versal

This section provides information on the mechanism and infrastructure required to work with custom IPs that have readily (hand-stitched) available dtci files instead of relying on the DTG to generate them when the FPGA manager is enabled. This generates/packs the dtbo and bin/pdi files into the rootfs /lib/firmware/xilinx directory.

Prerequisites

- dtci and its corresponding pdi/bit.bin or
- dtbo and its corresponding pdi/bit.bin files

Build Steps

1. Source the PetaLinux tool.

```
source /opt/petalinux/petalinux-v<petalinuxversion>/settings.sh
```

2. Create a Versal template project or BSP project

```
petalinux-create -t project -n zynqmp-dfx --template versal
petalinux-create -t project -s <bsp path> -n zynqmp
```

3. Go to the project

```
cd zynqmp-dfx
```

4. Configure the project with static.xsa/base.xsa

```
petalinux-config --get-hw-description <base.xsa/static.xsa>
```

5. Enable FPGA manager using the following command:

```
petalinux-config -> FPGA MANAGER
```

Note: The PetaLinux FPGA manager configuration performs the following:

- fpga-overlay Machine features
- Enables the required kernel configs to load the fpgamanager driver

6. Create the FPGA manager template

```
petalinux-create -t apps --template dfx_user_dts -n can-interface --  
srcuri "can.dtsi can.bit.bin" --enable
```

Or

```
petalinux-create -t apps --template dfx_user_dts -n can-interface --  
srcuri "can.dtbo can.bit.bin" --enable
```

Or

```
petalinux-create -t apps --template dfx_user_dts -n can-interface --  
srcuri "can.dtsi can.pdi" --enable
```

Or

```
petalinux-create -t apps --template dfx_user_dts -n can-interface --  
srcuri "can.dtbo can.pdi" --enable
```

Or

```
petalinux-create -t apps -template dfx_user_dts -n usrapp -scruri  
"user.dts user1.dtsi user2.dtsi system.pdi" -enable
```

7. Build the application.

If you want to build as part of rootfs

```
petalinux-build
```

Or

```
petalinux-build -c rootfs
```

If you want to build only the application.

```
petalinux-build -c can interface
```

Boot Steps

Up the target with previously built images using any of the boot methods in the documentation. Once the target is up, run the following commands.

```
fpgautil -o /lib/firmware/can_interface/pl.dtbo -b /lib/firmware/xilinx/can  
interface/design_1_wrapper.bit.bin
```

Customizing the Generated Dynamic Configuration PL dtsi Files

If you have any custom user dtsi files to this PL applications or PL properties, you can refer to this section

- For Zynq full PL programming if you have a custom pl dtsi file, provide the custom dtsi file to the srcuri and it is used while generating pl.dtbo file

```
petalinux-create -t apps --template dfx_dtg_zynq_full -n can-interface --srcuri "<path-to-xsa>/system.xsa pl-custom.dtsi" --enable
```

- For ZynqMP full PL programming, if you have a custom pl dtsi file, provide the custom dtsi file to the srcuri and it is used while generating pl.dtbo file

```
petalinux-create -t apps --template dfx_dtg_zynqmp_full -n can-interface --srcuri "<path-to-xsa>/system.xsa pl-custom.dtsi" --enable
```

- For Versal full PL programming, if you have a custom pl dtsi file, provide the custom dtsi file to the srcuri and it is used while generating pl.dtbo file

```
petalinux-create -t apps --template dfx_dtg_versal_full -n can-interface --srcuri "<path-to-xsa>/system.xsa pl-custom.dtsi" --enable
```

- For ZynqMP DFX programming if you have a custom pl dtsi file for static PL, provide the custom dtsi file to the srcuri and it is used while generating pl.dtbo file

```
petalinux-create -t apps --template dfx_dtg_zynqmp_static -n <static-app> --enable --srcuri "<static xsa> pl-custom.dtsi"
```

- For ZynqMP DFX programming, if you have a custom pl dtsi file for rm PL application, provide the custom dtsi file to the srcuri, and it is used while generating pl-partial-rm.dtbo file

```
petalinux-create -t apps --template dfx_dtg_zynqmp_partial -n <rm-app> --enable --srcuri <rm.xsa> pl-partial-custom.dtsi" --static-pn <static-app>
```

- For Versal DFX programming, if you have a custom pl dtsi file for static PL, provide the custom dtsi file to the srcuri and it is used while generating pl.dtbo file

```
petalinux-create -t apps --template dfx_dtg_versal_static -n <static-app> --enable --srcuri "<static xsa> pl-custom.dtsi"
```

- For Versal DFX programming, if you have a custom pl dtsi file for rm PL application, provide the custom dtsi file to the srcuri, and it is used while generating pl-partial-rm.dtbo file

```
petalinux-create -t apps --template dfx_dtg_versal_partial -n <rm-app> --enable --srcuri <rm.xsa> pl-partial-custom.dtsi" --static-pn <static-app>
```

Examples

1.

```
petalinux-create -t apps -template dfx_user_dts -n usrapp -scruri "user.dtsi system.bit" -enable
```

2. petalinux-create -t apps -template dfx_user_dts -n usrapp -scruri "user.dts user1.dtsi user1.dtsi system.bit" -enable
3. petalinux-create -t apps -template dfx_user_dts -n usrapp -scruri "user.dtsi system.pdi" -enable
4. petalinux-create -t apps -template dfx_user_dts -n usrapp -scruri "user.dts user1.dtsi user2.dtsi system.pdi" -enable
5. petalinux-create -t apps -template dfx_user_dts -n usrapp -scruri "user.dtbo system.bit.bin" -enable
6. petalinux-create -t apps -template dfx_user_dts -n usrapp -scruri "user.dtbo system.pdi" -enable
7. petalinux-create -t apps -template dfx_user_dts -n usrapp -scruri "user.dtbo" -enable
8. petalinux-create -t apps -template dfx_dtg_zynq_full -n usrapp -scruri "flat.xsa" -enable
9. petalinux-create -t apps -template dfx_dtg_zynq_full -n usrapp -scruri "flat.xsa custom.dtsi" -enable
10. petalinux-create -t apps -template dfx_dtg_zynqmp_full -n usrapp -scruri "flat.xsa" -enable
11. petalinux-create -t apps -template dfx_dtg_zynqmp_full -n usrapp -scruri "flat.xsa custom.dtsi" -enable

Login Changes

Follow the steps for login change:

1. serial-autologin-root option in PetaLinux tool is disabled by default. If required, you can enable this with **\$ petalinux-config -c rootfs → Image Features → [] serial-autologin-root**.
2. The root login is disabled by default. The default user is **petalinux** and the password should be set on first boot. If you want to use ssh login, you should first boot via console and need to set password.
3. Yocto removed plain text passwords support. The password should be in salt form. PetaLinux converts the provided password in config to static salt password using openssl and add to **plnxtool.conf**.

Config settings to set user and password:

```
petalinux-config -c rootfs → PetaLinux RootFS Settings →  
(root:root;petalinux:petalinux:passwd-expire;)
```

In plnxtool.conf EXTRA_USERS_PARAMS looks like:

```
EXTRA_USERS_PARAMS = "usermod -p '\$6\$xx\$CkLCyUiVJPvNL9C/gQans7jV3BvnVoW60raOcEE.KlziPn8pPMe8WvAQ3cPwwO4YRcYRBjlUhSfw/kaubhpt1' root; \
useradd -p '\$6\$xx\$12fBVQgrkB6ZmS6DSUIeVLW3Cot/hC4g8ZhbcwLZtKG93spibFJ/.0rF688RnsqMrgIbnHfkZnJu0Tk9.7F41' petalinux;passwd-expire petalinux; \"
```

Note: Display WARNING message when 'root' password is set to 'root'. **Warning: Root password set to 'root', It is highly recommended to change Root password.**

4. Add config support to specify **passwd-expire**.

```
petalinux-config -c rootfs --> PetaLinux RootFS Settings --> (root:root;petalinux:petalinux:passwd-expire;) Add Extra Users
```

5. Removed the ROOTFS_PASSWD configuration option. Use EXTRA_USERS configuration to specify the root user password.
6. You can skip the root password using **petalinux-config -c rootfs -> Image Features -> empty-root-password**

Systemv to Systemd

SYSVINIT

- systemv (sysv) is one of the first and traditional init systems for the UNIX/Linux operating system.
- If you are starting and stopping init services using /etc/init.d/sshd start, you Are on a SystemVinit system.

SYSTEMD

- systemd is a new init system and service manager for Linux operating systems.
- If you are starting and stopping things using systemctl restart sshd, you are on a SystemD system.

By default SYSTEMD enabled as init-manager for AMD Zynq™ UltraScale+™ MPSoC and AMD Versal™ platforms, SYSVINIT enabled as init-manager for Zynq and MicroBlaze™ platforms.

Note: Customers using their own init-script should modify as required to work with SYSTEMD. In old releases, it used to be SYSVINIT.

To Switch between sysv and systemd use the following config:

```
petalinux-config -c rootfs, select Image Features -> Init-manager (systemd) save and exit.
```

Run **petalinux-build**.

Yocto Override Syntax Changes

The [:]colon character replaces the use of [_] underscore referring to an override.

Check the link for more information<https://docs.yoctoproject.org/migration-guides/migration-3.4.html#override-syntax-changes>.

Yocto Recipe Name Changes

Yocto updated the following recipe names from 2021.1:

Table 36: Updated Yocto Recipe Names

Old Recipe File Name	New Recipe File Name
fsbl.bb	fsbl-firmware.bb
plm.bb	plm-firmware.bb

If you are creating or using any bbappends for these components, you need to update them.

For example:

```
mkdir <plnx-proj-root>/project-spec/meta-user/recipes-bsp/embeddedsw/
touch <plnx-proj-root>/project-spec/meta-user/recipes-bsp/embeddedsw/fsbl-
firmware_%.bbappend
Add recipe changes into fsbl-firmware_%.bbappend file.
```

U-Boot Image Changes

From 2021.1, PetaLinux removed DTB from `u-boot.elf`. U-Boot now uses the DTB from `BOOT.BIN` instead of `u-boot.elf`. If you are using a custom BIF file to generate `BOOT.BIN`, include the DTB in the file so that the boot does not fail.

Table 37: U-Boot Files Deployed by PetaLinux

File name	Description
<code>u-boot.elf</code>	Self-extractable U-Boot elf containing the U-Boot binary symbols with a larger size than previous versions
<code>u-boot-dtb.elf</code>	Self-extractable U-Boot elf that has the U-Boot binary with DTB
<code>u-boot.bin</code>	Contains the U-Boot binary
<code>u-boot-dtb.bin</code>	Contains the U-Boot binary with DTB

By default, Petalinux uses `u-boot.elf` to create JTAG/BOOT.BIN. During BOOT.BIN creation, Bootgen removes the symbols from `u-boot.elf` so that there is no size difference in the final BOOT.BIN. You can also use `u-boot-dtb.elf` for BOOT.BIN. In that case, U-Boot uses the DTB from `u-boot-dtb.elf` instead of BOOT.BIN.

MCS File Support

From 2021.2, Petalinux supports generating MCS file for AMD Zynq™, Zynq UltraScale+ MPSoC, and AMD Versal™ platforms using Bootgen utility. This enables you to program the MCS file directly to a target board and boot. The output MCS file can be found in `<plnx-proj-root>/images/linux/` with the name `boot.mcs`.

The following is the command to generate the MCS file:

```
petalinux-package --boot --u-boot --kernel --offset <Fit Image Flash Offset Address> --format MCS --force
```

For more details see Generate MCS Image in [Packaging Boot Image](#) for specific platform.

BIF File Changes

In releases prior to 2021.2, Petalinux used to maintain the `bootgen.bif` file in the `<plnx-proj-root>/build` directory and the images path pointed to `/tmp/tmp-xxx/<imagename>`. From 2022.2, the `bootgen.bif` file is generated in the `<plnx-proj-root>/images/linux` folder with the exact image paths from where you are pointing.

The following is the BIF file generated with the old releases for AMD Zynq™ UltraScale+™ MPSoC:

```
the_ROM_image:
{
[bootloader, destination_cpu=a53-0] /tmp/tmp.wpgCyCGXpV/zynqmp_fsbl.elf
[pmufw_image] /tmp/tmp.wpgCyCGXpV	pmufw.elf
[destination_device=p1] /tmp/tmp.wpgCyCGXpV/project_1.bit
[destination_cpu=a53-0, exception_level=el-3, trustzone] /tmp/
tmp.wpgCyCGXpV/bl31.elf
[destination_cpu=a53-0, load=0x00100000] /tmp/tmp.wpgCyCGXpV/system.dtb
[destination_cpu=a53-0, exception_level=el-2] /tmp/tmp.wpgCyCGXpV/u-boot.elf
}
```

The following is the BIF file generated with the 2023.2 release for Zynq UltraScale+ MPSoC:

```
the_MCS_image:  
{  
[bootloader, destination_cpu=a53-0] <plnx-proj-root>/images/linux/  
zynqmp_fsbl.elf  
[pmufw_image] <plnx-proj-root>/images/linux/pmufw.elf  
[destination_device=pl] <plnx-proj-root>/project-spec/hw-description/  
project_1.bit  
[destination_cpu=a53-0, exception_level=el-3, trustzone] <plnx-proj-root>/  
images/linux/b131.elf  
[destination_cpu=a53-0, load=0x00100000] <plnx-proj-root>/images/linux/  
system.dtb  
[destination_cpu=a53-0, exception_level=el-2] <plnx-proj-root>/images/  
linux/u-boot.elf  
}
```

Switch_root in petalinux-config

The switch_root is a utility used to switch from one root file system to another file system as the root of the mount tree. It is primarily used for initramfs/initrd cases.

In PetaLinux Zynq UltraScale+ MPSoC and Versal BSPs, the default menuconfig option for **petalinux-config → Image Package Configuration → INITRAMFS/INITRD image name** is set to `petalinux-initramfs-image` which is treated as switch_root enabled. The respective config image is packaged in the `image.ub` file and is also copied into the `images/linux` folder with `ramdisk` prefixed to the file name (for example, `ramdisk.cpio.gz.u-boot`), if the image specified in the config option name contains the initramfs.

The default generated `ramdisk.cpio.gz.u-boot` is a tiny-based search of the root file system for the EXT partition in SD/eMMC that is used as a real time rootfs file system irrespective of boot mode. If the init script in the tiny rootfs system does not find any rootfs in real time, it lands in to the tiny-based rootfs with the '`/#`' prompt.

Figure 38: Tiny Root File System Prompt

```
[    8.310363] Run /init as init process
udhcpc: started, v1.32.0
[    9.025699] macb ff0c0000.ethernet eth0: PHY [ff0c0000.etherr
[    9.026522] macb ff0c0000.ethernet eth0: configuring for phy/
udhcpc: sending discover
[    9.098489] macb ff0c0000.ethernet eth0: Link is Up - 1Gbps/F
[    9.099420] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes
udhcpc: sending discover
udhcpc: sending select for 10.0.2.15
udhcpc: lease of 10.0.2.15 obtained, lease time 86400
/etc/udhcpc.d/50default: Adding DNS 10.0.2.3
ERROR: There's no '/dev' on rootfs.

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

For example, if you are in the QSPI boot mode and you have flashed the `BOOT.BIN`, `boot.scr`, and `image.ub` files, it boots up to tiny rootfs and searches for the ext2/3/4 to mount and use it as real rootfs.

The default Zynq UltraScale+ MPSoC and Versal BSP prebuilt/built images are configured as mentioned previously to facilitate the larger rootfs support in PetaLinux. In TEMPLATE projects, `switch_root` is disabled by default starting from the PetaLinux 2022.1 release.

Added Distroboot Support for MicroBlaze Processors

Distroboot support for MicroBlaze processors has been added. Be careful when loading the `boot.scr` file into the DDR/Flash as per the bootmode. You can check the images offsets in the `boot.scr`.

Note: Use the `cat` command to check the `boot.scr` file. Using the `vi`, `vim`, and `gvim` commands to check might corrupt the file causing the boot to fail.

Use Yocto Environment Variables in petalinux-config Option

From this release, PetaLinux supports using all Yocto variables in `petalinux-config`, for example, `TMPDIR`. To use the shell variables, you need to export the shell variable to `BB_ENV_PASSTHROUGH_ADDITIONS`. You cannot use normal shell variables directly to align with Yocto like the previous versions.

```
export BB_ENV_PASSTHROUGH_ADDITIONS= "$BB_ENV_PASSTHROUGH_ADDITIONS  
BB_USERNAME"
```

Note: In the previous example, the host machine exports the user environment variable by default.

```
export BB_USERNAME=$(whoami)  
export BB_ENV_PASSTHROUGH_ADDITIONS= "$BB_ENV_PASSTHROUGH_ADDITIONS  
BB_USERNAME"
```

Note: In the previous example, `BB_USERNAME` is a new environment variable defined and exported to the `BB_ENV_PASSTHROUGH_ADDITIONS` Yocto variable. After it is exported to `BB_ENV_PASSTHROUGH_ADDITIONS`, you can use `BB_USERNAME` for the `petalinux-config` options

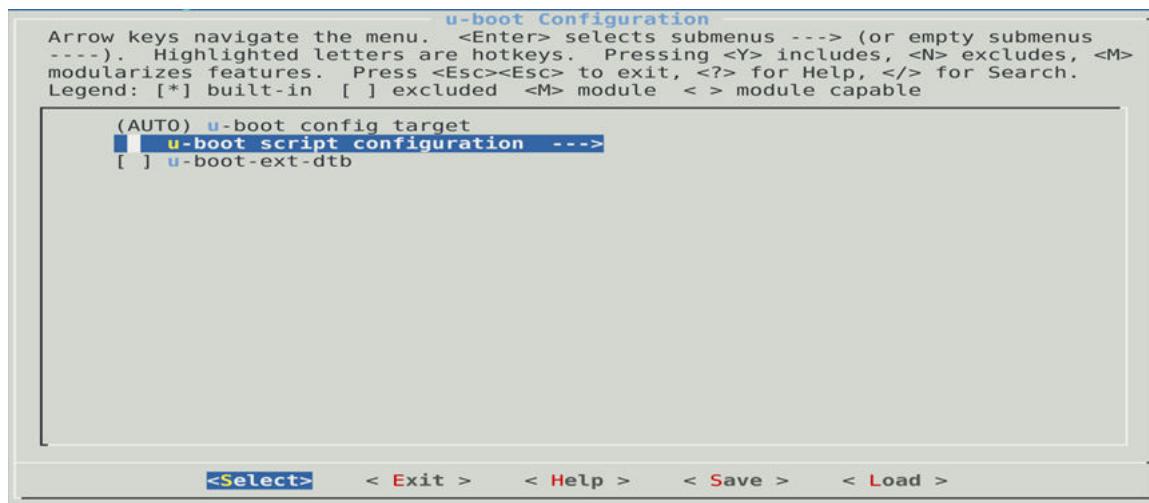
Removed Webtalk From PetaLinux

From 2021.2, PetaLinux removed Webtalk content. This was used to send the PetaLinux usage statistics and OS specifications to AMD servers.

U-Boot Configuration Changes

Added a new menuconfig option to check/modify the default bootscript images offsets or images names at **petalinux-config → u-boot Configuration → u-boot script configuration**. The modified offsets are added into the final `boot.scr` that is generated using `petalinux-build` command. For more details, see [Configuring U-Boot Boot Script \(boot.scr\)](#).

Figure 39: U-Boot Script Configuration



Changes to petalinux-boot Command

The following `petalinux-boot -qemu / --jtag` command line options which are broken in past releases are fixed in this release.

- `petalinux-boot -qemu/--jtag --kernel/--u-boot`
- `petalinux-boot -qemu/--jtag --dtb`
- `petalinux-boot -qemu/--jtag --kernel/--u-boot --pmufw` for Zynq UltraScale+ MPSoC.
- `petalinux-boot --qemu/--jtag --kernel --rootfs <specify custom cpio.gz.u-boot rootfs path>`

Note: `--rootfs` with `petalinux-boot --qemu` does not work for Versal platforms starting from 2022.1. Since `rootfs` is part of the `qemu_boot.img` file, you need to repack the image file using `petalinux-package --boot --u-boot --qemu-rootfs rootfs.cpio.gz.u-boot` and run the `petalinux-boot --qemu --kernel` command to use `rootfs.cpio.gz.u-boot`.

- Specify this option to disable GDB via QEMU boot:

```
petalinux-boot --qemu --prebuilt 2/--prebuilt 3 --qemu-no-gdb
petalinux-boot --qemu --u-boot/--kernel --qemu-no-gdb
```

FPGA Manager Changes

When you create fpgamanager or fpgamanager_dtg template apps using the petalinux-create command, ensure to enable FPGA Manager. Not enabling FPGA Manager could cause issues in loading the dtbo and bin files on target.

petalinux-config → FPGA Manager → [*] Fpga Manager

1. Generating dtbo from pl.xsa and packaging them in rootfs

```
petalinux-create -t apps --template fpgamanager_dtg -n gpio --enable --  
srcuri "<path>/gpio.xsa <path>/shell.json"
```

2. Generating dtbo and its corresponding pdi/bit.bin from dtsi file using the following command:

```
petalinux-create -t apps --template fpgamanager -n gpio --enable --  
srcuri "<path>/pl.dtsi <path>/system.bit <path>/shell.json"
```

3. Packaging prebuilt overlay dtbo file and its corresponding pdi/bit.bin file.

```
petalinux-create -t apps --template fpgamanager -n gpio --enable --  
srcuri "<path>/pl.dtbo <path>/system.bit.bin/<path>/pdi" (code section)
```

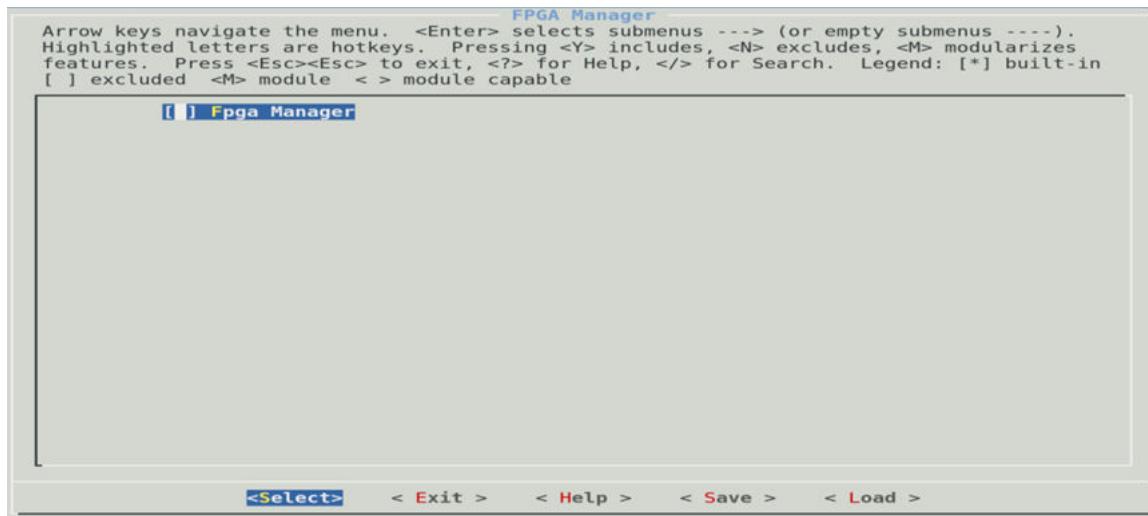
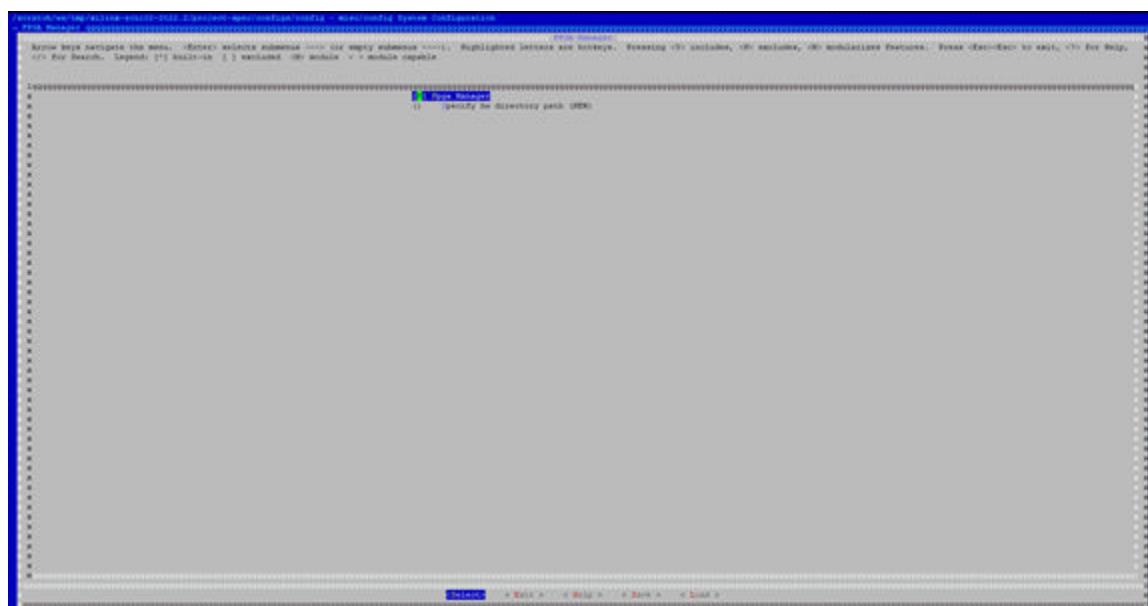
You can find the dtbo and bin/pdi files of the generated base and template applications in /lib/firmware/xilinx/

When Fpga Manager Configuration is enabled :

1. FPGA manager kernel configs gets enabled
2. Dtoverlay in DTG is enabled
3. Yocto FPGA manager plugins are enabled

fpga_manager_util is deprecated. You can generate dtbo and its bit.bin/pdi using fpgamanager_dtg template.

This can be achieved using fpgamanager_dtg class.

Figure 40: FPGA Manager*Figure 41: In 2022.1*

Host GCC Version Upgrade

From this release, the gcc version should be greater than 6. Using lower versions of gcc causes build issues. You can also enable the **Enable buildtools extended** option from **petalinux -config → Yocto** settings, which uses the pre-compiled gcc binaries from the PetaLinux tool.

Figure 42: GCC Version Upgrade

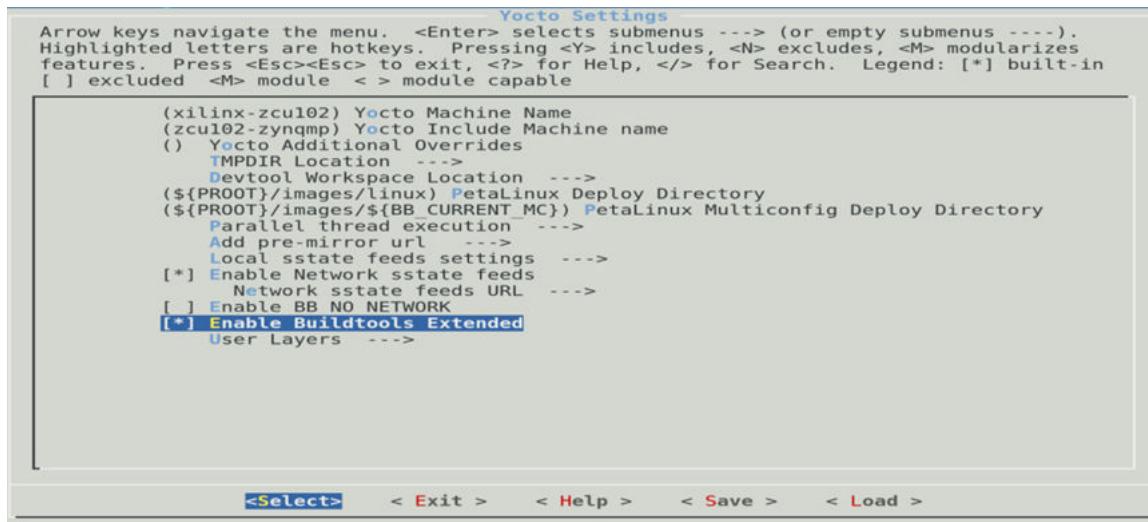


Image Selector

The Image Selector (ImgSel) is a bare-metal application running out of OCM after a POR/SRST to select the correct boot image. Image Selector is a generic application designed to share across different ZU+ boards.

Image Selector currently supports the selection of boot images based on the following parameters for different boards. You can enable this option in PetaLinux using the following: `petalinux-config → Linux Components Selection → [] Image Selector`

- Board EEPROM data
- QSPI A/B update register data

Note: Image Selector is optional and only required if the boot image has to be selected based on Board EEPROM data or QSPI A/B mechanism.

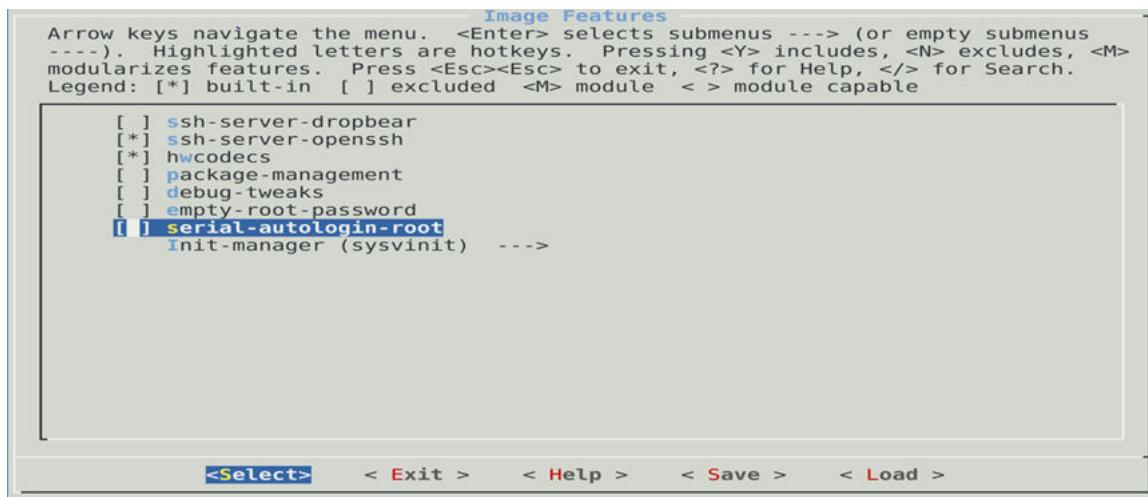
Board/Board Variant Changes

The `petalinux-config → yocto settings → yocto board settings → YOCTO_BOARD_NAME/YOCTO_BOARD_VARIANT_NAME` option is removed.

To align with upstream Yocto, use Yocto machine variables only.

Removed Auto Login

Figure 43: Auto Login



To align with upstream Yocto, auto login configuration is removed from the 2023.1 release.

By default, PetaLinux users have sudo access. You can use `petalinux-config -c rootfs -> Image features -> serial_autologin_root` to achieve the same.

Removed platform-auto.h

Earlier, for MicroBlaze platforms, there was a need to define the static configuration and design-based configuration in `platform-auto.h` in `<proot>/project-spec/configs/u-boot-xlnx` directory. Now most of the not required configs are removed and required configs are in `microblaze_generic.h`.

Changes in Machine Conf

In 2022.2 release, yocto machine conf files were in PetaLinux tool.

```
/opt/petalinux-v2022.2-final/components/misc/yocto_machines/
avnet-digilent-zedboard.conf xilinx-k26-kv.conf xilinx-ultra96-
rev.aconf xilinx-vek280.conf xilinx-zc702.conf xilinx-zcu1275.conf
eval-brd-sc.conf xilinx-k26-som.conf xilinx-
vck190.conf xilinx-vhk158.conf xilinx-zc706.conf xilinx-
zcu1285.conf xilinx-ac701.conf xilinx-kc705.conf xilinx-vck190-
```

```
sc.conf      xilinx-vmk180.conf      xilinx-zcu102.conf  xilinx-zcu208.conf  
xilinx-k24-kd.conf      xilinx-kcu105.conf      xilinx-  
vck5000.conf      xilinx-vpk120.conf      xilinx-zcu104.conf  xilinx-  
zcu216.conf  
xilinx-k24-som.conf      xilinx-kcu105-tmr.conf  xilinx-vc-p-  
a2197-00.conf  xilinx-vpk180.conf      xilinx-zcu106.conf  xilinx-  
zcu670.conf  
xilinx-k26-kr.conf      xilinx-sp701.conf      xilinx-  
vcu118.conf      xilinx-zc1751-dc1.conf  xilinx-zcu111.conf
```

From 2023.1 release, PetaLinux is not maintaining any static machine conf files. This is generated dynamically, and it is available in the machine conf file located in <plnx-proj>/build/conf/machine based on xsa.

Removed config.project

Removed config.project metadata file from PetaLinux, as the metadata is in petalinux/metadata file.

Usage of uenv.txt

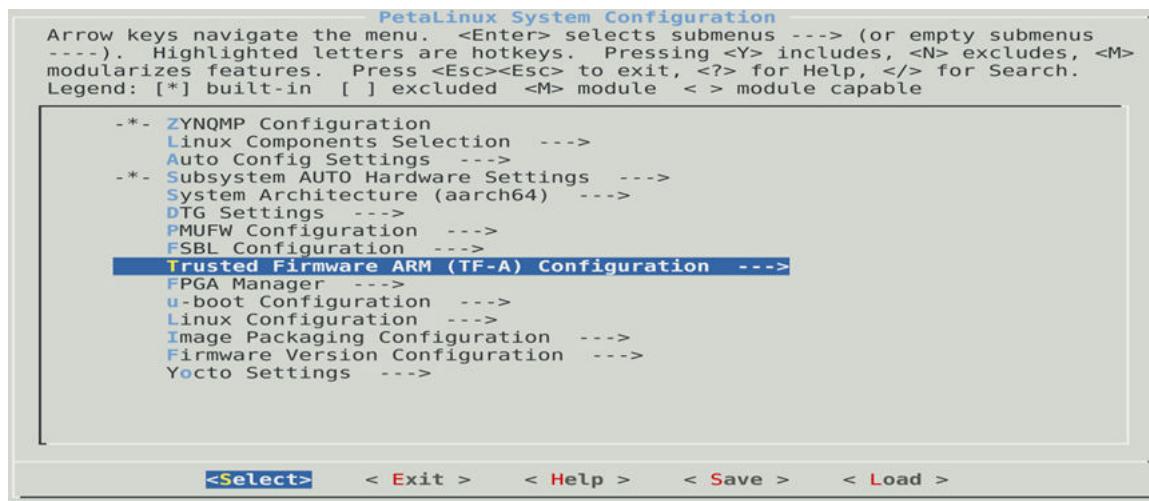
Set any U-Boot environment through Uenv.txt file:

- Add the file into fat partition of sd/emmc/usb where U-Boot sits
- U-Boot reads the file and makes changes from Uenv.txt.

This is possible only in sd/emmc/usb boot modes.

Renamed ARM Trusted Firmware Configuration

From 2023.1, the Arm trusted firmware configuration menu is changed to Trusted Firmware configuration.

Figure 44: Trusted Firmware Configuration

PetaLinux to Create PL Applications and Install on Target

This section is useful for PL application developers who have setup bootable images from the BSP and want to create only PL applications.

1. Source the PetaLinux tool.
2. Create the project.

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

3. Go to the bsp.
4. Use the `petalinux-create` command to add the PL xsa files into the PetaLinux project.

The following command creates the `fpgamanager_dtg` app with the `xsa` file to generate the `dtsi` and `bit` files.

```
petalinux-create -t apps --template fpgamanager_dtg -n can-interface --  
srcuri <path-to-xsa>/system.xsa
```

```
INFO: Create apps: can-interface  
INFO: Copying source urls into the project directory  
INFO: New apps successfully created in <plnx-proj-root>/project-spec/  
meta-user/recipes-apps/can-interface  
INFO: Enabling created component...  
INFO: Sourcing build environmentINFO: Silentconfig rootfs  
INFO: can-interface has been enabled
```

Note: For each XSA, create a separate app using the previous command. FPGA manager should be enabled in `petalinux-config` for both `fpgamanager` and `fpgamanager_dtg` template apps.

5. Run `petalinux-build`.
6. You can find the `dtbo` and `bin` files into `<proj>/build/tmp/deploy/rpm` dir.
7. You can get the required `rpm` files onto the target using `tftp` and install the rpms.

```
rpm -i <rpmname>
```

PetaLinux Project Structure

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

- Device tree
- First stage boot loader (optional)
- U-Boot
- Linux kernel
- The root file system is composed of the following components:
 - Prebuilt packages
 - Linux user applications (optional)
 - User modules (optional)

A PetaLinux project directory contains configuration files of the project, the Linux subsystem, and the components of the subsystem. The `petalinux-build` command builds the project with those configuration files. You can run `petalinux-config` to modify them. The following is an example of a PetaLinux project:

```
project-spec
    hw-description
    configs
    meta-user
pre-built
    linux
        implementation
        images
        xen
hardware
    <project-name>
components
    plnx_workspace
        device-tree
config.project
README
```

Table 38: PetaLinux Project Description

File / Directory in a PetaLinux Project	Description
<code>/.petalinux/</code>	Directory to hold tools usage.
<code>/config.project/</code>	Project configuration file.
<code>/project-spec</code>	Project specification.
<code>/project-spec/hw-description</code>	Hardware description imported from AMD Vivado™ design tools.
<code>/project-spec/configs</code>	Configuration files of top level config and RootFS config.
<code>/project-spec/configs/config</code>	Configuration file used to store user settings.
<code>/project-spec/configs/rootfs_config</code>	Configuration file used for root file system.
<code>/project-spec/configs/busybox</code>	Configuration file for busybox.
<code>/project-spec/configs/init-ifupdown</code>	Configuration file for Ethernet.
<code>/components/plnx_workspace/device-tree/device-tree/</code>	<p>Device tree files used to build device tree. The following files are auto generated by <code>petalinux-config</code>:</p> <ul style="list-style-type: none"> • <code>skeleton.dtsi</code> (Zynq 7000 devices only) • <code>zynq-7000.dtsi</code> (Zynq 7000 devices only) • <code>zynqmp.dtsi</code> (Zynq UltraScale+ MPSoC only) • <code>pcw.dtsi</code> (Zynq 7000 devices and Zynq UltraScale+ MPSoC only) • <code>pl.dtsi</code> • <code>system-conf.dtsi</code> • <code>system-top.dts</code> • <code><bsp name>.dtsi</code> • <code>versal.dtsi</code> for Versal <p>It is not recommended to edit these files, as these files are regenerated by the tools.</p>
<code>/project-spec/meta-user/recipes-bsp/device-tree/files/</code>	<code>system-user.dtsi</code> 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 <code><plnx-proj-root>/project-spec/meta-user/recipes-bsp/device-tree/device-tree.bbappend</code> by adding your DTSI file.
<code>/project-spec/meta-user/recipes-bsp/u-boot/files/platform-top.h</code> (only for MicroBlaze processors)	<code>platform-top.h</code> is copied to <code>include/configs/</code> directory in the U-Boot source
<code>/project-spec/meta-user/conf/petalinuxbsp.conf</code>	This configuration file contains all the local user configurations for your build environment. It is a substitute for <code>local.conf</code> in the Yocto meta layers.

Notes:

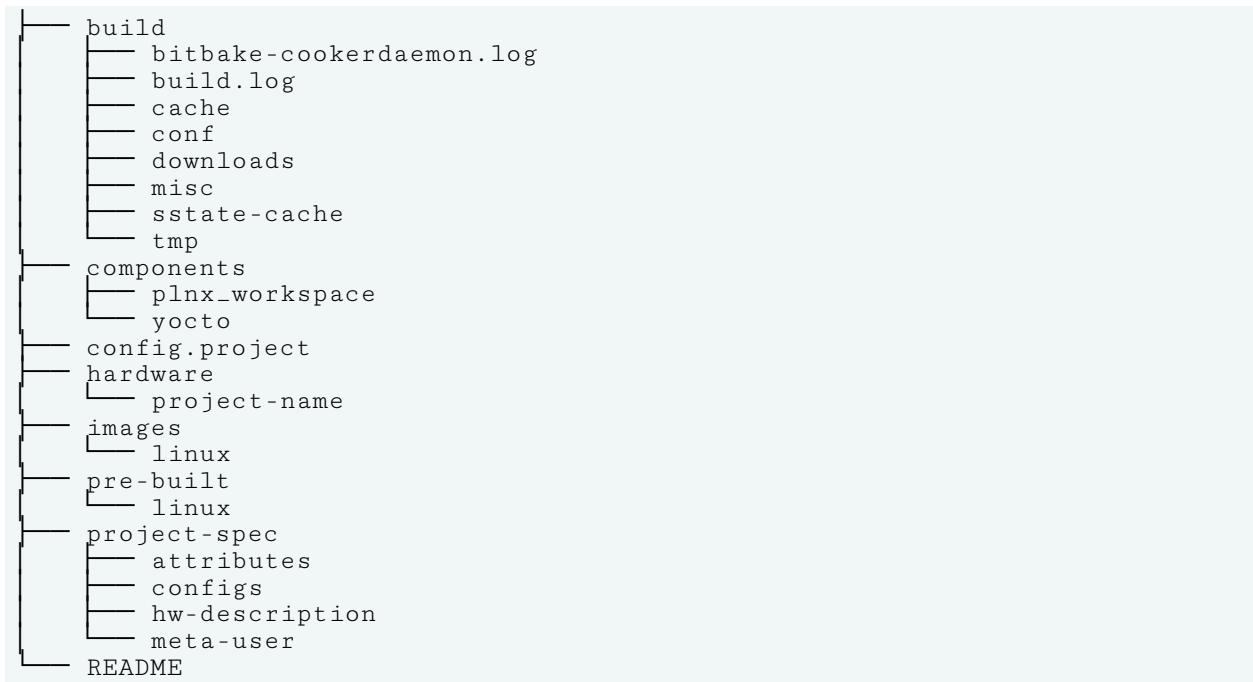
1. All the paths are relative to `<plnx-proj-root>`.

When the project is built, three directories are auto generated:

- `<plnx-proj-root>/build` for the files generated for build.
- `<plnx-proj-root>/images` for the bootable images.
- `<plnx-proj-root>/build/tmp` for the files generated by Yocto. This directory is configurable through `petalinux-config`.

- <plnx-proj-root>/components/yocto has Yocto eSDK. This file is generated when execute petalinux-config/petalinux-build.
- When petalinux-package --prebuilt is run, it creates a pre-built/directory in <PROJECT>/ and copies the build image files from the <PROJECT>/images/linux directory to the <PROJECT>/pre-built/linux/images/ directory.

Here is an example:



Note: <plnx-proj-root>/build/ are automatically generated. Do not manually edit files in this directory. Contents in this directory are updated when you run petalinux-config or petalinux-build. <plnx-proj-root>/images/ are also automatically generated. Files in this directory are updated when you run petalinux-build.

The following table is an example for AMD Zynq™ UltraScale+™ MPSoC.

By default the build artifacts are removed to preserve space after petalinux-build. To preserve the build artifacts, you have to add the INHERIT:remove = "rm_work" in <plnx-proj-root>/project-spec/meta-user/conf/petalinuxbsp.conf, but it increases the project space.

Table 39: 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.

Table 39: Build Directory in a PetaLinux Project (cont'd)

Build Directory in a PetaLinux Project	Description
<code> \${TMPDIR}/work/zynqmp-generic-xilinx-linux/petalinux-image-minimal/1.0-r0/rootfs</code>	RootFS copy of target. This is the staging directory.
<code> \${TMPDIR}/zynqmp-generic-xilinx-linux</code>	Stage directory to hold the libs and header files required to build user apps/libs.
<code> \${TMPDIR}/work/zynqmp-generic-xilinx-linux/linux-xlnx/</code>	Directory to hold files related to the kernel build.
<code> \${TMPDIR}/work/zynqmp-generic-xilinx-linux/u-boot-xlnx</code>	Directory to hold files related to the U-Boot build.
<code><plnx-proj-root>/components/plnx_workspace/device-tree/device-tree"</code>	Directory to hold files related to the device tree build.
<code><plnx-proj-root>/components/yocto</code>	Directory to hold Yocto eSDK content.

Table 40: Image Directory in a PetaLinux Project

Image Directory in a PetaLinux Project	Description
<code><plnx-proj-root>/images/linux/</code>	Directory to hold the bootable images for Linux subsystem
<code><plnx-proj-root>/images/linux</code>	Directory to hold the bootable images for xen hypervisor

Project Layers

The PetaLinux project has the following layer under `<plnx-proj-root>/project-spec`.

meta-user

This layer is a place holder for all user-specific changes. You can add your own bbappend and configuration files in this layer.

Generating Boot Components

Platform Loader and Manager Firmware (PLM)

This is for AMD Versal™ adaptive SoC. This is mandatory. By default, the top-level system settings are set to generate the PLM.

If you had disabled PLM from menuconfig previously, you can configure the project to build PLM as follows:

1. Launch top level system settings configuration menu and configure:

```
petalinux-config
```

- a. Select **Linux Components Selection** ---> submenu.
- b. Select PLM.
- c. Enter your settings.
- d. Exit the menu and save the change.

2. Build the PLM when building the project:

```
petalinux-build
```

3. Build the PLM only:

```
petalinux-build -c plm
```

The PLM ELF file is installed as `plm.elf` for Versal adaptive SoC in `images/linux` inside the project root directory.

For more information on PLM, see *Versal Adaptive SoC System Software Developers Guide* ([UG1304](#)).

Processing System Management Firmware (PSM)

This is for AMD Versal™ adaptive SoC. This is mandatory. By default, the top-level system settings are set to generate the PSM.

If you had disabled PSM from menuconfig previously, you can configure the project to build PSM as follows:

1. Launch top level system settings configuration menu and configure:

```
petalinux-config
```

- a. Select the **Linux Components Selection** submenu.
- b. Select PSM firmware.
- c. Enter your settings.
- d. Exit the menu and save the change.

2. Build the PSM when building the project:

```
petalinux-build
```

3. Build the PSM only:

```
petalinux-build -c psm-firmware
```

The PSM ELF file is installed as `psmfw.elf` for Versal adaptive SoC in `images/Linux` inside the project root directory.

For more information on PSM, see *Versal Adaptive SoC System Software Developers Guide (UG1304)*.

Image Selector

Note: This section is only for AMD Zynq™ UltraScale+™ MPSoCs.

By default, the top-level system settings are not set to generate the Image Selector. If you want enable Image Selector from menuconfig, configure the project to build Image Selector as follows:

1. Launch top level system settings configuration menu and configure:

```
petalinux-config
```

- a. Select the Linux Components Selection submenu.

- b. Select Image Selector.
 - c. Enter your settings.
 - d. Exit the menu and save the change.
2. Build the Image Selector when building the project:

```
petalinux-build
```

3. Build the Image Selector only:

```
petalinux-build -c imgsel
```

The image selector ELF file is installed as `imgsel.elf` for Zynq UltraScale+ MPSoCs in the `images/Linux` inside the project root directory.

First Stage Boot Loader for Zynq UltraScale+ and Zynq 7000 Devices

By default, the top level system settings are set to generate the first stage boot loader. This is optional.

Note: If you do not want the PetaLinux build FSBL/FS-BOOT, you need to manually build it on your own. Else, your system does 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** ---> submenu.
- b. Select **First Stage Boot Loader** option.

```
[ * ] First Stage Bootloader
```
- c. Select the **FSBL Configuration** ---> submenu.
- d. For application compiler flags, select **FSBL Configuration** → **FSBL compiler flags**.
- e. For BSP compiler flags, select **FSBL Configuration** → **FSBL BSP extra compiler flags**.
- f. Enter your compilation flags.
- g. Exit the menu and save the change.

2. Launch `petalinux-build` to build the FSBL:

Build the FSBL when building the project:

```
petalinux-build
```

Build the FSBL only:

```
petalinux-build -c fsbl (for MicroBlaze, it is fs-boot)
```

The boot loader ELF file is installed as `zynqmp_fsbl.elf` for Zynq UltraScale+ MPSoC, `zynq_fsbl.elf` for AMD 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>.

Trusted Firmware-A (TF-A)

This is for Zynq UltraScale+ MPSoC and Versal adaptive SoC. This is mandatory. By default, the top level system settings are set to generate the TF-A.

You can set the TF-A configurable options as follows:

1. Launch top level system settings configuration menu and configure:

```
petalinux-config
```

- a. Select the **Trusted Firmware-A Compilation Configuration** ---> submenu.
- b. Enter your settings.
- c. Exit the menu and save the change.

2. Build the TF-A when building the project:

```
petalinux-build
```

Build the TF-A only:

```
petalinux-build -c arm-trusted-firmware
```

The TF-A ELF file is installed as `b131.elf` for Zynq UltraScale+ MPSoC and Versal adaptive SoC in `images/linux` inside the project root directory.

For more information on TF-A, see <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842107/Arm+Trusted+Firmware>.

PMU Firmware

This is for Zynq UltraScale+ MPSoC only. This is optional. By default, the top level system settings are set to generate the PMU firmware.



CAUTION! If you do not want PetaLinux to build the PMU firmware, you have to manually build it on your own. Else, your system does not boot properly.

You can configure the project to build PMU firmware as follows:

1. Launch top level system settings configuration menu and configure:

```
petalinux-config
```

- a. Select **Linux Components Selection**.
- b. Select **PMU Firmware** option.

```
[ * ] PMU Firmware
```

- c. Select the **PMUFW Configuration** → **PMUFW compiler flags** submenu.
- d. Enter your compilation flags.
- e. Exit the menu and save the change.

2. Build the PMU firmware when building the project:

```
petalinux-build
```

Build the PMU firmware only:

```
petalinux-build -c pmufw
```

The PMU firmware ELF file is installed as `pmufw.elf` for Zynq UltraScale+ MPSoC in `images/linux` inside the project root directory.

For more information on PMU Firmware, see <https://www.wiki.xilinx.com/PMU+Firmware>.

FS-Boot for MicroBlaze Platform Only

FS-Boot in PetaLinux is a first stage boot loader demo for MicroBlaze™ platform only. It is to demonstrate how to load images from flash to the memory and jump to it. If you want to try FS-Boot, you must have a minimum of 8 KB block RAM.

FS-Boot supports parallel flash and SPI flash in standard SPI mode and Quad SPI mode only.

In order for FS-Boot to know where in the flash should get the image, macro CONFIG_FS_BOOT_START needs to be defined. This is done by the PetaLinux tools. PetaLinux tools set this macro automatically from the `boot` partition settings in the menuconfig primary flash partition table settings. For parallel flash, it is the start address of boot partition. For SPI flash, it is the start offset of `boot` partition.

The image in the flash requires a wrapper header followed by a BIN file. FS-Boot gets the target memory location from wrapper. The wrapper needs to contain the following information:

Table 41: Wrapper Information

Offset	Description	Value
0x0	FS-Boot bootable image magic code	x0b8b40008
0x4	BIN image size	User-defined
0x100	FS-Boot bootable image target memory address	User-defined. The PetaLinux tool automatically calculates it from the U-Boot text base address offset from the Memory Settings from the menuconfig.
0x10c	Where the BIN file start	None

The FS-Boot ignores other fields in the wrapper header. The PetaLinux tool generates the wrapper header to wrap around the U-Boot BIN file.

Note: PetaLinux only supports 32-bit MicroBlaze processors.

The FS-Boot supports symmetric multi processing (SMP) from the 2020.1 release onwards. You can have multiple MicroBlaze processors in your design. A maximum of eight cores is supported.

The same FS-Boot which is built as part of the `petalinux-build/petalinux-build -c fsboot` works for all the cores. XSDB is needed to flash the FS-Boot on all the cores. The following is an example for four cores. `xsdb > ta` lists all the available cores. To boot your target with SMP support, follow these steps:

```
<plnx-tool>/tools/xsct/bin/xsdb
xsdb > connect -url <target-url>
xsdb > fpga -f <plnx-proj-root>/images/linux/system.bit
xsdb > ta
xsdb > ta <core number>
xsdb > dow -f <plnx-proj-root>/images/linux/fs-boot.elf
the above two steps for all available cores.
xsdb > dow -f <plnx-proj-root>/images/linux/u-boot.elf
xsdb > dow -f <plnx-proj-root>/images/linux/image.ub
```

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.

- **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.
- **Root Mode:** QEMU creates a subnet on a virtual Ethernet adapter, and relies on a DHCP server on the host system.

The following sections detail how to use the modes, including redirecting the non-root mode so it is accessible from your local host.

Specifying the QEMU Virtual Subnet

By default, PetaLinux uses 192.168.10.* as the QEMU virtual subnet in --root mode. If it has been used by your local network or other virtual subnet, you can use another subnet. You can configure PetaLinux to use other subnet settings for QEMU by running `petalinux-boot` as follows on the command console:

Note: This feature requires sudo access on the local machine, and must be used with the --root option.

```
petalinux-boot --qemu --root --u-boot --subnet <subnet gateway IP>/<number of the bits of the subnet mask>
```

For example, to use subnet 192.168.20.*:

```
petalinux-boot --qemu --root --u-boot --subnet 192.168.20.0/24
```

Appendix F

AMD IP Models Supported by QEMU

Note: By default, QEMU disables any devices for which there is no model available. For this reason it is not possible to use QEMU to test your own customized IP cores (unless you develop C/C++ models for them according to QEMU standard).

For more information on AMD IP models supported by QEMU, see [Quick Emulator User Guide: QEMU](#).

Xen Zynq UltraScale+ MPSoC and Versal Adaptive SoC Example

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

Prerequisites

This section assumes that the following prerequisites are satisfied:

- You have PetaLinux tools software platform ready for building a Linux system customized to your hardware platform. For more information, see [Importing Hardware Configuration](#).
 - You have created a PetaLinux project from the reference BSP.
 - There are DOM0 Xen related prebuilt in the `pre-built/linux/xen` directory, which are `xen.dtb`, `xen-openamp.dtb`, `xen-qemu.dtb`, `xen-Image`, and `xen-rootfs.cpio.gz`.
-

Boot Prebuilt Linux as dom0

1. Copy the prebuilt Xen images to your TFTP directory so that you can load them from U-Boot with TFTP.

```
cd <plnx-proj-root>
cp pre-built/linux/xen/xen.dtb <tftpboot>/
cp pre-built/linux/xen/xen-openamp.dtb <tftpboot>/
cp pre-built/linux/xen/xen-qemu.dtb <tftpboot>/
cp pre-built/linux/xen/xen-Image <tftpboot>/
cp pre-built/linux/xen/xen-rootfs.cpio.gz <tftpboot>/
cp pre-built/linux/xen/xen_boot_tftp.scr <tftpboot>/
cp pre-built/linux/xen/xen_boot_sd.scr <tftpboot>/
cp pre-built/linux/xen/xen <tftpboot>/
```

2. Boot the prebuilt U-Boot image on the board with either JTAG boot or boot from SD card.

Note: For SD card boot, see [Booting PetaLinux Image on Hardware with an SD Card](#) and for JTAG boot, see [Booting PetaLinux Image on Hardware with JTAG](#).

3. Setup TFTP server IP from U-Boot:

```
platform> setenv serverip <TFTP SERVERIP>
```

4. Load Xen images from U-Boot.

- **TFTP Boot:** `xen_boot_tftp.scr`, to be loaded at address `0xC00000` as shown:

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

- **SD Boot:** `xen_boot_sd.scr`, to be loaded at address `0xC00000` as shown:

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

Rebuild Xen

After creating a PetaLinux project for Zynq UltraScale+ MPSoC and AMD Versal™ adaptive SoC, follow the steps to build Xen images:

1. Go to `cd <proj root directory>`.
2. In the `petalinux-config` command, select **Image Packaging Configuration → Root filesystem type (INITRD)**.
3. In `petalinux-config -c rootfs`, select **PetaLinux Package Groups → Packagegroup-petalinux-xen → [*] packagegroup-petalinux-xen**.

Note: If you enable Xen when `/switch_root` is enabled, you see build failures as Xen only supports ramfs boot. ext4-based boot is enable if you enable `switch_root`. To resolve the issue, change the previous config to `petalinux-image-minimal` from `petalinux-initramfs-image`.

4. Enable the following configuration to add the `xen.dtsi` file into `system.dtb` **petalinux-config → DTG Settings → Enable Xen dtsi**
5. Select the `xen rootfs` packages or package groups.
6. Run `petalinux-build`.
7. The build artifacts are in `images/linux` in the project directory.

Note: By default, the `petalinux-build` command does not build Xen. The default root file system does not contain the Xen tools. You have to use Xen RootFS.



IMPORTANT! You are required to update `dom0` memory in `xen-bootargs` in the `xen.dtsi` file based on the image/RootFS size. Also, adjust the previous load addresses based on the image/RootFS size without overlapping.

Boot Built Linux as dom0

1. Copy built Xen images to your TFTP directory so that you can load them from U-Boot with TFTP.

```
cd <plnx-proj-root>
cp images/linux/system.dtb <tftpboot>/
cp images/linux/Image <tftpboot>/
cp images/linux/xen_boot_tftp.scr <tftpboot>/
cp images/linux/xen_boot_sd.scr <tftpboot>/
cp images/linux/xen <tftpboot>/
cp images/linux/rootfs.cpio.gz <tftpboot>/
```

2. Boot built U-Boot image on the board with either JTAG boot or boot from SD card.

Note: For SD card boot, see [Booting PetaLinux Image on Hardware with an SD Card](#) and for JTAG boot, see [Booting PetaLinux Image on Hardware with JTAG](#).

Note: You can also point external built images for dom1 and dom2 to the domU kernels in the configuration, so that Xen boot script is updated with the images that are being pointed. Example to edit the configuration file and build xen boot script as follows:

```
vi images/linux/xen.cfg
export XEN_CONFIG="<Absolute path for xen.cfg>"
export XEN_CONFIG_SKIP="1"
export BB_ENV_PASSTHROUGH_ADDITIONS="$BB_ENV_PASSTHROUGH_ADDITIONS
XEN_CONFIG XEN_CONFIG_SKIP"
petalinux-build -c kernel -x do_deploy
```

Note: Xen boot files are generated in the <plnx-proj-root>/images/linux folder.

3. Setup TFTP server IP from U-Boot:

```
Platform> setenv serverip <TFTP SERVERIP>
```

Note: Platform refers to AMD Versal™ or AMD Zynq™ UltraScale+™ MPSoC.

4. Load Xen images from U-Boot:

- **TFTP Boot:** xen_boot_tftp.scr, to be loaded at address 0xC00000 as shown::

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

- **SD Boot:** xen_boot_sd.scr, to be loaded at address 0xC00000 as shown::

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

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

Booting Prebuilt OpenAMP

Use the following steps to execute OpenAMP:

To boot prebuilt Linux for AMD Versal™ adaptive SoC, follow these steps:

1. Generate BOOT.BIN for Versal adaptive SoC.

```
petalinux-package --boot --plm pre-built/linux/images/plm.elf --psmfw  
pre-built/linux/images/psmfw.elf --dtb pre-built/linux/images/  
openamp.dtb --u-boot -o pre-built/linux/images/BOOT.BIN --force.
```

2. Boot Linux

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

To boot prebuilt Linux for AMD Zynq™ UltraScale+™ MPSoC, follow these steps:

```
cd <plnx-proj-root>  
cp pre-built/linux/images/openamp.dtb pre-built/linux/images/system.dtb  
petalinux-boot --jtag --prebuilt 3 --hw_server-url <hostname:3121>
```

Once Linux is booted, run the following commands for Versal devices only:

1. modprobe virtio_rpmsg_bus
2. modprobe zynqmp_r5_remoteproc

To load OpenAMP firmware and run OpenAMP test application, run the following command:

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

For example, to load image_echo_test, run:

```
echo image_echo_test > /sys/class/remoteproc/remoteproc0/firmware  
echo start > /sys/class/remoteproc/remoteproc0/state  
echo test  
echo stop > /sys/class/remoteproc/remoteproc0/state
```

To stop running, run the following command:

```
echo stop > /sys/class/remoteproc/remoteproc0/state
```

For more examples, see *Libmetal and OpenAMP for Zynq Devices User Guide* ([UG1186](#)).

Partitioning and Formatting an SD Card

For partitioning and formatting an SD card, the following tools are required:

- fdisk
- mkfs

The steps and logs for partitioning are as follows:

- sudo fdisk /dev/sdb

```
Welcome to fdisk (util-linux 2.31.1).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.
```

- Command (m for help): n

Partition type

- p primary (0 primary, 0 extended, 4 free)
- e extended (container for logical partitions)

- Select (default p): p

```
Partition number (1-4, default 1):
First sector (2048-62333951, default 2048):
```

- Last sector, +sectors or +size{K,M,G,T,P} (2048-62333951, default 62333951): 21111220

Creates a new partition 1 of type 'Linux' and of size 10.1 GB. Partition #1 contains a vfat signature.

- Do you want to remove the signature? [Y]es/[N]o: y

The signature is removed by a write command.

- Command (m for help): n

Partition type

- p primary (1 primary, 0 extended, 3 free)
- e extended (container for logical partitions)

- Select (default p): p

```
Partition number (2-4, default 2):  
First sector (21111221-62333951, default 21112832):  
Last sector, +sectors or +size{K,M,G,T,P} (21112832-62333951, default  
62333951):  
Created a new partition 2 of type 'Linux' and of size 19.7 GB.
```

- Command (m for help): w

```
The partition table has been altered.  
Calling ioctl() to re-read partition table.  
Syncing disks.  
Steps and log for formatting:
```

- sudo mkfs.vfat /dev/sdb1

mkfs.fat 4.1 (2017-01-24)

- sudo mkfs.ext4 /dev/sdb2

```
mke2fs 1.44.1 (24-Mar-2018)  
Creating file system with 5152640 4k blocks and 1289280 inodes  
File system UUID: ad549f34-ee6e-4efc-ab03-fba390e98ede  
Superblock backups stored on blocks:  
32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,  
4096000  
Allocating group tables: done  
Writing inode tables: done  
Creating journal (32768 blocks): done  
Writing superblocks and file system accounting information: done
```

- SD EXT ROOTFS BOOT:

```
Mount the fat partition and copy BOOT.BIN, boot.scr, Image, and  
system.dtb files on it.  
Mount the EXT partition and untar rootfs.tar.gz to it.  
Finally unmount the SD card and use it for booting.
```

Auto-mounting an SD Card

Auto-mounting an SD Card during the Build

To auto-mount an SD card during the build, follow the instructions in <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842475/PetaLinux+Yocto+Tips#PetaLinuxYoctoTips-HowtoAutoMountSDcardinYoctoRecipes>.

Auto-mounting SD Partitions after Linux is Running

To mount the SD partitions to a user-defined path, follow these steps:

1. Boot the target to the Linux prompt.
2. Create the direct path, for example:

```
mkdir /media/card
```

3. Edit the file as follows:

```
vim /etc/fstab
```

Once the editor opens, add the following line:

```
/dev/mmcblk0p2 /media/card ext4 defaults 0 1 4
```

Save and exit.

```
reboot
```

4. Check the SD mount point using the `mount` command:

```
root:16:12:48:~# mount
mount
/dev/mmcblk0p2 on / type ext4 (rw,relatime)
devtmpfs on /dev type devtmpfs
(rw,relatime,size=504544k,nr_inodes=126136,mode=755)
proc on /proc type proc (rw,relatime)
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
securityfs on /sys/kernel/security type securityfs
(rw,nosuid,nodev,noexec,relatime)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)
devpts on /dev/pts type devpts (rw,relatime,gid=5,mode=620,ptmxmode=666)
tmpfs on /run type tmpfs
(rw,nosuid,nodev,size=407068k,nr_inodes=819200,mode=755)
tmpfs on /sys/fs/cgroup type tmpfs
(ro,nosuid,nodev,noexec,size=4096k,nr_inodes=1024,mode=755)
cgroup2 on /sys/fs/cgroup/unified type cgroup2
(rw,nosuid,nodev,noexec,relatime,nsdelegate)
```

```
cgroup on /sys/fs/cgroup/systemd type cgroup
(rw,nosuid,nodev,noexec,relatime,xattr,name=systemd)
cgroup on /sys/fs/cgroup/pids type cgroup
(rw,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup
(rw,nosuid,nodev,noexec,relatime,cpu,cpuacct)
cgroup on /sys/fs/cgroup/cpuset type cgroup
(rw,nosuid,nodev,noexec,relatime,cpuset)
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup
(rw,nosuid,nodev,noexec,relatime,net_cls,net_prio)
cgroup on /sys/fs/cgroup/freezer type cgroup
(rw,nosuid,nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/perf_event type cgroup
(rw,nosuid,nodev,noexec,relatime,perf_event)
cgroup on /sys/fs/cgroup/memory type cgroup
(rw,nosuid,nodev,noexec,relatime,memory)
cgroup on /sys/fs/cgroup/blkio type cgroup
(rw,nosuid,nodev,noexec,relatime,blkio)
cgroup on /sys/fs/cgroup/debug type cgroup
(rw,nosuid,nodev,noexec,relatime,debug)
cgroup on /sys/fs/cgroup/devices type cgroup
(rw,nosuid,nodev,noexec,relatime,devices)
hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime,pagesize=2M)
mqueue on /dev/mqueue type mqueue (rw,nosuid,nodev,noexec,relatime)
debugfs on /sys/kernel/debug type debugfs
(rw,nosuid,nodev,noexec,relatime)
tmpfs on /tmp type tmpfs (rw,nosuid,nodev,nr_inodes=1048576)
nfsd on /proc/fs/nfsd type nfsd (rw,relatime)
configfs on /sys/kernel/config type configfs
(rw,nosuid,nodev,noexec,relatime)
ramfs on /run/credentials/systemd-sysusers.service type ramfs
(ro,nosuid,nodev,noexec,relatime,mode=700)
tmpfs on /var/volatile type tmpfs (rw,relatime)
/dev/mmcblk0p1 on /boot type vfat
(rw,relatime,fmask=0022,dmask=0022,codepage=437,iocharset=iso8859-1,short
name=mixed,errors=remount-ro)
/dev/sda on /run/media/sda type vfat
(rw,relatime,gid=6,fmask=0007,dmask=0007,allow_utime=0020,codepage=437,io
charset=iso8859-1,shortname=mixed,errors=remount-ro)
/dev/sdb on /run/media/sdb type vfat
(rw,relatime,gid=6,fmask=0007,dmask=0007,allow_utime=0020,codepage=437,io
charset=iso8859-1,shortname=mixed,errors=remount-ro)
tmpfs on /run/user/1000 type tmpfs
(rw,nosuid,nodev,relatime,size=203532k,nr_inodes=50883,mode=700,uid=1000,
gid=1000)
```

PetaLinux Commands

There are eight independent commands that make up the PetaLinux design flow. They are:

- [petalinux-create](#)
- [petalinux-config](#)
- [petalinux-build](#)
- [petalinux-boot](#)
- [petalinux-package](#)
- [petalinux-util](#)
- [petalinux-upgrade](#)
- [petalinux-devtool](#)

In most cases, the PetaLinux commands are flexible such that the specific options passed to the tools present you with a unique use model, compared to other options for the same tool.

For the purposes of this document, command line arguments that behave as modifiers for workflows are referred to as "options." User-specified values that are accepted by options are shown in italics. In some cases, omitting the user-specified value might result in a built-in default behavior. See the "Default Value" column in the tables for details about relevant default values.

petalinux-create

The `petalinux-create` tool creates objects that are part of a PetaLinux project. This tool provides two separate workflows. In the `petalinux-create -t project` workflow, the tool creates a new PetaLinux project directory structure. In the `petalinux-create -t COMPONENT` workflow, the tool creates a component within the specified project.

These workflows are executed with `petalinux-create -t project` or `petalinux-create -t COMPONENT`, respectively.

petalinux-create Command Line Options

The following table details the command line options that are common to all `petalinux-create` workflows.

Table 42: petalinux-create Command Line Options

Option	Functional Description	Value Range	Default Value
<code>-t, --type TYPE</code>	Specify the TYPE of object to create. This is required.	<ul style="list-style-type: none"> • project • apps • modules 	None
<code>-n, --name NAME</code>	Create object with the specified NAME. This is optional when creating a project from a BSP source. Otherwise, this is required.	User-specified	When creating a project from a BSP source, the project takes the name of the source BSP.
<code>-p, --project PROJECT</code>	PetaLinux project directory path for component creation in a project. This is optional.	User-specified	Current Directory
<code>--force</code>	Overwrite existing files on disk. This is optional.	None	None
<code>-h, --help</code>	Display usage information. This is optional.	None	None
<code>--tmpdir</code>	Specify the local drive path as TMPDIR location when creating the project. Default TMPDIR cannot be under NFS. By default, PetaLinux sets the TMPDIR under /tmp when project is on NFS. You can set your own local drive as TMPDIR PATH using the <code>--tmpdir</code> option	None	None

petalinux-create -t project

The `petalinux-create -t project` command creates a new PetaLinux project at the specified location with a specified name. If the specified location is on the Network File System (NFS), it changes the TMPDIR automatically to `/tmp/<projname_timestamp>`. If `/tmp/<projname_timestamp>` is also on NFS, it throws an error. You can change the TMPDIR through `petalinux-config`. Do not configure the same location as TMPDIR for two different PetaLinux projects as this can cause build errors.

petalinux-create -t project Options

The following table details options used when creating a project. These options are mutually exclusive and one of them must be used when creating a new project.

Table 43: petalinux-create -t project Options

Option	Functional Description	Value Range	Default Value
--template TEMPLATE	Assumes the specified CPU architecture, and is only required when --source is not provided.	<ul style="list-style-type: none"> • microblaze • zynqMP • zynq • versal 	None
-s, --source SOURCE	Creates project based on specified BSP file. SOURCE is the full path on disk to the BSP file. This is optional.	User-specified	None

Note: For AMD boards, the -s, --source BSP flows are suggested. For custom boards, the --template flow is required.

petalinux-create -t project Examples

The following examples demonstrate proper usage of the `petalinux-create -t project` command.

- Create a new project from a reference BSP file:

```
petalinux-create -t project -s <PATH-TO-BSP>
```

- Create a new project based on the MicroBlaze™ processor template:

```
petalinux-create -t project -n <NAME> --template microblaze
```

- Create a project from the PetaLinux project BSP and specify the TMPDIR PATH:

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

- Create a project from a template and specify the TMPDIR PATH:

```
petalinux-create -t project -n <PROJECT> --template <TEMPLATE> --tmpdir <TMPDIR PATH>
```

By default, the directory structure created by --template is minimal, and is not useful for building a complete system until initialized using the `petalinux-config --get-hw-description` command. Projects created using a BSP file as their source are suitable for building immediately.

petalinux-create -t COMPONENT

The `petalinux-create -t COMPONENT` command allows you to create various components within the specified PetaLinux project. These components can be selectively included or excluded from the final system by toggling them using the `petalinux-config -c rootfs` workflow.

petalinux-create -t COMPONENT Options

The `petalinux-create -t apps` command allows you to customize how application components are created. The following table details options that are common when creating applications within a PetaLinux project

Table 44: petalinux-create -t apps Options

Option	Functional Description	Value Range	Default Value
<code>-s, --source SOURCE</code>	Create the component from pre-existing content on disk. Valid formats are <code>.tar.gz</code> , <code>.tar.bz2</code> , <code>.tar</code> , <code>.zip</code> , and source directory (uncompressed). This is optional.	User-specified	None
<code>--template TEMPLATE</code>	Create the component using a pre-defined application template. This is optional.	<ul style="list-style-type: none"> • c • c++ • autoconf, for GNU autoconfig • fpgamanager: Pack the <code>.dtbo</code>, <code>bit.bin</code>, <code>shell.json</code> and <code>.xclbin</code> files into rootfs, support only for Zynq, Zynq UltraScale+ MPSoC. <code>fpgamanger_dtg</code> : Extract the <code>.xsa</code> and pack the <code>pl.dtbo</code>, <code>.bin/pdi</code>, <code>shell.json</code> and <code>.xclbin</code> files into rootfs, support only for Zynq, Zynq UltraScale+ MPSoC and Versal <code>fpgamanager_dtg_dfx</code>: Extract the <code>rprm.xsa</code> and pack the <code>rprm.dtbo</code>, <code>.pdi</code>, <code>shell.json</code> and <code>.xclbin</code> files into rootfs, support only for Versal DFX flow users. • install, for applications which have prebuilt binary only 	c
<code>--enable</code>	Upon creating the component, enable it in the project's root file system. You can also enable using the <code>petalinux-config -c rootfs</code> . This is optional.	None	Disabled
<code>--srcuri</code>	Creates an application with local sources or from remote source.	None	None

petalinux-create -t COMPONENT Examples

The following examples demonstrate proper usage of the `petalinux-create -t COMPONENT` command.

- Create an application component that is enabled in the root file system.

```
petalinux-create -t apps -n <NAME> --template <template> --enable
```

- Create a new install-only application component. In this flow, nothing is compiled.

```
petalinux-create -t apps -n <NAME> --template install
```

- Create a new kernel module and enable it.

```
petalinux-create -t modules -n <name> --template <template> --enable
```

- Create an application with multiple source files.

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

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

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

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

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

Note: This is applicable for applications and modules.

petalinux-config

The `petalinux-config` tool allows you to customize the specified project. This tool provides two separate workflows. In the `petalinux-config --get-hw-description` workflow, a project is initialized or updated to reflect the specified hardware configuration. In the `petalinux-config -c COMPONENT` workflow, the specified component is customized using a menuconfig interface.

petalinux-config Command Line Options

The following table details the available options for the `petalinux-config` tool.

Table 45: petalinux-config Command Line Options

Option	Functional Description	Value Range	Default Value
<code>-p, --project <path to project directory></code>	Specifies path to the project to be configured.	User-specified	Current Directory
<code>--get-hw-description <DIR containing XSA>/--get-hw-description=<DIR containing XSA>/--get-hw-description=<PATH-TO-XSA></code>	Initializes or updates the hardware configuration for the PetaLinux project. Mutually exclusive with <code>-c</code> . This is required.	User-specified	Current Directory

Table 45: petalinux-config Command Line Options (cont'd)

Option	Functional Description	Value Range	Default Value
<code>-c, --component COMPONENT</code>	Configures the specified system component. Mutually exclusive with <code>--get-hw-description</code> . This is required.	<ul style="list-style-type: none"> kernel rootfs u-boot bootloader (for AMD Zynq™ UltraScale+™ MPSoC, Zynq architecture, and MicroBlaze™ CPU) pmufw, for Zynq UltraScale+ MPSoC only device-tree plm, for AMD Versal™ adaptive SoC psmfw, for Versal adaptive SoC 	None
<code>--silentconfig</code>	Allows you to restore a prior configuration. Example: Execute the following command after enabling or disabling different configs by editing <proj-root>/project-spec/configs/config <code>\$ petalinux-config --silentconfig</code>	None	None
<code>-v, --verbose</code>	Displays additional output messages. This is optional.	None	None
<code>-h, --help</code>	Displays tool usage information. This is optional.	None	None

petalinux-config --get-hw-description

The `petalinux-config --get-hw-description` command allows you to initialize or update a PetaLinux project with hardware-specific information from the specified AMD Vivado™ Design Suite hardware project. The components affected by this process can include FSBL configuration, U-Boot options, Linux kernel options, and the Linux device tree configuration. This workflow should be used carefully to prevent accidental and/or unintended changes to the hardware configuration for the PetaLinux project. The path used with this workflow is the directory that contains the XSA file rather than the full path to the XSA file itself. This entire option can be omitted if run from the directory that contains the XSA file.

petalinux-config --get-hw-description Examples

The following examples demonstrate proper usage of the `petalinux-config --get-hw-description` command.

- Initialize a PetaLinux project within the project directory with an external XSA.

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

- Initialize a PetaLinux project from within the directory containing an XSA.

```
petalinux-config --get-hw-description -p <PATH-TO-PETALINUX-PROJECT>
```

- Initialize a PetaLinux project from a neutral location.

```
petalinux-config --get-hw-description <PATH-TO-XSA DIRECTORY>/--get-hw-  
description=<PATH-TO-XSA DIRECTORY>/--get-hw-description=<PATH-TO-XSA> -p  
<PATH-TO-PETALINUX-PROJECT>
```

petalinux-config -c COMPONENT

The `petalinux-config -c COMPONENT` command allows you to use a standard menuconfig interface to control how the embedded Linux system is built, and also generates the source code for embedded software applications. When `petalinux-config` is executed with no other options, it launches the system-level or "generic" menuconfig. This interface allows you to specify the information, such as the desired boot device, or metadata about the system, such as the default hostname. The `petalinux-config -c kernel`, `petalinux-config -c u-boot`, and `petalinux-config -c rootfs` workflows launch the menuconfig interfaces for customizing the Linux kernel, U-Boot, and the root file system, respectively.

The `--silentconfig` option allows you to restore a prior configuration.

Example:

Execute the following command after enabling or disabling different configs by editing `<proj-root>/project-spec/configs/rootfs_config`

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

Use this command to use the existing configurations without editing. In this case, the menuconfig is not launched.

petalinux-config -c COMPONENT Examples

The following examples demonstrate proper usage of the `petalinux-config -c COMPONENT` command:

- Start the menuconfig for the system-level configuration.

```
petalinux-config
```

- Enable different rootfs packages without opening the menuconfig. Execute the following command after enabling or disabling different packages by editing <proj-root>/project-spec/configs/rootfs_config

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

The following warning message appears when `petalinux-config` or `petalinux-build` for components (for example: `petalinux-build -c u-boot`) is run. This message can be ignored.



WARNING! SRC_URI is conditionally overridden in this recipe. Thus several devtool-override-* branches are created, one for each override that makes changes to SRC_URI. It is recommended that you make changes to the devtool branch first, checkout, and rebase each devtool-override-* branch and update any unique patches there (duplicates on those branches are ignored by devtool finish/update-recipe).

petalinux-build

The `petalinux-build` tool builds either the entire embedded Linux system or a specified component of the Linux system. This tool uses the Yocto Project underneath. Whenever `petalinux-build` is invoked, it internally calls `bitbake`. While the tool provides a single workflow, the specifics of its operation can be dictated using the `petalinux-build -c` and `petalinux-build -x` options.

petalinux-build Command Line Options

The following table outlines the valid options for the `petalinux-build` tool.

Table 46: petalinux-build Command Line Options

Option	Functional Description	Value Range	Default Value
<code>-p, --project PROJECT</code>	PetaLinux project directory path. This is optional.	User-specified	None

Table 46: petalinux-build Command Line Options (cont'd)

Option	Functional Description	Value Range	Default Value
<code>-c, --component COMPONENT</code>	Builds specified component. These are the default values which are supported. You can build against your own target (such as your application or module). This is optional.	<ul style="list-style-type: none"> bootloader (AMD Zynq™ UltraScale+™ MPSoC, Zynq architecture, and MicroBlaze™ CPU) kernel u-boot rootfs pmufw, only for Zynq UltraScale+ MPSoC arm-trusted-firmware, for Zynq UltraScale+ MPSoC and AMD Versal™ adaptive SoC. device-tree plm, only for Versal adaptive SoC psmfw, only for Versal adaptive SoC apps modules 	None
<code>-x, --execute STEP</code>	Executes specified build step. All Yocto tasks can be passed through this option. To get all tasks of a component, use "listtasks". This is optional.	<ul style="list-style-type: none"> build clean cleanall cleansstate distclean install listtasks populate_sysroot package mrproper 	Build
<code>-v, --verbose</code>	Displays additional output messages. This is optional.	None	None
<code>-s, --sdk</code>	Builds Yocto SDK. This is optional.	None	None
<code>--esdk</code>	Builds Yocto e-SDK. This is optional.	None	Nnone
<code>-h</code>	Lists all the sub-components of a component. Valid only for rootfs. This is optional.	rootfs	None
<code>-f, --force</code>	Forces a specific task to run against a component, or a single task in the component, ignoring the stamps. This is optional.	None	None

petalinux-build --component

The `petalinux-build -c` option builds the specified component of the embedded system. When no components are specified, the `petalinux-build` tool operates on the project as a whole. User-created components for the root file system can be built by targeting those components by name (for example, with `-c <APP-NAME>`). This is equivalent to `bitbake <COMPONENT>`. Each recipe can be specified as a component for `petalinux-build -c <component>`. The component can be a user created app or package/package group in the root filesystem.

The `petalinux-build` command with no arguments runs `bitbake petalinux-user-image` internally. The default image target is `petalinux-image-minimal`. There is no restriction on the components, and you can build your own packages. For the names of the packages, search in `petalinux-config -c rootfs`.

An example to build base-files is as follows:

```
petalinux-build -c base-files
```

petalinux-build -c component options

The following table summarizes the available components that can be targeted with this command:

Table 47: petalinux-build -c Components

Component	Equivalent Bitbake Commands	Description
bootloader	<code>bitbake virtual/fsbl</code> <code>bitbake virtual/fsboot</code> (for MicroBlaze™ processor)	Build only the boot loader elf image and copy it into <code><plnx-proj-root>/images/linux/</code> . For AMD Zynq™ UltraScale+™ MPSoC and Zynq 7000 devices, it is FSBL and for MicroBlaze™ processor, it is fs-boot.
device tree	<code>bitbake virtual/dtb</code>	Build only the device tree DTB file and copy it into <code><plnx-proj-root>/images/linux/</code> . The device tree source is in <code><plnx-proj-root>/components/plnx_workspace/device-tree/device-tree/</code> .
arm-trusted-firmware	<code>bitbake virtual/arm-trusted-firmware</code>	Build only the TF-A image and copy it into <code><plnx-proj-root>/images/linux</code> .
pmufw	<code>bitbake virtual/pmufw</code>	Build only the PMU firmware image and copy it into <code><plnx-proj-root>/images/linux</code> .
kernel	<code>bitbake virtual/kernel</code>	Build only the Linux kernel image and copy it into <code><plnx-proj-root>/images/linux</code> .
rootfs	<code>bitbake petalinux-user-image -c do_image_complete</code>	Build only the root file system. It generates the target rootfs in <code>\$ {TMPDIR}/work/\${MACHINE}/petalinux-user-image/1.0-r0/rootfs/</code> and the sysroot in <code>\$ {TMPDIR}/tmp/sysroots/\${MACHINE}</code> .
u-boot	<code>bitbake virtual/bootloader</code>	Build only the U-Boot elf image and copy it into <code><plnx-proj-root>/images/linux</code> .

Table 47: petalinux-build -c Components (cont'd)

Component	Equivalent Bitbake Commands	Description
plm	virtual/plm	Build only the PLM image and copy it into <plnx-proj-root>/images/linux.
psmfw	virtual/psm-firmware	Build only the PSM firmware image and copy it into <plnx-proj-root>/image/linux.
image selector	imgsel	Build only the Image Selector firmware image and copy it into <plnx-proj-root>/image/linux.

petalinux-build --execute

The `petalinux-build -x` option allows you to specify a build step to the `petalinux-build` tool to control how the specified components are manipulated. The Yocto task name has a `do_` prefix to the `petalinux-build` step. All Yocto tasks can be passed through this option. To get all tasks of a component, use `listtasks`.

Commands for petalinux-build -x

The following table summarizes some of the available commands that can be used with this option:

Table 48: petalinux-build -x options

Component	Description
clean	Cleans build data for the target component.
cleansstate/ distclean	Removes the shared state cache of the corresponding component.
cleanall	Removes the downloads and shared state cache. Cleans the work directory of a component.
mrproper	Cleans the build area. This removes the <plnx-proj-root>/build/, <TMPDIR>, and <plnx-proj-root>/images/ directories. This is the recommended way of cleaning the entire project.
build	Builds the target component.
install	Installs the target component. For bootloader, TF-A, Linux kernel, U-Boot, and device tree, it copies the generated binary into <plnx-proj-root>/images/linux/. For the root file system and root file system component, it copies the generated binary to target the root file system host copy \${TMPDIR}/work/\${MACHINE}/petalinux-user-image/1.0-r0/rootfs/.
package	Generates FIT image <code>image.ub</code> from build area and copies it into <plnx-proj-root>/images/linux/. Valid for <code>-c</code> all or when no component is specified only.
listtasks	Gets all tasks of a specific component.

petalinux-build Examples

The following examples demonstrate proper usage of the `petalinux-build` command.

- Clear the build area of the PetaLinux project for archiving as a BSP or for revision control. This example retains the images directory of the project.

```
petalinux-build -x distclean
```

- Clean all build collateral from the U-Boot component of the PetaLinux project.

```
petalinux-build -c u-boot -x cleansstate
```

- Clean all build collateral. It removes build/, \${TMPDIR} and images. This brings the project to its initial state.

```
petalinux-build -x mrproper
```

- Create an updated FIT image from the current contents of the deploy area.

```
petalinux-build -x package
```

- Build the entire PetaLinux project.

```
petalinux-build
```

- Build the kernel forcefully by ignoring the stamps (output of tasks from last successful build).

```
petalinux-build -c kernel -f
```

- Compile kernel forcefully by ignoring do_compile task stamp.

```
petalinux-build -c kernel -x compile -f
```

- Build the eSDK and copy it to <proj_root>/images/linux/esdk.sh

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

- Pack all the components of petalinux-build.

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

- Pack only the sysroot components.

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

Note: You can find the archiver tar in <pplx-proj-root>/images/linux.

petalinux-boot

The `petalinux-boot` command boots MicroBlaze™ CPU, AMD Zynq™ devices, AMD Versal™ adaptive SoC, and AMD Zynq™ UltraScale+™ MPSoC with PetaLinux images through JTAG/QEMU. This tool provides two distinct workflows:

- In `petalinux-boot --jtag` workflow, images are downloaded and booted on a physical board using a JTAG cable connection.

- In `petalinux-boot --qemu` workflow, images are loaded and booted using the QEMU software emulator.

Either the `--jtag` or the `--qemu` is mandatory for the `petalinux-boot` tool. By default, the `petalinux-boot` tool loads binaries from the `<plnx-proj-root>/images/linux/` directory.

petalinux-boot Command Line Options

The following table details the command line options that are common to all `petalinux-boot` workflows.

Table 49: petalinux-boot Command Line Options

Option	Functional Description	Value Range	Default Value
<code>--jtag</code>	Use the JTAG workflow. Mutually exclusive with the QEMU workflow. One of the two, <code>--jtag</code> or <code>--qemu</code> is required.	None	None
<code>--qemu</code>	Use the QEMU workflow. Mutually exclusive with the JTAG workflow. One of the two, <code>--jtag</code> or <code>--qemu</code> is required.	None	None
<code>--prebuilt</code>	Boot a prebuilt image. This is optional.	<ul style="list-style-type: none"> • 1 (bitstream /FSBL) • 2 (U-Boot) • 3 (Linux kernel) <p>Note: 1 is not a valid option for the QEMU workflow.</p>	None
<code>--boot-addr BOOT_ADDR</code>	Boot address. This is optional.	None	None
<code>--u-boot</code>	This option can be used to download specified U-Boot binary along with dependent files to boot into the U-Boot. It selects an U-Boot ELF image from <code><plnx-proj-root>/images/linux/u-boot.elf</code> . This is optional.	User-specified	<code><plnx-proj-root>/images/linux/u-boot.elf</code>
<code>--kernel</code>	This option can be used to download specified kernel binary along with dependent files to boot kernel. This option picks the kernel image from <code><plnx-proj-root>/images/linux/</code> . This is optional.	User-specified	<ul style="list-style-type: none"> • zImage for AMD Zynq™ 7000 devices • Image for AMD Zynq™ UltraScale+™ MPSoC, and AMD Versal™ adaptive SoC • image.elf for MicroBlaze™ CPU <p>The default image is in <code><plnx-proj-root>/images/linux</code>.</p>
<code>-v, --verbose</code>	Displays additional output messages. This is optional.	None	None
<code>-h, --help</code>	Displays tool usage information. This is optional.	None	None

petalinux-boot --jtag

The `petalinux-boot --jtag` command boots the MicroBlaze™ CPUs, the AMD Zynq™ UltraScale+™ MPSoCs, Zynq 7000 devices, or AMD Versal™ adaptive SoCs with a PetaLinux image using a JTAG connection.

Note: The `petalinux-boot --jtag` command might not work as expected when executed within a virtual machine because virtual machines often have problems with JTAG cable drivers.

petalinux-boot --jtag Options

The following table contains details of options specific to the JTAG boot workflow.

Table 50: petalinux-boot --jtag Options

Option	Functional Description	Value Range	Default Value
<code>--xsdb-conn COMMAND</code>	Customized XSDB connection command to run prior to boot. This is optional.	User-specified	None
<code>--hw_server-url URL</code>	URL of the hw_server to connect to. This is optional.	User-specified	None
<code>--tcl OUTPUTFILE</code>	Log JTAG Tcl commands used for boot. This is optional.	User-specified	None
<code>--fpga</code>	Program FPGA bitstream. This is optional.	None	If no bitstream is specified with the <code>--bitstream</code> option, it uses the bitstream from one of the following locations: <ul style="list-style-type: none"> • If you are using build images to boot, it picks the bitstream from <code><plnx-proj-root>/images/linux/system.bit</code> • If you are using prebuilt images to boot, it picks the bitstream pick <code><plnx-proj-root>/prebuilt/linux/implementation/download.bit</code>
<code>--bitstream BITSTREAM</code>	Specify a bitstream. This is optional.	User-specified	None
<code>--pmufw PMUFW-ELF</code>	PMU firmware image. This is optional and applicable for AMD Zynq™ UltraScale+™ MPSoC. PMU firmware image is loaded by default, unless it is specified otherwise. To skip loading PMU firmware, use <code>--pmufw no</code> .	None	<code><plnx-proj-root>/images/linux/pmufw.elf</code>
<code>before-connect <CMD></code>	Extra command to run before XSDB connect command. Ensure the command is properly quoted in your shell. This is optional and can be used multiple times.	None	None

Table 50: **petalinux-boot --jtag Options (cont'd)**

Option	Functional Description	Value Range	Default Value
after-connect <CMD>	Extra commands to run after XSDB connect command. Ensure the command is properly quoted in your shell. This is optional and can be used multiple times.	None	None
--rootfs	Specify the cpio rootfile system needs to be used for jtag boot. Supports for: Zynq, Zynq UltraScale+ MPSoC, Versal, and MicroBlaze.	User-specified	<plnx-proj-root>/images/linux/rootfs.cpio.gz.u-boot

petalinux-boot --jtag Examples

Select the images for loading on target from the following:

1. Prebuilt directory: <plnx-proj-root>/pre-built/linux/images. These are prebuilt images packed along with the BSP.
2. Images directory: <plnx-proj-root>/images/linux. These are the images built by the user.

The following examples demonstrate some use-cases of the `petalinux-boot --jtag` command.

- Download bitstream and FSBL for Zynq 7000 devices, and FSBL and PMU firmware for Zynq UltraScale+ MPSoC

```
petalinux-boot --jtag --prebuilt 1
```

Note: Images are taken from <plnx-proj-root>/pre-built/linux/images directory.

- Boot U-Boot on target board after loading bitstream/boot loader.

```
petalinux-boot --jtag --prebuilt 2
```

Note: Images are taken from <plnx-proj-root>/pre-built/linux/images directory.

```
petalinux-boot --jtag --u-boot --fpga
```

Note: Images are taken from <plnx-proj-root>/images/linux directory.

- For MicroBlaze™ processors, the previous commands download the bitstream to the target board, and boot the U-Boot on the target board.
- For Zynq 7000 devices, they download the bitstream and FSBL to the target board, and boot the U-Boot on the target board.
- For Zynq UltraScale+ MPSoC, they download the bitstream, PMU firmware, FSBL, and U-Boot, and boot upto U-Boot on the target board.
- For AMD Versal™ adaptive SoC, they download `BOOT.BIN` (which contains the PDI, PLM firmware, PSM firmware, U-Boot, and DTB) and boot the U-Boot on the target board.

- Boot prebuilt kernel on target board after loading bitstream, boot loader, and U-Boot.

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

Note: Images are taken from <plnx-proj-root>/pre-built/linux/images directory.

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

Note: Images are taken from <plnx-proj-root>/images/linux directory.

- Generate debug messages while loading images:

- ```
petalinux-boot --jtag --u-boot/--kernel -v
```

**Note:** Images are taken from <plnx-proj-root>/images/linux directory.

- For JTAG boot, specify the cpio rootfile system. It supports Zynq , Zynq UltraScale+ MPSoC, Versal, and MicroBlaze.
- Boot customized rootfs image with kernel using JTAG:

```
petalinux-boot --jtag --kernel --rootfs images/linux/rootfs.cpio.gz.u-
boot --hw-server-url xhdbfarmrki7:3121
[INFO] Sourcing buildtools
INFO: Use bitstream: "xilinx-zcu102-2023.2/images/linux/system.bit".
INFO: Please use --fpga --bitstream <BITSTREAM> to specify a bitstream if
you want to use other bitstream.
INFO: Launching XSDB for file download and boot.
INFO: This may take a few minutes, depending on the size of your image.
rlwrap: warning: your $TERM is 'xterm-256color' but rlwrap couldn't find
it in the terminfo database. Expect some problems.: Inappropriate ioctl
for device
INFO: Configuring the
FPGA...
INFO: Downloading bitstream: xilinx-zcu102-2023.2/images/linux/system.bit
to the target.
```

```
INFO: Downloading ELF file: /wrk/everest_fcv_nobkup/kavyasre/_20230819_/
xilinx-zcu102-2023.2/images/linux/pmufw.elf to the target.
INFO: Downloading ELF file: xilinx-zcu102-2023.2/images/linux/
zynqmp_fsbl.elf to the target.
INFO: Downloading ELF file: xilinx-zcu102-2023.2/images/linux/u-boot.elf
to the target.
INFO: Loading image: xilinx-zcu102-2023.2/images/linux/Image at
0x00200000 xilinx-zcu102-2023.2/images/linux/system.dtb at 0x00100000
INFO: Loading image: xilinx-zcu102-2023.2/images/linux/rootfs.cpio.gz.u-
boot at 0x04000000
INFO: Loading image: xilinx-zcu102-2023.2/images/linux/boot.scr at
0x20000000
INFO: Downloading ELF file: xilinx-zcu102-2023.2/images/linux/bl31.elf to
the target.

[xhd-zcu102-6] Systest# connect com0
Connecting to device com0. Use Ctrl-\ to escape.
[xhd-zcu102-6] Systest# power 0 power 1
Power: OFF
Removing Uart devices...
Power: ON
```

```
[xhd-zcu102-6] Systest# connect com0
Connecting to device com0. Use Ctrl-\ to escape.
Zynq MP First Stage Boot Loader
Release 2023.2 Aug 11 2023 - 10:10:30
NOTICE: BL31: Secure code at 0x60000000
NOTICE: BL31: Non secure code at 0x80000000
NOTICE: BL31: v2.8(release):xlnx_rebase_v2.8_2023.2_ksb-14-gf0ba7ad93
NOTICE: BL31: Built : 04:12:26, Jul 5 2023

U-Boot 2023.01 (Aug 11 2023 - 04:59:53 +0000)

CPU: ZynqMP
Silicon: v3
Chip: zu9eg
Model: ZynqMP ZCU102 Rev1.0
Board: Xilinx ZynqMP
DRAM: 2 GiB (effective 4 GiB)
PMUFW: v1.1
PMUFW: No permission to change config object
EL Level: EL2
Secure Boot: not authenticated, not encrypted
Core: 75 devices, 31 uclasses, devicetree: board
NAND: 0 MiB
MMC: mmc@ff170000: 0
Loading Environment from nowhere... OK
In: serial
Out: serial
Err: serial
Bootmode: JTAG_MODE
Reset reason: SRST
ofnode_read_bootscript_address: Missing /u-boot node
ofnode_read_bootscript_flash: Missing /u-boot node
Net:
ZYNN GEM: ff0e0000, mdio bus ff0e0000, phyaddr 12, interface rgmii-id

Warning: ethernet@ff0e0000 (eth0) using random MAC address -
ba:af:19:9b:ba:af
eth0: ethernet@ff0e0000
scanning bus for devices...
SATA link 0 timeout.
Target spinup took 0 ms.
AHCI 0001.0301 32 slots 2 ports 6 Gbps 0x3 impl SATA mode
flags: 64bit ncq pm clo only pmp fbss pio slum part ccc apst
Device 0: (1:0) Vendor: ATA Prod.: TS240GSSD220S Rev: S103
 Type: Hard Disk
 Capacity: 228936.5 MB = 223.5 GB (468862128 x 512)
starting USB...
Bus usb@fe200000: Register 2000440 NbrPorts 2
Starting the controller
USB XHCI 1.00
scanning bus usb@fe200000 for devices... 2 USB Device(s) found
 scanning usb for storage devices... 1 Storage Device(s) found
Hit any key to stop autoboot: 0
JTAG: Trying to boot script at 20000000
Executing script at 20000000
Trying to load boot images from jtag
Loading init Ramdisk from Legacy Image at 04000000 ...
 Image Name: petalinux-image-minimal-xilinx-z
 Created: 2011-04-05 23:00:00 UTC
 Image Type: AArch64 Linux RAMDisk Image (uncompressed)
 Data Size: 31711975 Bytes = 30.2 MiB
 Load Address: 00000000
```

```
Entry Point: 00000000
Verifying Checksum ... OK
Flattened Device Tree blob at 00100000
Booting using the fdt blob at 0x100000
Working FDT set to 100000
 Loading Ramdisk to 79db3000, end 7bbf12e7 ... OK
 Loading Device Tree to 0000000079da2000, end 0000000079db2f76 ... OK
Working FDT set to 79da2000

Starting kernel ...

[0.000000] Booting Linux on physical CPU 0x000000000000 [0x410fd034]
[0.000000] Linux version 6.1.30-xilinx-v2023.2 (oe-user@oe-host)
(aarch64-xilinx-linux-gcc (GCC) 12.2.0, GNU ld (GNU Binutils)
2.39.0.20220819) #1 SMP Fri Aug 4 10:14:47 UTC 2023
[0.000000] Machine model: ZynqMP ZCU102 Rev1.0
[0.000000] earlycon: cdns0 at MMIO 0x00000000ff000000 (options
'115200n8')
[0.000000] printk: bootconsole [cdns0] enabled
[0.000000] efi: UEFI not found.
[0.000000] Zone ranges:
[0.000000] DMA32 [mem 0x0000000000000000-0x00000000ffffffffff]
[0.000000] Normal [mem 0x0000000100000000-0x000000087ffffffff]
[0.000000] Movable zone start for each node
[0.000000] Early memory node ranges
[0.000000] node 0: [mem 0x0000000000000000-0x000000007fefffff]
[0.000000] node 0: [mem 0x0000000800000000-0x000000087ffffffff]
[0.000000] Initmem setup node 0 [mem
0x0000000000000000-0x000000087ffffffff]
[0.000000] On node 0, zone Normal: 256 pages in unavailable ranges
[0.000000] cma: Reserved 256 MiB at 0x0000000069c00000
[0.000000] psci: probing for conduit method from DT.
[0.000000] psci: PSCIv1.1 detected in firmware.
[0.000000] psci: Using standard PSCI v0.2 function IDs
[0.000000] psci: MIGRATE_INFO_TYPE not supported.
[0.000000] psci: SMC Calling Convention v1.2
[0.000000] percpu: Embedded 18 pages/cpu s35816 r8192 d29720 u73728
[0.000000] Detected VIPT I-cache on CPU0
[0.000000] CPU features: detected: ARM erratum 845719
[0.000000] alternatives: applying boot alternatives
[0.000000] Built 1 zonelists, mobility grouping on. Total pages:
1031940
[0.000000] Kernel command line: earlycon console=ttyPS0,115200
clk_ignore_unused root=/dev/ram0 rw init_fatal_sh=1
[0.000000] Unknown kernel command line parameters "init_fatal_sh=1",
will be passed to user space.
[0.000000] Dentry cache hash table entries: 524288 (order: 10,
4194304 bytes, linear)
[0.000000] Inode-cache hash table entries: 262144 (order: 9, 2097152
bytes, linear)
[0.000000] mem auto-init: stack:all(zero), heap alloc:off, heap
free:off
[0.000000] software IO TLB: area num 4.
[0.000000] software IO TLB: mapped [mem
0x000000007bf00000-0x000000007ff00000] (64MB)
[0.000000] Memory: 3730672K/4193280K available (14656K kernel code,
1028K rwdata, 4184K rodata, 2240K init, 377K bss, 200464K reserved,
262144K cma-reserved)
[0.000000] rcu: Hierarchical RCU implementation.
[0.000000] rcu: RCU event tracing is enabled.
[0.000000] rcu: RCU restricting CPUs from NR_CPUS=16 to
nr_cpu_ids=4.
[0.000000] rcu: RCU calculated value of scheduler-enlistment delay is
```

```
25 jiffies.
[0.000000] rcu: Adjusting geometry for rcu_fanout_leaf=16,
nr_cpu_ids=4
[0.000000] NR_IRQS: 64, nr_irqs: 64, preallocated irqs: 0
[0.000000] GIC: Adjusting CPU interface base to 0x00000000f902f000
[0.000000] Root IRQ handler: gic_handle_irq
[0.000000] GIC: Using split EOI/Deactivate mode
[0.000000] rcu: srcu_init: Setting srcu_struct sizes based on
contention.
[0.000000] arch_timer: cp15 timer(s) running at 99.99MHz (phys).
[0.000000] clocksource: arch_sys_counter: mask: 0xfffffffffffffff
max_cycles: 0x170f8de2d3, max_idle_ns: 440795206112 ns
[0.000000] sched_clock: 57 bits at 100MHz, resolution 10ns, wraps
every 4398046511101ns
[0.008465] Console: colour dummy device 80x25
[0.012558] Calibrating delay loop (skipped), value calculated using
timer frequency.. 199.98 BogoMIPS (lpj=399960)
[0.022973] pid_max: default: 32768 minimum: 301
[0.027787] Mount-cache hash table entries: 8192 (order: 4, 65536
bytes, linear)
[0.034992] Mountpoint-cache hash table entries: 8192 (order: 4, 65536
bytes, linear)
[0.043973] rcu: Hierarchical SRCU implementation.
[0.047601] rcu: Max phase no-delay instances is 1000.
[0.053169] EFI services will not be available.
[0.057519] smp: Bringing up secondary CPUs ...
[0.062297] Detected VIPT I-cache on CPU1
[0.062374] CPU1: Booted secondary processor 0x0000000001 [0x410fd034]
[0.062818] Detected VIPT I-cache on CPU2
[0.062877] CPU2: Booted secondary processor 0x0000000002 [0x410fd034]
[0.063297] Detected VIPT I-cache on CPU3
[0.063355] CPU3: Booted secondary processor 0x0000000003 [0x410fd034]
[0.063403] smp: Brought up 1 node, 4 CPUs
[0.097538] SMP: Total of 4 processors activated.
[0.102238] CPU features: detected: 32-bit EL0 Support
[0.107372] CPU features: detected: CRC32 instructions
[0.112555] CPU: All CPU(s) started at EL2
[0.116596] alternatives: applying system-wide alternatives
[0.123288] devtmpfs: initialized
[0.132291] clocksource: jiffies: mask: 0xffffffff max_cycles:
0xffffffff, max_idle_ns: 7645041785100000 ns
[0.136419] futex hash table entries: 1024 (order: 4, 65536 bytes,
linear)
[0.149588] pinctrl core: initialized pinctrl subsystem
[0.150100] DMI not present or invalid.
[0.153425] NET: Registered PF_NETLINK/PF_ROUTE protocol family
[0.159720] DMA: preallocated 512 KiB GFP_KERNEL pool for atomic
allocations
[0.166066] DMA: preallocated 512 KiB GFP_KERNEL|GFP_DMA32 pool for
atomic allocations
[0.173923] audit: initializing netlink subsys (disabled)
[0.179376] audit: type=2000 audit(0.116:1): state=initialized
audit_enabled=0 res=1
[0.179781] hw-breakpoint: found 6 breakpoint and 4 watchpoint
registers.
[0.193870] ASID allocator initialised with 65536 entries
[0.199343] Serial: AMBA PL011 UART driver
[0.224372] HugeTLB: registered 1.00 GiB page size, pre-allocated 0
pages
[0.225525] HugeTLB: 0 KiB vmemmap can be freed for a 1.00 GiB page
[0.231802] HugeTLB: registered 32.0 MiB page size, pre-allocated 0
pages
[0.238594] HugeTLB: 0 KiB vmemmap can be freed for a 32.0 MiB page
```

```
[0.244846] HugeTLB: registered 2.00 MiB page size, pre-allocated 0 pages
[0.251630] HugeTLB: 0 KiB vmemmap can be freed for a 2.00 MiB page
[0.257897] HugeTLB: registered 64.0 KiB page size, pre-allocated 0 pages
[0.264682] HugeTLB: 0 KiB vmemmap can be freed for a 64.0 KiB page
[0.339007] raid6: neonx8 gen() 2265 MB/s
[0.407061] raid6: neonx4 gen() 2218 MB/s
[0.475132] raid6: neonx2 gen() 2119 MB/s
[0.543198] raid6: neonx1 gen() 1806 MB/s
[0.611248] raid6: int64x8 gen() 1415 MB/s
[0.679312] raid6: int64x4 gen() 1562 MB/s
[0.747380] raid6: int64x2 gen() 1396 MB/s
[0.815451] raid6: int64x1 gen() 1034 MB/s
[0.815490] raid6: using algorithm neonx8 gen() 2265 MB/s
[0.887519] raid6: xor() 1655 MB/s, rmw enabled
[0.887564] raid6: using neon recovery algorithm
[0.892003] iommu: Default domain type: Translated
[0.896325] iommu: DMA domain TLB invalidation policy: strict mode
[0.902798] SCSI subsystem initialized
[0.906477] usbcore: registered new interface driver usbfs
[0.911839] usbcore: registered new interface driver hub
[0.917140] usbcore: registered new device driver usb
[0.922247] mc: Linux media interface: v0.10
[0.926452] videodev: Linux video capture interface: v2.00
[0.931933] pps_core: LinuxPPS API ver. 1 registered
[0.936874] pps_core: Software ver. 5.3.6 - Copyright 2005-2007
Rodolfo Giometti <giometti@linux.it>
[0.946017] PTP clock support registered
[0.949941] EDAC MC: Ver: 3.0.0
[0.953337] zynqmp-ipi-mbox mailbox@ff9905c0: Registered ZynqMP IPI
mbox with TX/RX channels.
[0.961818] FPGA manager framework
[0.965117] Advanced Linux Sound Architecture Driver Initialized.
[0.971404] Bluetooth: Core ver 2.22
[0.974641] NET: Registered PF_BLUETOOTH protocol family
[0.979939] Bluetooth: HCI device and connection manager initialized
[0.986293] Bluetooth: HCI socket layer initialized
[0.991163] Bluetooth: L2CAP socket layer initialized
[0.996214] Bluetooth: SCO socket layer initialized
[1.001413] clocksource: Switched to clocksource arch_sys_counter
[1.007328] VFS: Disk quotas dquot_6.6.0
[1.011117] VFS: Dquot-cache hash table entries: 512 (order 0, 4096
bytes)
[1.022985] NET: Registered PF_INET protocol family
[1.023185] IP idents hash table entries: 65536 (order: 7, 524288
bytes, linear)
[1.032964] tcp_listen_portaddr_hash hash table entries: 2048 (order:
3, 32768 bytes, linear)
[1.038787] Table-perturb hash table entries: 65536 (order: 6, 262144
bytes, linear)
[1.046508] TCP established hash table entries: 32768 (order: 6,
262144 bytes, linear)
[1.054625] TCP bind hash table entries: 32768 (order: 8, 1048576
bytes, linear)
[1.062639] TCP: Hash tables configured (established 32768 bind 32768)
[1.068419] UDP hash table entries: 2048 (order: 4, 65536 bytes,
linear)
[1.075112] UDP-Lite hash table entries: 2048 (order: 4, 65536 bytes,
linear)
[1.082325] NET: Registered PF_UNIX/PF_LOCAL protocol family
[1.088159] RPC: Registered named UNIX socket transport module.
[1.093751] RPC: Registered udp transport module.
```

```
[1.098450] RPC: Registered tcp transport module.
[1.103144] RPC: Registered tcp NFSv4.1 backchannel transport module.
[1.109587] PCI: CLS 0 bytes, default 64
[1.113806] Trying to unpack rootfs image as initramfs...
[1.126499] hw perfevents: enabled with armv8_pmuv3 PMU driver, 7
counters available
[1.130003] Initialise system trusted keyrings
[1.133198] workingset: timestamp_bits=46 max_order=20 bucket_order=0
[1.140253] NFS: Registering the id_resolver key type
[1.144564] Key type id_resolver registered
[1.148715] Key type id_legacy registered
[1.152751] nfs4filelayout_init: NFSv4 File Layout Driver
Registering...
[1.159420] nfs4flexfilelayout_init: NFSv4 Flexfile Layout Driver
Registering...
[1.167161] jffs2: version 2.2. (NAND) (SUMMARY) © 2001-2006 Red Hat,
Inc.
[1.210967] NET: Registered PF_ALG protocol family
[1.211043] xor: measuring software checksum speed
[1.218827] 8regs : 2521 MB/sec
[1.223162] 32regs : 2522 MB/sec
[1.227800] arm64_neon : 2351 MB/sec
[1.227960] xor: using function: 32regs (2522 MB/sec)
[1.233022] Key type asymmetric registered
[1.237102] Asymmetric key parser 'x509' registered
[1.242055] Block layer SCSI generic (bsg) driver version 0.4 loaded
(major 244)
[1.249377] io scheduler mq-deadline registered
[1.253896] io scheduler kyber registered
[1.261454] irq-xilinx: mismatch in kind-of-intr param
[1.263026] irq-xilinx: /amba_pl@0/interrupt-controller@80020000:
num_irq=32, sw_irq=0, edge=0x1
[1.305552] Serial: 8250/16550 driver, 4 ports, IRQ sharing disabled
[1.307545] Serial: AMBA driver
[1.315906] brd: module loaded
[1.319395] loop: module loaded
[1.323488] tun: Universal TUN/TAP device driver, 1.6
[1.323648] CAN device driver interface
[1.327783] usbcore: registered new interface driver asix
[1.332187] usbcore: registered new interface driver ax88179_178a
[1.338269] usbcore: registered new interface driver cdc_ether
[1.344079] usbcore: registered new interface driver net1080
[1.349737] usbcore: registered new interface driver cdc_subset
[1.355663] usbcore: registered new interface driver zaurus
[1.361256] usbcore: registered new interface driver cdc_ncm
[1.366884] usbcore: registered new interface driver r8153_ecm
[1.373118] VFIO - User Level meta-driver version: 0.3
[1.378814] usbcore: registered new interface driver uas
[1.383168] usbcore: registered new interface driver usb-storage
[1.390101] rtc_zynqmp ffa60000 rtc: registered as rtc0
[1.394368] rtc_zynqmp ffa60000 rtc: setting system clock to
2023-08-19T12:13:19 UTC (1692447199)
[1.403389] i2c_dev: i2c /dev entries driver
[1.409832] usbcore: registered new interface driver uvcvideo
[1.414471] Bluetooth: HCI UART driver ver 2.3
[1.417680] Bluetooth: HCI UART protocol H4 registered
[1.422801] Bluetooth: HCI UART protocol BCSP registered
[1.428141] Bluetooth: HCI UART protocol LL registered
[1.433242] Bluetooth: HCI UART protocol ATH3K registered
[1.438661] Bluetooth: HCI UART protocol Three-wire (H5) registered
[1.445022] Bluetooth: HCI UART protocol Intel registered
[1.450309] Bluetooth: HCI UART protocol QCA registered
[1.455555] usbcore: registered new interface driver bcm203x
```

```
[1.461208] usbcore: registered new interface driver bpa10x
[1.466782] usbcore: registered new interface driver bfusb
[1.472252] usbcore: registered new interface driver btusb
[1.477740] usbcore: registered new interface driver ath3k
[1.483377] EDAC MC: ECC not enabled
[1.487010] EDAC DEVICE0: Giving out device to module edac controller
cache_err: DEV edac (POLLED)
[1.495716] cortex_edac edac: cortex 11/12 driver is deprecated
[1.502021] EDAC DEVICE1: Giving out device to module zynqmp_ocm-edac
controller zynqmp_ocm: DEV ff960000.memory-controller (INTERRUPT)
[1.514647] sdhci: Secure Digital Host Controller Interface driver
[1.519999] sdhci: Copyright(c) Pierre Ossman
[1.524336] sdhci-pltfm: SDHCI platform and OF driver helper
[1.531234] ledtrig-cpu: registered to indicate activity on CPUs
[1.536090] SMCCC: SOC_ID: ID = jep106:0049:0000 Revision = 0x24738093
[1.542644] zynqmp_firmware_probe Platform Management API v1.1
[1.548413] zynqmp_firmware_probe Trustzone version v1.0
[1.584985] securefw securefw: securefw probed
[1.585458] zynqmp-aes zynqmp-aes.0: will run requests pump with
realtime priority
[1.592235] usbcore: registered new interface driver usbhid
[1.597058] usbhid: USB HID core driver
[1.604266] ARM CCI_400_r1 PMU driver probed
[1.605041] fpga_manager fpga0: Xilinx ZynqMP FPGA Manager registered
[1.612361] usbcore: registered new interface driver snd-usb-audio
[1.618944] pktgen: Packet Generator for packet performance testing.
Version: 2.75
[1.630223] Initializing XFRM netlink socket
[1.630345] NET: Registered PF_INET6 protocol family
[1.635173] Segment Routing with IPv6
[1.638213] In-situ OAM (IOAM) with IPv6
[1.642213] sit: IPv6, IPv4 and MPLS over IPv4 tunneling driver
[1.648470] NET: Registered PF_PACKET protocol family
[1.653068] NET: Registered PF_KEY protocol family
[1.657862] can: controller area network core
[1.662232] NET: Registered PF_CAN protocol family
[1.666982] can: raw protocol
[1.669943] can: broadcast manager protocol
[1.674123] can: netlink gateway - max_hops=1
[1.678694] Bluetooth: RFCOMM TTY layer initialized
[1.683349] Bluetooth: RFCOMM socket layer initialized
[1.688482] Bluetooth: RFCOMM ver 1.11
[1.692228] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
[1.697520] Bluetooth: BNEP filters: protocol multicast
[1.702745] Bluetooth: BNEP socket layer initialized
[1.707698] Bluetooth: HIDP (Human Interface Emulation) ver 1.2
[1.713627] Bluetooth: HIDP socket layer initialized
[1.718615] 8021q: 802.1Q VLAN Support v1.8
[1.723008] 9pnet: Installing 9P2000 support
[1.727055] Key type dns_resolver registered
[1.731547] registered taskstats version 1
[1.735376] Loading compiled-in X.509 certificates
[1.742532] Btrfs loaded, crc32c=crc32c-generic, zoned=no, fsverity=no
[1.746886] alg: No test for xilinx-zynqmp-rsa (zynqmp-rsa)
[2.570357] Freeing initrd memory: 30968K
[2.945124] ff000000.serial: ttyPS0 at MMIO 0xff000000 (irq = 25,
base_baud = 6249375) is a xuartps
[2.954214] printk: console [ttyPS0] enabled
[2.954214] printk: console [ttyPS0] enabled
[2.958521] printk: bootconsole [cdns0] disabled
[2.958521] printk: bootconsole [cdns0] disabled
[2.968138] ff010000.serial: ttyPS1 at MMIO 0xff010000 (irq = 26,
base_baud = 6249375) is a xuartps
```

```
[2.981279] of-fpga-region fpga-full: FPGA Region probed
[2.988919] nwl-pcie fd0e0000.pcie: host bridge /axi/pcie@fd0e0000
ranges:
[2.995824] nwl-pcie fd0e0000.pcie: MEM
0x00e000000..0x00efffffff -> 0x00e0000000
[3.003836] nwl-pcie fd0e0000.pcie: MEM
0x0600000000..0x07ffffffff -> 0x0600000000
[3.011937] nwl-pcie fd0e0000.pcie: Link is UP
[3.016587] nwl-pcie fd0e0000.pcie: PCI host bridge to bus 0000:00
[3.022772] pci_bus 0000:00: root bus resource [bus 00-ff]
[3.028255] pci_bus 0000:00: root bus resource [mem
0xe0000000-0xefffffff]
[3.035129] pci_bus 0000:00: root bus resource [mem
0x600000000-0x7fffffff pref]
[3.042636] pci 0000:00:00.0: [10ee:d021] type 01 class 0x060400
[3.048698] pci 0000:00:00.0: PME# supported from D0 D1 D2 D3hot
[3.056853] pci 0000:01:00.0: [10ec:8168] type 00 class 0x020000
[3.062880] pci 0000:01:00.0: reg 0x10: [io 0x0000-0x00ff]
[3.068483] pci 0000:01:00.0: reg 0x18: [mem 0x00000000-0x00000fff
64bit]
[3.075292] pci 0000:01:00.0: reg 0x20: [mem 0x00000000-0x00003fff
64bit pref]
[3.082642] pci 0000:01:00.0: supports D1 D2
[3.086911] pci 0000:01:00.0: PME# supported from D0 D1 D2 D3hot D3cold
[3.093748] pci 0000:00:00.0: BAR 8: assigned [mem
0xe0000000-0xe00fffff]
[3.100535] pci 0000:00:00.0: BAR 9: assigned [mem
0x600000000-0x6000fffff 64bit pref]
[3.108451] pci 0000:01:00.0: BAR 4: assigned [mem
0x600000000-0x600003fff 64bit pref]
[3.116381] pci 0000:01:00.0: BAR 2: assigned [mem
0xe0000000-0xe0000fff 64bit]
[3.123706] pci 0000:01:00.0: BAR 0: no space for [io size 0x0100]
[3.129967] pci 0000:01:00.0: BAR 0: failed to assign [io size 0x0100]
[3.136574] pci 0000:00:00.0: PCI bridge to [bus 01-0c]
[3.141802] pci 0000:00:00.0: bridge window [mem
0xe0000000-0xe00fffff]
[3.148589] pci 0000:00:00.0: bridge window [mem
0x600000000-0x6000fffff 64bit pref]
[3.160332] xilinx-zynqmp-dpdma fd4c0000.dma-controller: Xilinx DPDMA
engine is probed
[3.168945] spi-nor spi0.0: found mt25qu512a, expected m25p80
[3.174906] spi-nor spi0.0: mt25qu512a (131072 Kbytes)
[3.180147] 3 fixed-partitions partitions found on MTD device spi0.0
[3.186498] Creating 3 MTD partitions on "spi0.0":
[3.191280] 0x000000000000-0x000001e00000 : "qspi-boot"
[3.197339] 0x000001e00000-0x000001e40000 : "qspi-bootenv"
[3.203552] 0x000001e40000-0x000004240000 : "qspi-kernel"
[3.212294] macb ff0e0000.ethernet: Not enabling partial store and
forward
[3.242506] xilinx-axipmon ffa00000.perf-monitor: Probed Xilinx APM
[3.249065] xilinx-axipmon fd0b0000.perf-monitor: Probed Xilinx APM
[3.255564] xilinx-axipmon fd490000.perf-monitor: Probed Xilinx APM
[3.262064] xilinx-axipmon ffa10000.perf-monitor: Probed Xilinx APM
[3.269311] i2c i2c-0: using pinctrl states for GPIO recovery
[3.275271] i2c i2c-0: using generic GPIOs for recovery
[3.280876] pca953x 0-0020: supply vcc not found, using dummy regulator
[3.287599] pca953x 0-0020: using no AI
[3.292274] pca953x 0-0021: supply vcc not found, using dummy regulator
[3.298955] pca953x 0-0021: using no AI
[3.312183] i2c i2c-0: Added multiplexed i2c bus 2
[3.323760] i2c i2c-0: Added multiplexed i2c bus 3
[3.395137] i2c i2c-0: Added multiplexed i2c bus 4
```

```
[3.400056] i2c i2c-0: Added multiplexed i2c bus 5
[3.404852] pca954x 0-0075: registered 4 multiplexed busses for I2C
mux pca9544
[3.412213] cdns-i2c ff020000.i2c: 400 kHz mmio ff020000 irq 53
[3.419156] i2c i2c-1: using pinctrl states for GPIO recovery
[3.425088] i2c i2c-1: using generic GPIOs for recovery
[3.431027] at24 6-0054: supply vcc not found, using dummy regulator
[3.437930] at24 6-0054: 1024 byte 24c08 EEPROM, writable, 1 bytes/
write
[3.444674] i2c i2c-1: Added multiplexed i2c bus 6
[3.450029] si5341 7-0036: no regulator set, defaulting vdd_sel to
2.5V for out
[3.457337] si5341 7-0036: no regulator set, defaulting vdd_sel to
2.5V for out
[3.464642] si5341 7-0036: no regulator set, defaulting vdd_sel to
2.5V for out
[3.471941] si5341 7-0036: no regulator set, defaulting vdd_sel to
2.5V for out
[3.479242] si5341 7-0036: no regulator set, defaulting vdd_sel to
2.5V for out
[3.486544] si5341 7-0036: no regulator set, defaulting vdd_sel to
2.5V for out
[3.493843] si5341 7-0036: no regulator set, defaulting vdd_sel to
2.5V for out
[3.501148] si5341 7-0036: no regulator set, defaulting vdd_sel to
2.5V for out
[3.509568] si5341 7-0036: Chip: 5341 Grade: 1 Rev: 1
[3.548095] i2c i2c-1: Added multiplexed i2c bus 7
[3.555742] si570 8-005d: registered, current frequency 300000000 Hz
[3.562132] i2c i2c-1: Added multiplexed i2c bus 8
[3.581895] si570 9-005d: registered, current frequency 148500000 Hz
[3.588289] i2c i2c-1: Added multiplexed i2c bus 9
[3.593305] si5324 10-0069: si5328 probed
[3.661813] si5324 10-0069: si5328 probe successful
[3.666730] i2c i2c-1: Added multiplexed i2c bus 10
[3.671736] i2c i2c-1: Added multiplexed i2c bus 11
[3.676744] i2c i2c-1: Added multiplexed i2c bus 12
[3.681747] i2c i2c-1: Added multiplexed i2c bus 13
[3.686625] pca954x 1-0074: registered 8 multiplexed busses for I2C
switch pca9548
[3.694575] i2c i2c-1: Added multiplexed i2c bus 14
[3.699596] i2c i2c-1: Added multiplexed i2c bus 15
[3.704606] i2c i2c-1: Added multiplexed i2c bus 16
[3.709617] i2c i2c-1: Added multiplexed i2c bus 17
[3.714635] i2c i2c-1: Added multiplexed i2c bus 18
[3.719654] i2c i2c-1: Added multiplexed i2c bus 19
[3.724675] i2c i2c-1: Added multiplexed i2c bus 20
[3.729689] i2c i2c-1: Added multiplexed i2c bus 21
[3.734570] pca954x 1-0075: registered 8 multiplexed busses for I2C
switch pca9548
[3.742167] cdns-i2c ff030000.i2c: 400 kHz mmio ff030000 irq 54
[3.752118] cdns-wdt fd4d0000.watchdog: Xilinx Watchdog Timer with
timeout 60s
[3.759622] cdns-wdt ff150000.watchdog: Xilinx Watchdog Timer with
timeout 10s
[3.767278] cpufreq: cpufreq_online: CPU0: Running at unlisted initial
frequency: 1199880 KHz, changing to: 1199999 KHz
[3.779710] zynqmp-display fd4a0000.display: vtc bridge property not
present
[3.812040] mmc0: SDHCI controller on ff170000.mmc [ff170000.mmc]
using ADMA 64-bit
[4.168624] xilinx-dp-snd-codec fd4a0000.display:zynqmp-dp-snd-codec0:
Xilinx DisplayPort Sound Codec probed
```

```
[4.169011] mmc0: new high speed SDHC card at address aaaa
[4.178755] xilinx-dp-snd-pcm zynqmp_dp_snd_pcm0: Xilinx DisplayPort
Sound PCM probed
[4.184340] mmcblk0: mmc0:aaaa SL16G 14.8 GiB
[4.192012] xilinx-dp-snd-pcm zynqmp_dp_snd_pcm1: Xilinx DisplayPort
Sound PCM probed
[4.205309] xilinx-dp-snd-card fd4a0000.display:zynqmp_dp-snd-card:
Xilinx DisplayPort Sound Card probed
[4.209048] mmcblk0: p1
[4.214909] OF: graph: no port node found in /axi/display@fd4a0000
[4.223839] xlnx-drm xlnx-drm.0: bound fd4a0000.display (ops
0xfffff800008f3a090)
[5.309441] zynqmp-display fd4a0000.display: [drm] Cannot find any
crtc or sizes
[5.317096] [drm] Initialized xlnx 1.0.0 20130509 for fd4a0000.display
on minor 0
[5.324598] zynqmp-display fd4a0000.display: ZynqMP DisplayPort
Subsystem driver probed
[5.332844] ahci-ceva fd0c0000.ahci: supply ahci not found, using
dummy regulator
[5.340408] ahci-ceva fd0c0000.ahci: supply phy not found, using dummy
regulator
[5.347880] ahci-ceva fd0c0000.ahci: supply target not found, using
dummy regulator
[5.355776] ahci-ceva fd0c0000.ahci: AHCI 0001.0301 32 slots 2 ports 6
Gbps 0x3 impl platform mode
[5.364738] ahci-ceva fd0c0000.ahci: flags: 64bit ncq sntf pm clo only
pmp fbs pio slum part ccc sds apst
[5.375130] scsi host0: ahci-ceva
[5.378749] scsi host1: ahci-ceva
[5.382174] ata1: SATA max UDMA/133 mmio [mem 0xfd0c0000-0xfd0c1fff]
port 0x100 irq 59
[5.390083] ata2: SATA max UDMA/133 mmio [mem 0xfd0c0000-0xfd0c1fff]
port 0x180 irq 59
[5.399718] macb ff0e0000.ethernet: Not enabling partial store and
forward
[5.411491] macb ff0e0000.ethernet eth0: Cadence GEM rev 0x50070106 at
0xff0e0000 irq 51 (ba:af:19:9b:ba:af)
[5.447200] xhci-hcd xhci-hcd.1.auto: xHCI Host Controller
[5.452713] xhci-hcd xhci-hcd.1.auto: new USB bus registered, assigned
bus number 1
[5.460462] xhci-hcd xhci-hcd.1.auto: hcc params 0x0238f625 hci
version 0x100 quirks 0x0000000002010810
[5.469886] xhci-hcd xhci-hcd.1.auto: irq 60, io mem 0xfe200000
[5.475910] xhci-hcd xhci-hcd.1.auto: xHCI Host Controller
[5.481392] xhci-hcd xhci-hcd.1.auto: new USB bus registered, assigned
bus number 2
[5.489048] xhci-hcd xhci-hcd.1.auto: Host supports USB 3.0 SuperSpeed
[5.495693] usb usb1: New USB device found, idVendor=1d6b,
idProduct=0002, bcdDevice= 6.01
[5.503962] usb usb1: New USB device strings: Mfr=3, Product=2,
SerialNumber=1
[5.511181] usb usb1: Product: xHCI Host Controller
[5.516049] usb usb1: Manufacturer: Linux 6.1.30-xilinx-v2023.2 xhci-
hcd
[5.522742] usb usb1: SerialNumber: xhci-hcd.1.auto
[5.527963] hub 1-0:1.0: USB hub found
[5.531747] hub 1-0:1.0: 1 port detected
[5.535992] usb usb2: New USB device found, idVendor=1d6b,
idProduct=0003, bcdDevice= 6.01
[5.544259] usb usb2: New USB device strings: Mfr=3, Product=2,
SerialNumber=1
[5.551477] usb usb2: Product: xHCI Host Controller
```

```
[5.556346] usb usb2: Manufacturer: Linux 6.1.30-xilinx-v2023.2 xhci-hcd
[5.563043] usb usb2: SerialNumber: xhci-hcd.1.auto
[5.568182] hub 2-0:1.0: USB hub found
[5.571951] hub 2-0:1.0: 1 port detected
[5.579024] input: gpio-keys as /devices/platform/gpio-keys/input/input0
[5.586083] of_cfs_init
[5.588540] of_cfs_init: OK
[5.591400] clk: Not disabling unused clocks
[5.595724] ALSA device list:
[5.598684] #0: DP mon
[5.711647] ata1: SATA link down (SStatus 0 SControl 330)
[5.797418] usb 1-1: new high-speed USB device number 2 using xhci-hcd
[5.877417] ata2: SATA link up 6.0 Gbps (SStatus 133 SControl 330)
[5.888643] ata2.00: ATA-9: TS240GSSD220S, S1031B0, max UDMA/133
[5.894654] ata2.00: 468862128 sectors, multi 1: LBA48 NCQ (depth 32)
[5.908985] ata2.00: configured for UDMA/133
[5.913493] scsi 1:0:0:0: Direct-Access ATA TS240GSSD220S
1B0 PQ: 0 ANSI: 5
[5.922398] sd 1:0:0:0: [sda] 468862128 512-byte logical blocks: (240 GB/224 GiB)
[5.929903] sd 1:0:0:0: [sda] Write Protect is off
[5.934747] sd 1:0:0:0: [sda] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA
[5.943849] sd 1:0:0:0: [sda] Preferred minimum I/O size 512 bytes
[5.950974] sd 1:0:0:0: [sda] Attached SCSI disk
[5.956140] Freeing unused kernel memory: 2240K
[5.961735] usb 1-1: New USB device found, idVendor=0951, idProduct=1643, bcdDevice= 1.00
[5.961788] Run /init as init process
[5.969911] usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[5.969918] usb 1-1: Product: DataTraveler G3
[5.980698] usb 1-1: Manufacturer: Kingston
[5.989223] usb 1-1: SerialNumber: 001CC0EC32FEFC51270F2378
[5.995489] usb-storage 1-1:1.0: USB Mass Storage device detected
[6.001970] scsi host2: usb-storage 1-1:1.0
[6.012516] systemd[1]: systemd 251.8+ running in system mode (+PAM -AUDIT -SELINUX -APPARMOR +IMA -SMACK +SECCOMP -GCRYPT -GNUTLS -OPENSSL +ACL +BLKID -CURL -ELFUTILS -FIDO2 -IDN -IPPC +KMOD -LIBCRYPTSETUP +LIBFDISK -PCRE2 -PWQUALITY -P11KIT -QRENCODE -TPM2 -BZIP2 -LZ4 -XZ -ZLIB +ZSTD -BPF_FRAMEWORK +XKBCOMMON +UTMP +SYSVINIT default-hierarchy=hybrid)
[6.044246] systemd[1]: Detected architecture arm64.
```

Welcome to PetaLinux 2023.2+release-S08141938 (langdale)!

```
[6.117592] systemd[1]: Hostname set to <xilinx-zcu102-20232>.
[6.123547] systemd[1]: Initializing machine ID from random generator.
[6.191763] systemd-sysv-generator[190]: SysV service '/etc/init.d/sshd' lacks a native systemd unit file. Automatically generating a unit file for compatibility. Please update package to include a native systemd unit file, in order to make it more safe and robust.
[6.216729] systemd-sysv-generator[190]: SysV service '/etc/init.d/nfsserver' lacks a native systemd unit file. Automatically generating a unit file for compatibility. Please update package to include a native systemd unit file, in order to make it more safe and robust.
[6.241141] systemd-sysv-generator[190]: SysV service '/etc/init.d/inetd.busybox' lacks a native systemd unit file. Automatically generating a unit file for compatibility. Please update package to include a native systemd unit file, in order to make it more safe and robust.
[6.401447] zynqmp-display fd4a0000.display: [drm] Cannot find any crtc or sizes
```

```
[6.460621] systemd[1]: Queued start job for default target Multi-User System.
[6.513540] systemd[1]: Created slice Slice /system/getty.
[OK] Created slice Slice /system/getty.
[6.534894] systemd[1]: Created slice Slice /system/modprobe.
[OK] Created slice Slice /system/modprobe.
[6.558758] systemd[1]: Created slice Slice /system/serial-getty.
[OK] Created slice Slice /system/serial-getty.
[6.582469] systemd[1]: Created slice User and Session Slice.
[OK] Created slice User and Session Slice.
[6.605661] systemd[1]: Started Dispatch Password Requests to Console Directory Watch.
[OK] Started Dispatch Password ...ts to Console Directory Watch.
[6.629617] systemd[1]: Started Forward Password Requests to Wall Directory Watch.
[OK] Started Forward Password R...uests to Wall Directory Watch.
[6.653673] systemd[1]: Reached target Path Units.
[OK] Reached target Path Units.
[6.669526] systemd[1]: Reached target Remote File Systems.
[OK] Reached target Remote File Systems.
[6.689511] systemd[1]: Reached target Slice Units.
[OK] Reached target Slice Units.
[6.705532] systemd[1]: Reached target Swaps.
[OK] Reached target Swaps.
[6.721967] systemd[1]: Listening on RPCbind Server Activation Socket.
[OK] Listening on RPCbind Server Activation Socket.
[6.745500] systemd[1]: Reached target RPC Port Mapper.
[OK] Reached target RPC Port Mapper.
[6.765833] systemd[1]: Listening on Syslog Socket.
[OK] Listening on Syslog Socket.
[6.781676] systemd[1]: Listening on initctl Compatibility Named Pipe.
[OK] Listening on initctl Compatibility Named Pipe.
[6.805947] systemd[1]: Listening on Journal Audit Socket.
[OK] Listening on Journal Audit Socket.
[6.825735] systemd[1]: Listening on Journal Socket (/dev/log).
[OK] Listening on Journal Socket (/dev/log).
[6.845810] systemd[1]: Listening on Journal Socket.
[OK] Listening on Journal Socket.
[6.861944] systemd[1]: Listening on Network Service Netlink Socket.
[OK] Listening on Network Service Netlink Socket.
[6.885823] systemd[1]: Listening on udev Control Socket.
[OK] Listening on udev Control Socket.
[6.905702] systemd[1]: Listening on udev Kernel Socket.
[OK] Listening on udev Kernel Socket.
[6.925736] systemd[1]: Listening on User Database Manager Socket.
[OK] Listening on User Database Manager Socket.
[6.965690] systemd[1]: Mounting Huge Pages File System...
 Mounting Huge Pages File System...
[6.984384] systemd[1]: Mounting POSIX Message Queue File System...
 Mounting POSIX Message Queue File System...
[7.008517] systemd[1]: Mounting Kernel Debug File System...
 Mounting Kernel Debug File System...
[7.025852] systemd[1]: Kernel Trace File System was skipped because of a failed condition check (ConditionPathExists=/sys/kernel/tracing).
[7.053686] systemd[1]: Mounting Temporary Directory /tmp...
 Mounting Temporary Directory /tmp...
[7.069760] systemd[1]: Create List of Static Device Nodes was skipped because of a failed condition check (ConditionFileNotEmpty=/lib/modules/6.1.30-xilinx-v2023.2/modules.devname).
[7.089332] systemd[1]: Starting Load Kernel Module configfs...
 Starting Load Kernel Module configfs...
[7.108677] systemd[1]: Starting Load Kernel Module drm...
[7.114308] scsi 2:0:0:0: Direct-Access Kingston DataTraveler G3
```

```
1.00 PQ: 0 ANSI: 4
 Starting Load Kernel Module drm...
[7.123031] sd 2:0:0:0: [sdb] 7567964 512-byte logical blocks: (3.87
GB/3.61 Gib)
[7.135252] sd 2:0:0:0: [sdb] Write Protect is off
[7.140253] sd 2:0:0:0: [sdb] Write cache: disabled, read cache:
enabled, doesn't support DPO or FUA
[7.151544] sdb:
[7.153537] sd 2:0:0:0: [sdb] Attached SCSI removable disk
[7.177867] systemd[1]: Starting Load Kernel Module fuse...
 Starting Load Kernel Module fuse...
[7.196812] systemd[1]: Starting RPC Bind...
 Starting RPC Bind...
[7.213662] systemd[1]: File System Check on Root Device was skipped
because of a failed condition check (ConditionPathIsReadWrite=!').
[7.226599] systemd[1]: systemd-journald.service: unit configures an
IP firewall, but the local system does not support BPF/cgroup firewalls.
[7.239471] systemd[1]: (This warning is only shown for the first unit
using IP firewalls.)
[7.273871] systemd[1]: Starting Journal Service...
 Starting Journal Service...
[7.290111] systemd[1]: Load Kernel Modules was skipped because all
trigger condition checks failed.
[7.302105] systemd[1]: Mounting NFSD configuration filesystem...
 Mounting NFSD configuration filesystem...
[7.320822] systemd[1]: Starting Generate network units from Kernel
command line...
 Starting Generate network ...ts from Kernel command line...
[7.373945] systemd[1]: Starting Remount Root and Kernel File
Systems...
 Starting Remount Root and Kernel File Systems...
[7.400786] systemd[1]: Starting Apply Kernel Variables...
 Starting Apply Kernel Variables...
[7.420887] systemd[1]: Starting Coldplug All udev Devices...
 Starting Coldplug All udev Devices...
[7.442553] systemd[1]: Started RPC Bind.
[OK] Started RPC Bind.
[7.457934] systemd[1]: Started Journal Service.
[OK] Started Journal Service.
[OK] Mounted Huge Pages File System.
[OK] Mounted POSIX Message Queue File System.
[OK] Mounted Kernel Debug File System.
[OK] Mounted Temporary Directory /tmp.
[OK] Finished Load Kernel Module configfs.
[OK] Finished Load Kernel Module drm.
[OK] Finished Load Kernel Module fuse.
[OK] Mounted NFSD configuration filesystem.
[OK] Finished Generate network units from Kernel command line.
[OK] Finished Remount Root and Kernel File Systems.
[OK] Finished Apply Kernel Variables.
[OK] Reached target Preparation for Network.
 Mounting Kernel Configuration File System...
 Starting Flush Journal to Persistent Storage...
[7.728702] systemd-journald[200]: Received client request to flush
runtime journal.
 Starting Create System Users...
[OK] Mounted Kernel Configuration File System.
[OK] Finished Flush Journal to Persistent Storage.
[OK] Finished Create System Users.
 Starting Create Static Device Nodes in /dev...
[OK] Finished Create Static Device Nodes in /dev.
[OK] Reached target Preparation for Local File Systems.
 Mounting /var/volatile...
```

```
Starting Rule-based Manager for Device Events and Files...
[OK] Mounted /var/volatile.
 Starting Load/Save Random Seed...
[OK] Reached target Local File Systems.
 Starting Rebuild Dynamic Linker Cache...
 Starting Create Volatile Files and Directories...
[OK] Started Rule-based Manager for Device Events and Files.
[OK] Finished Rebuild Dynamic Linker Cache.
[OK] Finished Create Volatile Files and Directories.
 Starting Run pending postinsts...
 Starting Rebuild Journal Catalog...
 Starting Network Configuration...
[8.132520] Unloading old XRT Linux kernel modules
 Starting Network Name Resolution...
 Starting Network Time Synchronization...
[8.168666] Loading new XRT Linux kernel modules
 Starting Record System Boot/Shutdown in UTMP...
[8.185109] zocl: loading out-of-tree module taints kernel.
[OK] Finished Rebuild Journal Catalog.
[8.212295] [drm] Probing for xlnx,zocl
[8.217091] zocl-drm amba_pl@0:zyxclmm_drm: error -ENXIO: IRQ index 32
not found
[8.225101] [drm] FPGA programming device pcap founded.
[8.230583] [drm] PR[0] Isolation addr 0x0
[8.232213] [drm] Initialized zocl 2.15.0 20230819 for
amba_pl@0:zyxclmm_drm on minor 1
 Starting Update is Completed...
[OK] Finished Update is Completed.
[OK] Finished Record System Boot/Shutdown[8.290296] INFO:
Creating ICD entry for Xilinx Platform
in UTMP.
[8.640448] cfg80211: Loading compiled-in X.509 certificates for
regulatory database
[8.741440] random: crng init done
[8.853555] cfg80211: Loaded X.509 cert 'sforshee: 00b28ddf47aef9cea7'
[8.860453] platform regulatory.0: Direct firmware load for
regulatory.db failed with error -2
[8.860468] cfg80211: failed to load regulatory.db
[OK] Finished Load/Save Random Seed.
[OK] Started Network Configuration.
[OK] Started Network Name Resolution.
[OK] Finished Coldplug All udev Devices.
[OK] Finished Run pending postinsts.
[OK] Started Network Time Synchronization.
[9.438503] macb ff0e0000.ethernet eth0: PHY [ff0e0000.ethernet-
ffffffff:0c] driver [TI DP83867] (irq=_POLL)
[9.448298] macb ff0e0000.ethernet eth0: configuring for phy/rgmii-id
link mode
[9.478999] pps pps0: new PPS source ptp0
[9.502066] macb ff0e0000.ethernet: gem-ptp-timer ptp clock registered.
[OK] Created slice Slice /system/systemd-fsck.
[OK] Reached target Network.
[OK] Reached target Host and Network Name Lookups.
[OK] Reached target Sound Card.
[OK] Reached target System Initialization.
[OK] Started Daily Cleanup of Temporary Directories.
[OK] Reached target System Time Set.
[OK] Reached target Timer Units.
[OK] Listening on D-Bus System Message Bus Socket.
 Starting sshd.socket...
[OK] Listening on Load/Save RF Switch Status /dev/rfkill Watch.
[OK] Started NFS status monitor for NFSv2/3 locking..
 Starting File System Check on /dev/sda...
```

```
Starting File System Check on /dev/sdb...
[OK] Listening on sshd.socket.
[OK] Reached target Socket Units.
[OK] Reached target Basic System.
[OK] Started Kernel Logging Service.
[OK] Started System Logging Service.
 Starting D-Bus System Message Bus...
 Starting inetc.busybox.service...
 Starting LSB: Kernel NFS server support...
 Starting User Login Management...
 Starting Permit User Sessions...
 Starting Target Communication Framework agent...
 Starting OpenSSH Key Generation...
[OK] Started D-Bus System Message Bus.
[OK] Finished File System Check on /dev/sda.
[OK] Started inetc.busybox.service.
[OK] Finished Permit User Sessions.
[OK] Started Target Communication Framework agent.
 Mounting /run/media/root-sda...
[OK] Started Getty on ttym.
[OK] Started Serial Getty on ttymS0.
[OK] Reached target Login Prompts.
[OK] Found device /dev/mmcblk0p1.
 Starting File System Check on /dev[11.314729] EXT4-fs (sda):
mounted filesystem with ordered data mode. Quota mode: none.
/mmcblk0p1...
[OK] Mounted /run/media/root-sda.
[OK] Started User Login Management.
[OK] Finished File System Check on /dev/mmcblk0p1.
 Mounting /run/media/boot-mmcblk0p1...
[OK] Mounted /run/media/boot-mmcblk0p1.
[11.893649] sdb:
[OK] Finished File System Check on /dev/sdb.
 Mounting /run/media/sdb...
[OK] Mounted /run/media/sdb.
[12.313708] NFSD: Using /var/lib/nfs/v4recovery as the NFSv4 state
recovery directory
[12.321582] NFSD: Using legacy client tracking operations.
[12.327075] NFSD: starting 90-second grace period (net f0000000)
[FAILED] Failed to start LSB: Kernel NFS server support.
See 'systemctl status nfsserver.service' for details.
[OK] Reached target Multi-User System.
 Starting Record Runlevel Change in UTMP...
[OK] Finished Record Runlevel Change in UTMP.
[13.569717] macb ff0e0000.ethernet eth0: unable to generate target
frequency: 125000000 Hz
[13.579132] macb ff0e0000.ethernet eth0: Link is Up - 1Gbps/Full -
flow control tx
[13.586831] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready

PetaLinux 2023.2+release-S08141938 xilinx-zcu102-20232 ttymS0

xilinx-zcu102-20232 login: petalinux
You are required to change your password immediately (administrator
enforced).
New password:
Retype new password:
[50.419540] audit: type=1006 audit(1692447248.519:2): pid=767 uid=0
old-auid=4294967295 auid=1000 tty=(none) old-ses=4294967295 ses=1 res=1
[50.432144] audit: type=1300 audit(1692447248.519:2): arch=c00000b7
syscall=64 success=yes exit=4 a0=8 a1=fffffe0d3aef0 a2=4 a3=1 items=0
ppid=1 pid=767 auid=1000 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0
```

```
fsgid=0 tty=(none) ses=1 comm="(systemd)" exe="/lib/systemd/systemd"
key=(null)
[50.457759] audit: type=1327 audit(1692447248.519:2):
proctitle="(systemd)"
xilinx-zcu102-20232:~$
```

- For MicroBlaze processors, the previous commands download the bitstream to the target board, and boot the kernel image on the target board.
- For Zynq 7000 devices, they download the bitstream and FSBL to the target board, and boot the U-Boot and the kernel on the target board.
- For Zynq UltraScale+ MPSoC, they download the bitstream, PMU firmware, and FSBL, and boot the kernel with help of `linux-boot.elf` to set kernel start and DTB addresses.
- For AMD Versal™ adaptive SoC, they download the BOOT.BIN (which contains the PDI, PLM firmware, PSM firmware, U-Boot, and DTB) and boot the kernel (Image) with the help of U-Boot script (`boot.scr`).

## **`petalinux-boot --qemu`**

The `petalinux-boot --qemu` command boots the MicroBlaze™ CPU, AMD Zynq™ UltraScale+™ MPSoC, AMD Versal™ adaptive SoC, or Zynq 7000 devices with a PetaLinux image using the QEMU emulator. Many QEMU options require superuser (root) access to operate properly. The `--root` option enables root mode and prompts you for sudo credentials.

**Note:** For Versal adaptive SoC, you require BOOT.BIN to boot on QEMU. Check [petalinux-package](#) on how to create BOOT.BIN for Versal adaptive SoC.

### **`petalinux-boot --qemu Options`**

The following table contains details of options specific to the QEMU boot workflow:

**Table 51: `petalinux-boot --qemu Options`**

| Otion                           | Functional Description                                                                                           | Value Range       | Default Value   |
|---------------------------------|------------------------------------------------------------------------------------------------------------------|-------------------|-----------------|
| <code>--root</code>             | Boot in root mode                                                                                                | None              | None            |
| <code>--iptables-allowed</code> | Whether to allow to implement iptables commands. This is optional and applicable only in root mode.              | None              | None            |
| <code>--net-intf</code>         | Network interface on the host to bridge with the QEMU subnet. This option applies for root mode only.            | User-specified    | eth0            |
| <code>--qemu-args</code>        | Extra arguments to QEMU command. This is optional.                                                               | None              | None            |
| <code>--subnet SUBNET</code>    | Specifies subnet gateway IP and the number of valid bit of network mask. This option applies for root mode only. | User-specified    | 192.168.10.1/24 |
| <code>--dhcpd</code>            | Enable or disable dhcpcd. This is optional and applicable only for root mode.                                    | Enable<br>Disable | Enable          |
| <code>--tftp</code>             | Path to tftp boot directory                                                                                      | User-specified    | None            |

**Table 51: petalinux-boot --qemu Options (cont'd)**

| Otion           | Functional Description                                                                                 | Value Range    | Default Value                                       |
|-----------------|--------------------------------------------------------------------------------------------------------|----------------|-----------------------------------------------------|
| --pmu-qemu-args | Extra arguments for PMU instance of QEMU. This is optional.                                            | User-specified | None                                                |
| --rootfs        | The cpio rootfile system to use for QEMU boot. Supports Zynq, Zynq UltraScale+ MPSoC, and MicroBlaze . | User-specified | <plnx-proj-root>/images/linux/rootfs.cpio.gz.u-boot |
| --qemu-no-gdb   | To disable GDB through QEMU boot.                                                                      | None           | None                                                |

## ***petalinux-boot --qemu Examples***

The following examples demonstrate the correct usage of the `petalinux-boot --qemu` command.

- Load and boot a prebuilt U-Boot elf using QEMU.

```
petalinux-boot --qemu --prebuilt 2
[INFO] Sourcing buildtools
INFO: No DTB has been specified, use the default one "xilinx-zcu102-2023.2/pre-built/linux/images/system.dtb".
INFO: Starting microblaze QEMU
INFO: Starting the above QEMU command in the background
INFO: qemu-system-microblazeel -M microblaze-fdt -serial mon:stdio -serial /dev/null -display none -kernel /wrk/everest_fcv_nobkup/kavyasre/_20230819/_xilinx-zcu102-2023.2/pre-built/linux/images/pmu_rom_qemu_sha3.elf -device loader,file=/wrk/everest_fcv_nobkup/kavyasre/_20230819/_xilinx-zcu102-2023.2/pre-built/linux/images/pmufw.elf -hw-dtb /wrk/everest_fcv_nobkup/kavyasre/_20230819/_xilinx-zcu102-2023.2/pre-built/linux/images/zynqmp-qemu-multiarch-pmu.dtb -machine-path /tmp/tmp.pjOkTRdGR7 -device
loader,addr=0xfd1a0074,data=0x1011003,data-len=4 -device
loader,addr=0xfd1a007C,data=0x1010f03,data-len=4
qemu-system-microblazeel: Failed to connect to '/tmp/tmp.pjOkTRdGR7/qemu-report-_pmu@0': No such file or directory
qemu-system-microblazeel: info: QEMU waiting for connection on:
disconnected: unix:/tmp/tmp.pjOkTRdGR7/qemu-report-_pmu@0,server=on
INFO: TCP PORT is free 9000
INFO: Starting aarch64 QEMU
INFO: qemu-system-aarch64 -M arm-generic-fdt -serial mon:stdio -serial /dev/null -display none -device loader,file=/wrk/everest_fcv_nobkup/kavyasre/_20230819/_xilinx-zcu102-2023.2/pre-built/linux/images/bl31.elf,cpu-num=0 -device loader,file=/wrk/everest_fcv_nobkup/kavyasre/_20230819/_xilinx-zcu102-2023.2/pre-built/linux/images/system.dtb,addr=0x00100000,force-raw-on -device
loader,file=/wrk/everest_fcv_nobkup/kavyasre/_20230819/_xilinx-zcu102-2023.2/pre-built/linux/images/u-boot.elf -gdb tcp::9000 -net nic -net nic -net nic,netdev=eth0 -netdev user,id=eth0,tftp=/tftpboot -hw-dtb /wrk/everest_fcv_nobkup/kavyasre/_20230819/_xilinx-zcu102-2023.2/pre-built/linux/images/zynqmp-qemu-multiarch-arm.dtb -machine-path /tmp/tmp.pjOkTRdGR7 -global xlnx,zynqmp-boot.cpu-num=0 -global xlnx,zynqmp-boot.use-pmufw=true -m 4G
qemu-system-aarch64: warning: hub 0 is not connected to host network
QEMU 7.1.0 monitor - type 'help' for more information
(qemu) PMU Firmware 2023.2 Aug 11 2023 10:10:30
PMU_ROM Version: xpbr-v8.1.0-0
NOTICE: BL31: Secure code at 0x60000000
NOTICE: BL31: Non secure code at 0x80000000
```

```
NOTICE: BL31: v2.8(release):xlnx_rebase_v2.8_2023.2_ksb-14-gf0ba7ad93
NOTICE: BL31: Built : 04:12:26, Jul 5 2023

U-Boot 2023.01 (Aug 11 2023 - 04:59:53 +0000)

CPU: ZynqMP
Silicon: v3
Chip: unknown
Model: ZynqMP ZCU102 Rev1.0
Board: Xilinx ZynqMP
DRAM: 2 GiB (effective 4 GiB)
PMUFW: v1.1
PMUFW: No permission to change config object
EL Level: EL2
Secure Boot: not authenticated, not encrypted
Core: 75 devices, 31 uclasses, devicetree: board
NAND: 0 MiB
MMC: mmc@ff170000: 0
Loading Environment from nowhere... OK
In: serial
Out: serial
Err: serial
Bootmode: JTAG_MODE
Reset reason:
ofnode_read_bootscript_address: Missing /u-boot node
ofnode_read_bootscript_flash: Missing /u-boot node
Net:
ZYNO GEM: ff0e0000, mdio bus ff0e0000, phyaddr 12, interface rgmii-id

Warning: ethernet@ff0e0000 (eth0) using random MAC address -
56:e9:52:d9:7e:04
eth0: ethernet@ff0e0000
scanning bus for devices...
SATA link 0 timeout.
SATA link 1 timeout.
AHCI 0001.0000 32 slots 2 ports 1.5 Gbps 0x3 impl SATA mode
flags: 64bit ncq only
starting USB...
Bus usb@fe200000: Register 8000402 NbrPorts 8
Starting the controller
USB XHCI 1.00
scanning bus usb@fe200000 for devices... 1 USB Device(s) found
scanning usb for storage devices... 0 Storage Device(s) found
Hit any key to stop autoboot: 0
JTAG: Trying to boot script at 20000000
Executing script at 20000000
Wrong image format for "source" command
JTAG: SCRIPT FAILED: continuing...
BOOTP broadcast 1
DHCP client bound to address 10.0.2.15 (2 ms)
*** Warning: no boot file name; using '0A00020F.img'
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename '0A00020F.img'.
Load address: 0x8000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
missing environment variable: pxeuuid
Retrieving file: pxelinux.cfg/01-56-e9-52-d9-7e-04
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
```

```
Filename 'pxelinux.cfg/01-56-e9-52-d9-7e-04'.
Load address: 0x10000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
Retrieving file: pxelinux.cfg/0A00020F
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'pxelinux.cfg/0A00020F'.
Load address: 0x10000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
Retrieving file: pxelinux.cfg/0A00020
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'pxelinux.cfg/0A00020'.
Load address: 0x10000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
Retrieving file: pxelinux.cfg/0A0002
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'pxelinux.cfg/0A0002'.
Load address: 0x10000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
Retrieving file: pxelinux.cfg/0A000
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'pxelinux.cfg/0A000'.
Load address: 0x10000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
Retrieving file: pxelinux.cfg/0A00
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'pxelinux.cfg/0A00'.
Load address: 0x10000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
Retrieving file: pxelinux.cfg/0A0
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'pxelinux.cfg/0A0'.
Load address: 0x10000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
Retrieving file: pxelinux.cfg/0A
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'pxelinux.cfg/0A'.
Load address: 0x10000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
Retrieving file: pxelinux.cfg/0
Using ethernet@ff0e0000 device
```

```
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'pxelinux.cfg/0'.
Load address: 0x10000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
Retrieving file: pxelinux.cfg/default-arm-zynqmp-zynqmp
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'pxelinux.cfg/default-arm-zynqmp-zynqmp'.
Load address: 0x10000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
Retrieving file: pxelinux.cfg/default-arm-zynqmp
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'pxelinux.cfg/default-arm-zynqmp'.
Load address: 0x10000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
Retrieving file: pxelinux.cfg/default-arm
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'pxelinux.cfg/default-arm'.
Load address: 0x10000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
Retrieving file: pxelinux.cfg/default
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'pxelinux.cfg/default'.
Load address: 0x10000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
Config file not found
BOOTP broadcast 1
DHCP client bound to address 10.0.2.15 (0 ms)
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'boot.scr.uimg'.
Load address: 0x20000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
BOOTP broadcast 1
DHCP client bound to address 10.0.2.15 (0 ms)
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'boot.scr.uimg'.
Load address: 0x18000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
JTAG: Trying to boot script at 20000000
Executing script at 20000000
Wrong image format for "source" command
JTAG: SCRIPT FAILED: continuing...
MMC: no card present
MMC Device 1 not found
```

```
no mmc device at slot 1
SF: Detected n25q512a with page size 512 Bytes, erase size 128 KiB, total
128 MiB
device 0 offset 0x3e80000, size 0x80000
SF: 524288 bytes @ 0x3e80000 Read: OK
QSPI: Trying to boot script at 20000000
Executing script at 20000000
Wrong image format for "source" command
QSPI: SCRIPT FAILED: continuing...

no devices available
NAND: SCRIPT FAILED: continuing...

Device 0: unknown device

Device 1: unknown device

Device 0: unknown device
BOOTP broadcast 1
DHCP client bound to address 10.0.2.15 (0 ms)
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'boot.scr.uimg'.
Load address: 0x8000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
missing environment variable: pxeuuid
Retrieving file: pxelinux.cfg/01-56-e9-52-d9-7e-04
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'pxelinux.cfg/01-56-e9-52-d9-7e-04'.
Load address: 0x10000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
Retrieving file: pxelinux.cfg/0A00020F
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'pxelinux.cfg/0A00020F'.
Load address: 0x10000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
Retrieving file: pxelinux.cfg/0A00020
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'pxelinux.cfg/0A00020'.
Load address: 0x10000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
Retrieving file: pxelinux.cfg/0A0002
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'pxelinux.cfg/0A0002'.
Load address: 0x10000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
Retrieving file: pxelinux.cfg/0A000
Using ethernet@ff0e0000 device
```

```
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'pxelinux.cfg/0A000'.
Load address: 0x10000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
Retrieving file: pxelinux.cfg/0A00
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'pxelinux.cfg/0A00'.
Load address: 0x10000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
Retrieving file: pxelinux.cfg/0A0
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'pxelinux.cfg/0A0'.
Load address: 0x10000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
Retrieving file: pxelinux.cfg/0A
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'pxelinux.cfg/0A'.
Load address: 0x10000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
Retrieving file: pxelinux.cfg/0
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'pxelinux.cfg/0'.
Load address: 0x10000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
Retrieving file: pxelinux.cfg/default-arm-zynqmp-zynqmp
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'pxelinux.cfg/default-arm-zynqmp-zynqmp'.
Load address: 0x10000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
Retrieving file: pxelinux.cfg/default-arm-zynqmp
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'pxelinux.cfg/default-arm-zynqmp'.
Load address: 0x10000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
Retrieving file: pxelinux.cfg/default-arm
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'pxelinux.cfg/default-arm'.
Load address: 0x10000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
Retrieving file: pxelinux.cfg/default
```

```

Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'pxelinux.cfg/default'.
Load address: 0x10000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
Config file not found
BOOTP broadcast 1
DHCP client bound to address 10.0.2.15 (0 ms)
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'boot.scr.uimg'.
Load address: 0x20000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
BOOTP broadcast 1
DHCP client bound to address 10.0.2.15 (1 ms)
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'boot.scr.uimg'.
Load address: 0x18000000
Loading: *
TFTP error: 'File not found' (1)
Not retrying...
ZynqMP>

```

- Load and boot a prebuilt U-Boot elf using QEMU in root mode.

```
petalinux-boot --qemu --root --prebuilt 2
```

- Specify the cpio rootfile system for QEMU boot. Supports Zynq, Zynq UltraScale+ MPSoC, and MicroBlaze.
- Boot customized rootfs image with kernel using QEMU:

```

petalinux-boot --qemu --kernel --rootfs images/linux/rootfs.cpio.gz.uboot
[INFO] Sourcing buildtools
ERROR: Failed to boot. Rootfs file /wrk/everest_fcv_nobkup/kavyasre/_20230819/_xilinx-zcu102-2023.2/images/linux/rootfs.cpio.gz.uboot does not exist..
ssw-regress:xilinx-zcu102-2023.2$ petalinux-boot --qemu --kernel --rootfs images/linux/rootfs.cpio.gz.u-boot
[INFO] Sourcing buildtools
Image resized.
INFO: No DTB has been specified, use the default one "xilinx-zcu102-2023.2/images/linux/system.dtb".
INFO: Starting microblaze QEMU
INFO: Starting the above QEMU command in the background
INFO: qemu-system-microblazeel -M microblaze-fdt -serial mon:stdio -serial /dev/null -display none -kernel xilinx-zcu102-2023.2/images/linux/pmu_rom_qemu_sha3.elf -device loader,file=/wrk/everest_fcv_nobkup/kavyasre/_20230819/_xilinx-zcu102-2023.2/images/linux/pmufw.elf -hw-dtb /wrk/everest_fcv_nobkup/kavyasre/_20230819/_xilinx-zcu102-2023.2/images/linux/zynqmp-qemu-multiarch-pmu.dtb -machine-path /tmp/tmp.utHmiXaPUB -device loader,addr=0xfd1a0074,data=0x1011003,data-len=4 -device loader,addr=0xfd1a007C,data=0x1010f03,data-len=4
qemu-system-microblazeel: Failed to connect to '/tmp/tmp.utHmiXaPUB/qemu-report-_pmu@0': No such file or directory
qemu-system-microblazeel: info: QEMU waiting for connection on: disconnected:unix:/tmp/tmp.utHmiXaPUB/qemu-report-_pmu@0,server=on

```

```
INFO: TCP PORT is free 9000
INFO: Starting aarch64 QEMU
INFO: qemu-system-aarch64 -M arm-generic-fdt -serial mon:stdio -
 serial /dev/null -display none -device loader,file=xilinx-zcu102-2023.2/
 images/linux/zynqmp-qemu-multiarch-arm.dtb -machine-path /tmp/
 tmp.utHmiXaPUB -global xlnx,zynqmp-boot.cpu-num=0 -global xlnx,zynqmp-
 boot.use-pmufw=true -drive if=sd,format=raw,index=1,file=/wrk/
 everest_fcv_nobkup/kavyasre/_20230819_/xilinx-zcu102-2023.2/images/linux/
 rootfs.ext4 -m 4G
qemu-system-aarch64: warning: hub 0 is not connected to host network
QEMU 7.1.0 monitor - type 'help' for more information
(qemu) PMU Firmware 2023.2 Aug 11 2023 10:10:30
PMU_ROM Version: xpbr-v8.1.0-0
NOTICE: BL31: Secure code at 0x60000000
NOTICE: BL31: Non secure code at 0x80000000
NOTICE: BL31: v2.8(release):xlnx_rebase_v2.8_2023.2_ksb-14-gf0ba7ad93
NOTICE: BL31: Built : 04:12:26, Jul 5 2023
```

U-Boot 2023.01 (Aug 11 2023 - 04:59:53 +0000)

```
CPU: ZynqMP
Silicon: v3
Chip: unknown
Model: ZynqMP ZCU102 Rev1.0
Board: Xilinx ZynqMP
DRAM: 2 GiB (effective 4 GiB)
PMUFW: v1.1
PMUFW: No permission to change config object
EL Level: EL2
Secure Boot: not authenticated, not encrypted
Core: 75 devices, 31 uclasses, devicetree: board
NAND: 0 MiB
MMC: mmc@ff170000: 0
Loading Environment from nowhere... OK
In: serial
Out: serial
Err: serial
Bootmode: JTAG_MODE
Reset reason:
ofnode_read_bootscript_address: Missing /u-boot node
ofnode_read_bootscript_flash: Missing /u-boot node
Net:
ZYNN GEM: ff0e0000, mdio bus ff0e0000, phyaddr 12, interface rgmii-id

Warning: ethernet@ff0e0000 (eth0) using random MAC address -
6e:69:71:f1:31:76
eth0: ethernet@ff0e0000
scanning bus for devices...
SATA link 0 timeout.
SATA link 1 timeout.
AHCI 0001.0000 32 slots 2 ports 1.5 Gbps 0x3 impl SATA mode
flags: 64bit ncq only
starting USB...
Bus usb@fe200000: Register 8000402 NbrPorts 8
Starting the controller
USB XHCI 1.00
scanning bus usb@fe200000 for devices... 1 USB Device(s) found
scanning usb for storage devices... 0 Storage Device(s) found
Hit any key to stop autoboot: 0
JTAG: Trying to boot script at 20000000
Executing script at 20000000
Trying to load boot images from jtag
```

```
Loading init Ramdisk from Legacy Image at 04000000 ...
Image Name: petalinux-image-minimal-xilinx-z
Created: 2011-04-05 23:00:00 UTC
Image Type: AArch64 Linux RAMDisk Image (uncompressed)
Data Size: 31711975 Bytes = 30.2 MiB
Load Address: 00000000
Entry Point: 00000000
Verifying Checksum ... OK
Flattened Device Tree blob at 00100000
Booting using the fdt blob at 0x100000
Working FDT set to 100000
Loading Ramdisk to 79db3000, end 7bbf12e7 ... OK
Loading Device Tree to 0000000079da2000, end 0000000079db2f76 ... OK
Working FDT set to 79da2000

Starting kernel ...

[0.000000] Booting Linux on physical CPU 0x000000000000 [0x410fd034]
[0.000000] Linux version 6.1.30-xilinx-v2023.2 (oe-user@oe-host)
(aarch64-xilinx-linux-gcc (GCC) 12.2.0, GNU ld (GNU Binutils)
2.39.0.20220819) #1 SMP Fri Aug 4 10:14:47 UTC 2023
[0.000000] Machine model: ZynqMP ZCU102 Rev1.0
[0.000000] earlycon: cdns0 at MMIO 0x00000000ff000000 (options
'115200n8')
[0.000000] printk: bootconsole [cdns0] enabled
[0.000000] efi: UEFI not found.
[0.000000] Zone ranges:
[0.000000] DMA32 [mem 0x0000000000000000-0x00000000ffffffffff]
[0.000000] Normal [mem 0x0000000100000000-0x000000087ffffffff]
[0.000000] Movable zone start for each node
[0.000000] Early memory node ranges
[0.000000] node 0: [mem 0x0000000000000000-0x000000007fefffff]
[0.000000] node 0: [mem 0x0000000800000000-0x000000087ffffffff]
[0.000000] Initmem setup node 0 [mem
0x0000000000000000-0x000000087ffffffff]
[0.000000] On node 0, zone Normal: 256 pages in unavailable ranges
[0.000000] cma: Reserved 256 MiB at 0x0000000069c00000
[0.000000] psci: probing for conduit method from DT.
[0.000000] psci: PSCIv1.1 detected in firmware.
[0.000000] psci: Using standard PSCI v0.2 function IDs
[0.000000] psci: MIGRATE_INFO_TYPE not supported.
[0.000000] psci: SMC Calling Convention v1.2
[0.000000] percpu: Embedded 18 pages/cpu s35816 r8192 d29720 u73728
[0.000000] Detected VIPT I-cache on CPU0
[0.000000] CPU features: detected: ARM erratum 843419
[0.000000] CPU features: detected: ARM erratum 845719
[0.000000] alternatives: applying boot alternatives
[0.000000] Built 1 zonelists, mobility grouping on. Total pages:
1031940
[0.000000] Kernel command line: earlycon console=ttyPS0,115200
clk_ignore_unused root=/dev/ram0 rw init_fatal_sh=1
[0.000000] Unknown kernel command line parameters "init_fatal_sh=1",
will be passed to user space.
[0.000000] Dentry cache hash table entries: 524288 (order: 10,
4194304 bytes, linear)
[0.000000] Inode-cache hash table entries: 262144 (order: 9, 2097152
bytes, linear)
[0.000000] mem auto-init: stack:all(zero), heap alloc:off, heap
free:off
[0.000000] software IO TLB: area num 4.
[0.000000] software IO TLB: mapped [mem
0x0000000007bf00000-0x0000000007ff00000] (64MB)
[0.000000] Memory: 3730672K/4193280K available (14656K kernel code,
```

```
1028K rwdata, 4184K rodata, 2240K init, 377K bss, 200464K reserved,
262144K cma-reserved)
[0.000000] rcu: Hierarchical RCU implementation.
[0.000000] rcu: RCU event tracing is enabled.
[0.000000] rcu: RCU restricting CPUs from NR_CPUS=16 to
nr_cpu_ids=4.
[0.000000] rcu: RCU calculated value of scheduler-enlistment delay is
25 jiffies.
[0.000000] rcu: Adjusting geometry for rcu_fanout_leaf=16,
nr_cpu_ids=4
[0.000000] NR IRQS: 64, nr_irqs: 64, preallocated irqs: 0
[0.000000] GIC: Adjusting CPU interface base to 0x00000000f902f000
[0.000000] Root IRQ handler: gic_handle_irq
[0.000000] GIC: Using split EOI/Deactivate mode
[0.000000] rcu: srcu_init: Setting srcu_struct sizes based on
contention.
[0.000000] arch_timer: cp15 timer(s) running at 65.00MHz (phys).
[0.000000] clocksource: arch_sys_counter: mask: 0x1fffffffffffff
max_cycles: 0xefdb196da, max_idle_ns: 440795204367 ns
[0.000090] sched_clock: 57 bits at 65MHz, resolution 15ns, wraps
every 2199023255550ns
[0.012364] Console: colour dummy device 80x25
[0.014048] Calibrating delay loop (skipped), value calculated using
timer frequency.. 130.00 BogoMIPS (lpj=260000)
[0.014519] pid_max: default: 32768 minimum: 301
[0.021031] Mount-cache hash table entries: 8192 (order: 4, 65536
bytes, linear)
[0.021483] Mountpoint-cache hash table entries: 8192 (order: 4, 65536
bytes, linear)
[0.064347] rcu: Hierarchical SRCU implementation.
[0.064538] rcu: Max phase no-delay instances is 1000.
[0.069437] EFI services will not be available.
[0.073049] smp: Bringing up secondary CPUs ...
[0.088126] Detected VIPT I-cache on CPU1
[0.092117] CPU1: Booted secondary processor 0x0000000001 [0x410fd034]
[0.106547] Detected VIPT I-cache on CPU2
[0.108456] CPU2: Booted secondary processor 0x0000000002 [0x410fd034]
[0.117862] Detected VIPT I-cache on CPU3
[0.119609] CPU3: Booted secondary processor 0x0000000003 [0x410fd034]
[0.121026] smp: Brought up 1 node, 4 CPUs
[0.124700] SMP: Total of 4 processors activated.
[0.125341] CPU features: detected: 32-bit EL0 Support
[0.125845] CPU features: detected: CRC32 instructions
[0.129382] CPU: All CPU(s) started at EL2
[0.130026] alternatives: applying system-wide alternatives
[0.153389] devtmpfs: initialized
[0.186279] clocksource: jiffies: mask: 0xffffffff max_cycles:
0xffffffff, max_idle_ns: 7645041785100000 ns
[0.187263] futex hash table entries: 1024 (order: 4, 65536 bytes,
linear)
[0.198873] pinctrl core: initialized pinctrl subsystem
[0.211580] DMI not present or invalid.
[0.218599] NET: Registered PF_NETLINK/PF_ROUTE protocol family
[0.232291] DMA: preallocated 512 KiB GFP_KERNEL pool for atomic
allocations
[0.233614] DMA: preallocated 512 KiB GFP_KERNEL|GFP_DMA32 pool for
atomic allocations
[0.234760] audit: initializing netlink subsys (disabled)
[0.237675] audit: type=2000 audit(0.208:1): state=initialized
audit_enabled=0 res=1
[0.242720] hw-breakpoint: found 6 breakpoint and 4 watchpoint
registers.
[0.244348] ASID allocator initialised with 65536 entries
```

```
[0.246324] Serial: AMBA PL011 UART driver
[0.332645] HugeTLB: registered 1.00 GiB page size, pre-allocated 0
pages
[0.333507] HugeTLB: 0 KiB vmemmap can be freed for a 1.00 GiB page
[0.334049] HugeTLB: registered 32.0 MiB page size, pre-allocated 0
pages
[0.334572] HugeTLB: 0 KiB vmemmap can be freed for a 32.0 MiB page
[0.335222] HugeTLB: registered 2.00 MiB page size, pre-allocated 0
pages
[0.335704] HugeTLB: 0 KiB vmemmap can be freed for a 2.00 MiB page
[0.336162] HugeTLB: registered 64.0 KiB page size, pre-allocated 0
pages
[0.336624] HugeTLB: 0 KiB vmemmap can be freed for a 64.0 KiB page
[0.411208] raid6: neonx8 gen() 1655 MB/s
[0.480783] raid6: neonx4 gen() 1314 MB/s
[0.550297] raid6: neonx2 gen() 1861 MB/s
[0.619772] raid6: neonx1 gen() 1249 MB/s
[0.689285] raid6: int64x8 gen() 1098 MB/s
[0.758763] raid6: int64x4 gen() 772 MB/s
[0.828397] raid6: int64x2 gen() 1632 MB/s
[0.898064] raid6: int64x1 gen() 1240 MB/s
[0.898495] raid6: using algorithm neonx2 gen() 1861 MB/s
[0.967595] raid6: xor() 855 MB/s, rmw enabled
[0.968155] raid6: using neon recovery algorithm
[0.973278] iommu: Default domain type: Translated
[0.973667] iommu: DMA domain TLB invalidation policy: strict mode
[0.976748] SCSI subsystem initialized
[0.979443] usbcore: registered new interface driver usbfsl
[0.980438] usbcore: registered new interface driver hub
[0.981028] usbcore: registered new device driver usb
[0.982095] mc: Linux media interface: v0.10
[0.982747] videodev: Linux video capture interface: v2.00
[0.983533] pps_core: LinuxPPS API ver. 1 registered
[0.984098] pps_core: Software ver. 5.3.6 - Copyright 2005-2007
Rodolfo Giometti <giometti@linux.it>
[0.984958] PTP clock support registered
[0.985679] EDAC MC: Ver: 3.0.0
[0.990115] zynqmp-ipi-mbox mailbox@ff9905c0: Registered ZynqMP IPI
mbox with TX/RX channels.
[0.993802] FPGA manager framework
[0.995920] Advanced Linux Sound Architecture Driver Initialized.
[1.007658] Bluetooth: Core ver 2.22
[1.008414] NET: Registered PF_BLUETOOTH protocol family
[1.008978] Bluetooth: HCI device and connection manager initialized
[1.009752] Bluetooth: HCI socket layer initialized
[1.010251] Bluetooth: L2CAP socket layer initialized
[1.010955] Bluetooth: SCO socket layer initialized
[1.020042] clocksource: Switched to clocksource arch_sys_counter
[1.022978] VFS: Disk quotas dquot_6.6.0
[1.023601] VFS: Dquot-cache hash table entries: 512 (order 0, 4096
bytes)
[1.064549] NET: Registered PF_INET protocol family
[1.067305] IP idents hash table entries: 65536 (order: 7, 524288
bytes, linear)
[1.079666] tcp_listen_portaddr_hash hash table entries: 2048 (order:
3, 32768 bytes, linear)
[1.080622] Table-perturb hash table entries: 65536 (order: 6, 262144
bytes, linear)
[1.081380] TCP established hash table entries: 32768 (order: 6,
262144 bytes, linear)
[1.082720] TCP bind hash table entries: 32768 (order: 8, 1048576
bytes, linear)
[1.085108] TCP: Hash tables configured (established 32768 bind 32768)
```

```
[1.087489] UDP hash table entries: 2048 (order: 4, 65536 bytes, linear)
[1.088402] UDP-Lite hash table entries: 2048 (order: 4, 65536 bytes, linear)
[1.090863] NET: Registered PF_UNIX/PF_LOCAL protocol family
[1.096359] RPC: Registered named UNIX socket transport module.
[1.096918] RPC: Registered udp transport module.
[1.097249] RPC: Registered tcp transport module.
[1.097627] RPC: Registered tcp NFSv4.1 backchannel transport module.
[1.098251] PCI: CLS 0 bytes, default 64
[1.105316] Trying to unpack rootfs image as initramfs...
[1.111353] hw perfevents: enabled with armv8_pmuv3 PMU driver, 7 counters available
[1.119352] Initialise system trusted keyrings
[1.122406] workingset: timestamp_bits=46 max_order=20 bucket_order=0
[1.133246] NFS: Registering the id_resolver key type
[1.134036] Key type id_resolver registered
[1.134363] Key type id_legacy registered
[1.135143] nfs4filelayout_init: NFSv4 File Layout Driver
Registering...
[1.135756] nfs4flexfilelayout_init: NFSv4 Flexfile Layout Driver
Registering...
[1.139631] jffs2: version 2.2. (NAND) (SUMMARY) © 2001-2006 Red Hat, Inc.
[1.170288] NET: Registered PF_ALG protocol family
[1.171007] xor: measuring software checksum speed
[1.176041] 8regs : 2468 MB/sec
[1.181098] 32regs : 2213 MB/sec
[1.187480] arm64_neon : 1746 MB/sec
[1.187983] xor: using function: 8regs (2468 MB/sec)
[1.188371] Key type asymmetric registered
[1.188782] Asymmetric key parser 'x509' registered
[1.189565] Block layer SCSI generic (bsg) driver version 0.4 loaded
(major 244)
[1.190250] io scheduler mq-deadline registered
[1.190672] io scheduler kyber registered
[1.213890] irq-xilinx: mismatch in kind-of-intr param
[1.214356] irq-xilinx: /amba_pl@0/interrupt-controller@80020000: num_irq=32, sw_irq=0, edge=0x1
[1.424504] Serial: 8250/16550 driver, 4 ports, IRQ sharing disabled
[1.433575] Serial: AMBA driver
[1.486173] brd: module loaded
[1.509164] loop: module loaded
[1.532894] tun: Universal TUN/TAP device driver, 1.6
[1.534075] CAN device driver interface
[1.538003] usbcore: registered new interface driver asix
[1.538665] usbcore: registered new interface driver ax88179_178a
[1.539237] usbcore: registered new interface driver cdc_ether
[1.539969] usbcore: registered new interface driver net1080
[1.540544] usbcore: registered new interface driver cdc_subset
[1.541100] usbcore: registered new interface driver zaurus
[1.541701] usbcore: registered new interface driver cdc_ncm
[1.542243] usbcore: registered new interface driver r8153_ecm
[1.544431] VFIO - User Level meta-driver version: 0.3
[1.549653] usbcore: registered new interface driver uas
[1.550380] usbcore: registered new interface driver usb-storage
[1.563044] rtc_zynqmp ffa60000 rtc: registered as rtc0
[1.564416] rtc_zynqmp ffa60000 rtc: setting system clock to
2023-08-19T12:42:25 UTC (1692448945)
[1.565842] i2c_dev: i2c /dev entries driver
[1.581536] usbcore: registered new interface driver uvcvideo
[1.590145] Bluetooth: HCI UART driver ver 2.3
[1.590679] Bluetooth: HCI UART protocol H4 registered
```

```
[1.591165] Bluetooth: HCI UART protocol BCSP registered
[1.591673] Bluetooth: HCI UART protocol LL registered
[1.592369] Bluetooth: HCI UART protocol ATH3K registered
[1.592853] Bluetooth: HCI UART protocol Three-wire (H5) registered
[1.593695] Bluetooth: HCI UART protocol Intel registered
[1.594128] Bluetooth: HCI UART protocol QCA registered
[1.594649] usbcore: registered new interface driver bcm203x
[1.595197] usbcore: registered new interface driver bpa10x
[1.595733] usbcore: registered new interface driver bfusb
[1.596468] usbcore: registered new interface driver btusb
[1.597219] usbcore: registered new interface driver ath3k
[1.598663] EDAC MC: ECC not enabled
[1.601384] EDAC DEVICE0: Giving out device to module edac controller
cache_err: DEV edac (POLLED)
[1.602107] cortex_edac edac: cortex 11/12 driver is deprecated
[1.603716] EDAC ZynqMP-OCM: ECC not enabled - Disabling EDAC driver
[1.612454] sdhci: Secure Digital Host Controller Interface driver
[1.612921] sdhci: Copyright(c) Pierre Ossman
[1.613223] sdhci-pltfm: SDHCI platform and OF driver helper
[1.618713] ledtrig-cpu: registered to indicate activity on CPUs
[1.620305] SMCCC: SOC_ID: ID = jep106:0049:0003 Revision = 0x04600093
[1.622634] zynqmp_firmware_probe Platform Management API v1.1
[1.625662] zynqmp_firmware_probe Trustzone version v1.0
[2.195271] securefw securefw: securefw probed
[2.199090] zynqmp-aes zynqmp-aes.0: will run requests pump with
realtime priority
[2.206805] usbcore: registered new interface driver usbhid
[2.207218] usbhid: USB HID core driver
[2.239076] xilinx-ams ffa50000.ams: error -ETIMEDOUT: failed to
initialize AMS
[2.240073] xilinx-ams: probe of ffa50000.ams failed with error -110
[2.249898] ARM CCI_400_r1 PMU driver probed
[2.259326] fpga_manager fpga0: Xilinx ZynqMP FPGA Manager registered
[2.264340] usbcore: registered new interface driver snd-usb-audio
[2.284207] pktgen: Packet Generator for packet performance testing.
Version: 2.75
[2.311267] Initializing XFRM netlink socket
[2.313041] NET: Registered PF_INET6 protocol family
[2.324847] Segment Routing with IPv6
[2.325557] In-situ OAM (IOAM) with IPv6
[2.327221] sit: IPv6, IPv4 and MPLS over IPv4 tunneling driver
[2.332013] NET: Registered PF_PACKET protocol family
[2.332597] NET: Registered PF_KEY protocol family
[2.333710] can: controller area network core
[2.334545] NET: Registered PF_CAN protocol family
[2.334957] can: raw protocol
[2.335425] can: broadcast manager protocol
[2.336184] can: netlink gateway - max_hops=1
[2.337633] Bluetooth: RFCOMM TTY layer initialized
[2.340595] Bluetooth: RFCOMM socket layer initialized
[2.341059] Bluetooth: RFCOMM ver 1.11
[2.341450] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
[2.341838] Bluetooth: BNEP filters: protocol multicast
[2.342255] Bluetooth: BNEP socket layer initialized
[2.342648] Bluetooth: HIDP (Human Interface Emulation) ver 1.2
[2.343115] Bluetooth: HIDP socket layer initialized
[2.344355] 8021q: 802.1Q VLAN Support v1.8
[2.346054] 9pnet: Installing 9P2000 support
[2.346740] Key type dns_resolver registered
[2.348609] registered taskstats version 1
[2.348985] Loading compiled-in X.509 certificates
[2.394966] Btrfs loaded, crc32c=crc32c-generic, zoned=no, fsverity=no
[2.399108] alg: No test for xilinx-zynqmp-rsa (zynqmp-rsa)
```

```
[4.052483] Freeing initrd memory: 30968K
[4.798153] ff000000.serial: ttyPS0 at MMIO 0xff000000 (irq = 23,
base_baud = 2169921) is a xuartps
[4.801163] printk: console [ttyPS0] enabled
[4.801163] printk: console [ttyPS0] enabled
[4.801964] printk: bootconsole [cdns0] disabled
[4.801964] printk: bootconsole [cdns0] disabled
[4.819448] ff010000.serial: ttyPS1 at MMIO 0xff010000 (irq = 24,
base_baud = 2169921) is a xuartps
[4.822965] off-fpga-region fpga-full: FPGA Region probed
[4.869482] nwl-pcie fd0e0000.pcie: host bridge /axi/pcie@fd0e0000
ranges:
[4.870970] nwl-pcie fd0e0000.pcie: MEM
0x00e0000000..0x00efffffff -> 0x00e0000000
[4.871946] nwl-pcie fd0e0000.pcie: MEM
0x0600000000..0x07ffffffff -> 0x0600000000
[4.876198] nwl-pcie fd0e0000.pcie: Link is UP
[4.885284] nwl-pcie fd0e0000.pcie: PCI host bridge to bus 0000:00
[4.886046] pci_bus 0000:00: root bus resource [bus 00-ff]
[4.886706] pci_bus 0000:00: root bus resource [mem
0xe0000000-0xefffffff]
[4.887119] pci_bus 0000:00: root bus resource [mem
0x6000000000-0x7fffffff pref]
[4.929019] xilinx-zynqmp-dpdma fd4c0000.dma-controller: Xilinx DPDMA
engine is probed
[4.941563] spi-nor spi0.0: found n25q512a, expected m25p80
[4.947245] spi-nor spi0.0: n25q512a (131072 Kbytes)
[4.949803] 3 fixed-partitions partitions found on MTD device spi0.0
[4.950214] Creating 3 MTD partitions on "spi0.0":
[4.950707] 0x000000000000-0x000001e00000 : "qspi-boot"
[4.959961] 0x000001e00000-0x000001e40000 : "qspi-bootenv"
[4.963550] 0x000001e40000-0x000004240000 : "qspi-kernel"
[5.065045] macb ff0e0000.ethernet: Not enabling partial store and
forward
[5.065617] macb ff0e0000.ethernet: GEM doesn't support hardware ptp.
[5.172448] xilinx-axipmon ffa00000.perf-monitor: Probed Xilinx APM
[5.175062] xilinx-axipmon fd0b0000.perf-monitor: Probed Xilinx APM
[5.177044] xilinx-axipmon fd490000.perf-monitor: Probed Xilinx APM
[5.178364] xilinx-axipmon ffa10000.perf-monitor: Probed Xilinx APM
[5.195306] i2c i2c-0: using pinctrl states for GPIO recovery
[5.207024] i2c i2c-0: using generic GPIOs for recovery
[5.222621] pca953x 0-0020: supply vcc not found, using dummy regulator
[5.224723] pca953x 0-0020: using no AI
[5.232647] pca953x 0-0021: supply vcc not found, using dummy regulator
[5.233295] pca953x 0-0021: using no AI
[5.266573] i2c i2c-0: Added multiplexed i2c bus 2
[5.287098] i2c i2c-0: Added multiplexed i2c bus 3
[5.296398] max20751 4-0072: Failed to identify chip capabilities
[5.301561] max20751 4-0073: Failed to identify chip capabilities
[5.302361] i2c i2c-0: Added multiplexed i2c bus 4
[5.303194] i2c i2c-0: Added multiplexed i2c bus 5
[5.303508] pca954x 0-0075: registered 4 multiplexed busses for I2C
mux pca9544
[5.304458] cdns-i2c ff020000.i2c: 400 kHz mmio ff020000 irq 51
[5.320157] i2c i2c-1: using pinctrl states for GPIO recovery
[5.329982] i2c i2c-1: using generic GPIOs for recovery
[5.344510] at24 6-0054: supply vcc not found, using dummy regulator
[5.347976] at24 6-0054: 1024 byte 24c08 EEPROM, writable, 1 bytes/
write
[5.348581] i2c i2c-1: Added multiplexed i2c bus 6
[5.351928] si5341 7-0036: no regulator set, defaulting vdd_sel to
2.5V for out
[5.352377] si5341 7-0036: no regulator set, defaulting vdd_sel to
```

```
2.5V for out
[5.352712] si5341 7-0036: no regulator set, defaulting vdd_sel to
2.5V for out
[5.353093] si5341 7-0036: no regulator set, defaulting vdd_sel to
2.5V for out
[5.353449] si5341 7-0036: no regulator set, defaulting vdd_sel to
2.5V for out
[5.353789] si5341 7-0036: no regulator set, defaulting vdd_sel to
2.5V for out
[5.354123] si5341 7-0036: no regulator set, defaulting vdd_sel to
2.5V for out
[5.354577] si5341 7-0036: no regulator set, defaulting vdd_sel to
2.5V for out
[5.358667] si5341 7-0036: Chip: 5341 Grade: 0 Rev: 0
[5.487671] i2c i2c-1: Added multiplexed i2c bus 7
[5.501271] si570 8-005d: registered, current frequency 300000000 Hz
[5.502009] i2c i2c-1: Added multiplexed i2c bus 8
[5.525341] si570 9-005d: registered, current frequency 148500000 Hz
[5.526033] i2c i2c-1: Added multiplexed i2c bus 9
[5.527591] si5324 10-0069: si5328 probed
[5.618246] si5324 10-0069: si5328 probe successful
[5.619039] i2c i2c-1: Added multiplexed i2c bus 10
[5.620251] i2c i2c-1: Added multiplexed i2c bus 11
[5.621124] i2c i2c-1: Added multiplexed i2c bus 12
[5.621987] i2c i2c-1: Added multiplexed i2c bus 13
[5.622310] pca954x 1-0074: registered 8 multiplexed busses for I2C
switch pca9548
[5.624440] i2c i2c-1: Added multiplexed i2c bus 14
[5.625350] i2c i2c-1: Added multiplexed i2c bus 15
[5.626231] i2c i2c-1: Added multiplexed i2c bus 16
[5.627101] i2c i2c-1: Added multiplexed i2c bus 17
[5.628185] i2c i2c-1: Added multiplexed i2c bus 18
[5.629106] i2c i2c-1: Added multiplexed i2c bus 19
[5.630155] i2c i2c-1: Added multiplexed i2c bus 20
[5.631071] i2c i2c-1: Added multiplexed i2c bus 21
[5.631463] pca954x 1-0075: registered 8 multiplexed busses for I2C
switch pca9548
[5.632202] cdns-i2c ff030000.i2c: 400 kHz mmio ff030000 irq 52
[5.648872] cdns-wdt fd4d0000.watchdog: Xilinx Watchdog Timer with
timeout 60s
[5.650737] cdns-wdt ff150000.watchdog: Xilinx Watchdog Timer with
timeout 10s
[5.659725] cpufreq: cpufreq_online: CPU0: Running at unlisted initial
frequency: 277750 KHz, changing to: 299999 KHz
[5.681771] zynqmp-display fd4a0000.display: vtc bridge property not
present
[6.634386] xilinx-dp-snd-codec fd4a0000.display:zynqmp-dp-snd-codec0:
Xilinx DisplayPort Sound Codec probed
[6.637878] xilinx-dp-snd-pcm zynqmp_dp_snd_pcm0: Xilinx DisplayPort
Sound PCM probed
[6.639670] xilinx-dp-snd-pcm zynqmp_dp_snd_pcm1: Xilinx DisplayPort
Sound PCM probed
[6.661467] xilinx-dp-snd-card fd4a0000.display:zynqmp-dp-snd-card:
Xilinx DisplayPort Sound Card probed
[6.663424] OF: graph: no port node found in /axi/display@fd4a0000
[6.675090] xlnx-drm xlnx-drm.0: bound fd4a0000.display (ops
0xfffff800008f3a090)
[6.685196] mmc0: SDHCI controller on ff170000.mmc [ff170000.mmc]
using ADMA 64-bit
[6.734924] mmc0: new high speed SD card at address 4567
[6.740085] mmcblk0: mmc0:4567 QEMU! 256 MiB
[6.904780] Console: switching to colour frame buffer device 160x50
[6.935661] zynqmp-display fd4a0000.display: [drm] fb0: xlnxdrmfb
```

```
frame buffer device
[6.939737] [drm] Initialized xlnx 1.0.0 20130509 for fd4a0000.display
on minor 0
[6.941097] zynqmp-display fd4a0000.display: ZynqMP DisplayPort
Subsystem driver probed
[6.944824] ahci-ceva fd0c0000.ahci: supply ahci not found, using
dummy regulator
[6.945921] ahci-ceva fd0c0000.ahci: supply phy not found, using dummy
regulator
[6.947070] ahci-ceva fd0c0000.ahci: supply target not found, using
dummy regulator
[6.954579] ahci-ceva fd0c0000.ahci: AHCI 0001.0000 32 slots 2 ports
1.5 Gbps 0x3 impl platform mode
[6.955398] ahci-ceva fd0c0000.ahci: flags: 64bit ncq only
[6.964374] scsi host0: ahci-ceva
[6.968050] scsi host1: ahci-ceva
[6.969093] ata1: SATA max UDMA/133 mmio [mem 0xfd0c0000-0xfd0c1fff]
port 0x100 irq 57
[6.969677] ata2: SATA max UDMA/133 mmio [mem 0xfd0c0000-0xfd0c1fff]
port 0x180 irq 57
[7.040211] macb ff0e0000.ethernet: Not enabling partial store and
forward
[7.040742] macb ff0e0000.ethernet: GEM doesn't support hardware ptp.
[7.127704] macb ff0e0000.ethernet eth0: Cadence GEM rev 0x40070106 at
0xff0e0000 irq 49 (6e:69:71:f1:31:76)
[7.289948] ata2: SATA link down (SStatus 0 SControl 300)
[7.291599] dwc3 fe200000.usb: UDC doesn't support Gen 1
[7.295237] ata1: SATA link down (SStatus 0 SControl 300)
[7.296318] xhci-hcd xhci-hcd.1.auto: xHCI Host Controller
[7.297008] xhci-hcd xhci-hcd.1.auto: new USB bus registered, assigned
bus number 1
[7.302074] xhci-hcd xhci-hcd.1.auto: hcc params 0x00087001 hci
version 0x100 quirks 0x0000000000010810
[7.303008] xhci-hcd xhci-hcd.1.auto: irq 58, io mem 0xfe200000
[7.305056] xhci-hcd xhci-hcd.1.auto: xHCI Host Controller
[7.305467] xhci-hcd xhci-hcd.1.auto: new USB bus registered, assigned
bus number 2
[7.306112] xhci-hcd xhci-hcd.1.auto: Host supports USB 3.0 SuperSpeed
[7.312087] usb usb1: New USB device found, idVendor=1d6b,
idProduct=0002, bcdDevice= 6.01
[7.312623] usb usb1: New USB device strings: Mfr=3, Product=2,
SerialNumber=1
[7.313101] usb usb1: Product: xHCI Host Controller
[7.313463] usb usb1: Manufacturer: Linux 6.1.30-xilinx-v2023.2 xhci-
hcd
[7.313941] usb usb1: SerialNumber: xhci-hcd.1.auto
[7.319217] hub 1-0:1.0: USB hub found
[7.320450] hub 1-0:1.0: 4 ports detected
[7.325580] usb usb2: We don't know the algorithms for LPM for this
host, disabling LPM.
[7.326523] usb usb2: New USB device found, idVendor=1d6b,
idProduct=0003, bcdDevice= 6.01
[7.327001] usb usb2: New USB device strings: Mfr=3, Product=2,
SerialNumber=1
[7.327400] usb usb2: Product: xHCI Host Controller
[7.327667] usb usb2: Manufacturer: Linux 6.1.30-xilinx-v2023.2 xhci-
hcd
[7.328242] usb usb2: SerialNumber: xhci-hcd.1.auto
[7.329710] hub 2-0:1.0: USB hub found
[7.330229] hub 2-0:1.0: 4 ports detected
[7.347059] input: gpio-keys as /devices/platform/gpio-keys/input/
input0
[7.351488] of_cfs_init
```

```
[7.352385] of_cfs_init: OK
[7.353520] clk: Not disabling unused clocks
[7.423724] ALSA device list:
[7.424807] #0: DP mon
[7.471719] Freeing unused kernel memory: 2240K
[7.474208] Run /init as init process
[7.635608] systemd[1]: systemd 251.8+ running in system mode (+PAM -
AUDIT -SELINUX -APPARMOR +IMA -SMACK +SECCOMP -GCRYPT -GNUTLS -OPENSSL
+ACL +BLKID -CURL -ELFUTILS -FIDO2 -IDN2 -IDN -IPTC +KMOD -LIBCRYPTSETUP
+LIBFDISK -PCRE2 -PWQUALITY -P11KIT -QRENCODE -TPM2 -BZIP2 -LZ4 -XZ -ZLIB
+ZSTD -BPF_FRAMEWORK +XKBCOMMON +UTMP +SYSVINIT default-hierarchy=hybrid)
[7.638790] systemd[1]: Detected architecture arm64.

Welcome to PetaLinux 2023.2+release-S08141938 (langdale)!

[7.736543] systemd[1]: Hostname set to <xilinx-zcu102-20232>.
[7.738764] systemd[1]: Initializing machine ID from random generator.
[8.159552] systemd-sysv-generator[185]: SysV service '/etc/init.d/
sshd' lacks a native systemd unit file. Automatically generating a unit
file for compatibility. Please update package to include a native systemd
unit file, in order to make it more safe and robust.
[8.166823] systemd-sysv-generator[185]: SysV service '/etc/init.d/
nfsserver' lacks a native systemd unit file. Automatically generating a
unit file for compatibility. Please update package to include a native
systemd unit file, in order to make it more safe and robust.
[8.170675] systemd-sysv-generator[185]: SysV service '/etc/init.d/
inetd.busybox' lacks a native systemd unit file. Automatically generating
a unit file for compatibility. Please update package to include a native
systemd unit file, in order to make it more safe and robust.
[8.982149] systemd[1]: Queued start job for default target Multi-User
System.
[9.178108] systemd[1]: Created slice Slice /system/getty.
[OK] Created slice Slice /system/getty.
[9.190103] systemd[1]: Created slice Slice /system/modprobe.
[OK] Created slice Slice /system/modprobe.
[9.198226] systemd[1]: Created slice Slice /system/serial-getty.
[OK] Created slice Slice /system/serial-getty.
[9.205636] systemd[1]: Created slice User and Session Slice.
[OK] Created slice User and Session Slice.
[9.210874] systemd[1]: Started Dispatch Password Requests to Console
Directory Watch.
[OK] Started Dispatch Password ...ts to Console Directory Watch.
[9.216037] systemd[1]: Started Forward Password Requests to Wall
Directory Watch.
[OK] Started Forward Password R...uests to Wall Directory Watch.
[9.218970] systemd[1]: Reached target Path Units.
[OK] Reached target Path Units.
[9.221296] systemd[1]: Reached target Remote File Systems.
[OK] Reached target Remote File Systems.
[9.223499] systemd[1]: Reached target Slice Units.
[OK] Reached target Slice Units.
[9.225641] systemd[1]: Reached target Swaps.
[OK] Reached target Swaps.
[9.238290] systemd[1]: Listening on RPCbind Server Activation Socket.
[OK] Listening on RPCbind Server Activation Socket.
[9.240892] systemd[1]: Reached target RPC Port Mapper.
[OK] Reached target RPC Port Mapper.
[9.245349] systemd[1]: Listening on Syslog Socket.
[OK] Listening on Syslog Socket.
[9.249116] systemd[1]: Listening on initctl Compatibility Named Pipe.
[OK] Listening on initctl Compatibility Named Pipe.
[9.254887] systemd[1]: Listening on Journal Audit Socket.
[OK] Listening on Journal Audit Socket.
```

```
[9.259024] systemd[1]: Listening on Journal Socket (/dev/log).
[OK] Listening on Journal Socket (/dev/log).
[9.266998] systemd[1]: Listening on Journal Socket.
[OK] Listening on Journal Socket.
[9.273025] systemd[1]: Listening on Network Service Netlink Socket.
[OK] Listening on Network Service Netlink Socket.
[9.277744] systemd[1]: Listening on udev Control Socket.
[OK] Listening on udev Control Socket.
[9.281567] systemd[1]: Listening on udev Kernel Socket.
[OK] Listening on udev Kernel Socket.
[9.285254] systemd[1]: Listening on User Database Manager Socket.
[OK] Listening on User Database Manager Socket.
[9.331489] systemd[1]: Mounting Huge Pages File System...
 Mounting Huge Pages File System...
[9.351520] systemd[1]: Mounting POSIX Message Queue File System...
 Mounting POSIX Message Queue File System...
[9.373197] systemd[1]: Mounting Kernel Debug File System...
 Mounting Kernel Debug File System...
[9.378266] systemd[1]: Kernel Trace File System was skipped because
of a failed condition check (ConditionPathExists=/sys/kernel/tracing).
[9.397870] systemd[1]: Mounting Temporary Directory /tmp...
 Mounting Temporary Directory /tmp...
[9.402347] systemd[1]: Create List of Static Device Nodes was skipped
because of a failed condition check (ConditionFileNotEmpty=/lib/modules/
6.1.30-xilinx-v2023.2/modules.devname).
[9.429906] systemd[1]: Starting Load Kernel Module configfs...
 Starting Load Kernel Module configfs...
[9.452630] systemd[1]: Starting Load Kernel Module drm...
 Starting Load Kernel Module drm...
[9.474004] systemd[1]: Starting Load Kernel Module fuse...
 Starting Load Kernel Module fuse...
[9.502041] systemd[1]: Starting RPC Bind...
 Starting RPC Bind...
[9.505706] systemd[1]: File System Check on Root Device was skipped
because of a failed condition check (ConditionPathIsReadWrite=!/).
[9.510879] systemd[1]: systemd-journald.service: unit configures an
IP firewall, but the local system does not support BPF/cgroup firewalling.
[9.511628] systemd[1]: (This warning is only shown for the first unit
using IP firewalling.)
[9.552413] systemd[1]: Starting Journal Service...
 Starting Journal Service...
[9.559363] systemd[1]: Load Kernel Modules was skipped because all
trigger condition checks failed.
[9.584785] systemd[1]: Mounting NFSD configuration filesystem...
 Mounting NFSD configuration filesystem...
[9.606427] systemd[1]: Starting Generate network units from Kernel
command line...
 Starting Generate network ...ts from Kernel command line...
[9.643283] systemd[1]: Starting Remount Root and Kernel File
Systems...
 Starting Remount Root and Kernel File Systems...
[9.680967] systemd[1]: Starting Apply Kernel Variables...
 Starting Apply Kernel Variables...
[9.700675] systemd[1]: Starting Coldplug All udev Devices...
 Starting Coldplug All udev Devices...
[9.745122] systemd[1]: Started RPC Bind.
[OK] Started RPC Bind.
[9.765479] systemd[1]: Mounted Huge Pages File System.
[OK] Mounted Huge Pages File System.
[9.776750] systemd[1]: Mounted POSIX Message Queue File System.
[OK] Mounted POSIX Message Queue File System.
[9.784300] systemd[1]: Mounted Kernel Debug File System.
[OK] Mounted Kernel Debug File System.
```

```
[9.789661] systemd[1]: Mounted Temporary Directory /tmp.
[OK] Mounted Temporary Directory /tmp.
[9.801150] systemd[1]: modprobe@configfs.service: Deactivated
successfully.
[9.806965] systemd[1]: Finished Load Kernel Module configfs.
[OK] Finished Load Kernel Module configfs.
[9.817536] systemd[1]: modprobe@drm.service: Deactivated successfully.
[9.821098] systemd[1]: Finished Load Kernel Module drm.
[OK] Finished Load Kernel Module drm.
[9.830387] systemd[1]: modprobe@fuse.service: Deactivated
successfully.
[9.837722] systemd[1]: Finished Load Kernel Module fuse.
[OK] Finished Load Kernel Module fuse.
[9.841802] systemd[1]: Mounted NFSD configuration filesystem.
[OK] Mounted NFSD configuration filesystem.
[9.851077] systemd[1]: Finished Generate network units from Kernel
command line.
[OK] Finished Generate network units from Kernel command line.
[9.861570] systemd[1]: Reached target Preparation for Network.
[OK] Reached target Preparation for Network.
[9.866686] systemd[1]: FUSE Control File System was skipped because
of a failed condition check (ConditionPathExists=/sys/fs/fuse/
connections).
[9.922211] systemd[1]: Mounting Kernel Configuration File System...
Mounting Kernel Configuration File System...
[9.988317] systemd[1]: Finished Remount Root and Kernel File Systems.
[OK] Finished Remount Root and Kernel File Systems.
[9.996580] systemd[1]: Finished Apply Kernel Variables.
[OK] Finished Apply Kernel Variables.
[10.001009] systemd[1]: Rebuild Hardware Database was skipped because
all trigger condition checks failed.
[10.002340] systemd[1]: Platform Persistent Storage Archival was
skipped because of a failed condition check
(ConditionDirectoryNotEmpty=/sys/fs/pstore).
[10.046662] systemd[1]: Starting Create System Users...
Starting Create System Users...
[10.065445] systemd[1]: Mounted Kernel Configuration File System.
[OK] Mounted Kernel Configuration File System.
[10.215115] systemd[1]: Finished Create System Users.
[OK] Finished Create System Users.
[10.250754] systemd[1]: Starting Create Static Device Nodes in /dev...
Starting Create Static Device Nodes in /dev...
[10.383563] systemd[1]: Finished Create Static Device Nodes in /dev.
[OK] Finished Create Static Device Nodes in /dev.
[10.387570] systemd[1]: Reached target Preparation for Local File
Systems.
[OK] Reached target Preparation for Local File Systems.
[10.427049] systemd[1]: Starting Rule-based Manager for Device Events
and Files...
Starting Rule-based Manager for Device Events and Files...
[10.433331] systemd[1]: Started Journal Service.
[OK] Started Journal Service.
Starting Flush Journal to Persistent Storage...
[10.586744] systemd-journald[195]: Received client request to flush
runtime journal.
[OK] Finished Flush Journal to Persistent Storage.
Mounting /var/volatile...
[OK] Started Rule-based Manager for Device Events and Files.
Starting Network Configuration...
[OK] Mounted /var/volatile.
Starting Load/Save Random Seed...
[OK] Reached target Local File Systems.
Starting Rebuild Dynamic Linker Cache...
```

```
Starting Create Volatile Files and Directories...
[OK] Finished Rebuild Dynamic Linker Cache.
[OK] Finished Create Volatile Files and Directories.
 Starting Run pending postinsts...
 Starting Rebuild Journal Catalog...
[11.520354] cfg80211: Loading compiled-in X.509 certificates for
regulatory database
 Starting Network Name Resolution...
 Starting Network Time Synchronization...
 Starting Record System Boot/Shutdown in UTMP...
[11.830246] cfg80211: Loaded X.509 cert 'sforshee: 00b28ddf47aef9cea7'
[11.833700] platform regulatory.0: Direct firmware load for
regulatory.db failed with error -2
[11.834540] cfg80211: failed to load regulatory.db
[OK] Finished Rebuild Journal Catalog.
[11.994016] Unloading old XRT Linux kernel modules
 Starting Update is Completed...
[12.084679] Loading new XRT Linux kernel modules
[12.133474] zocl: loading out-of-tree module taints kernel.
[12.164597] [drm] Probing for xlnx,zocl
[12.168657] zocl-drm amba_pl@0:zyxclmm_drm: error -ENXIO: IRQ index 32
not found
[12.197790] [drm] FPGA programming device pcap founded.
[OK] Finished Record System Boot/Shutdown in UTMP.
[12.201987] [drm] PR[0] Isolation addr 0x0
[12.252049] [drm] Initialized zocl 2.15.0 20230819 for
amba_pl@0:zyxclmm_drm on minor 1
[OK] Finished Update is Completed.
[OK] Started Network Configuration.
[12.647732] INFO: Creating ICD entry for Xilinx Platform
[12.756343] random: crng init done
[OK] Finished Load/Save Random Seed.
[OK] Started Network Time Synchronization.
[OK] Started Network Name Resolution.
[OK] Reached target Network.
[OK] Reached target Host and Network Name Lookups.
[OK] Reached target System Time Set.
[OK] Started NFS status monitor for NFSv2/3 locking..
[OK] Finished Run pending postinsts.
[OK] Finished Coldplug All udev Devices.
[OK] Reached target System Initialization.
[OK] Started Daily Cleanup of Temporary Directories.
[OK] Reached target Timer Units.
[OK] Listening on D-Bus System Message Bus Socket.
 Starting sshd.socket...
[OK] Listening on sshd.socket.
[OK] Reached target Socket Units.
[OK] Reached target Basic System.
[OK] Started Kernel Logging Service.
[OK] Started System Logging Service.
 Starting D-Bus System Message Bus...
 Starting inetc.busybox.service...
 Starting LSB: Kernel NFS server support...
 Starting User Login Management...
 Starting Permit User Sessions...
 Starting Target Communication Framework agent...
 Starting OpenSSH Key Generation...
[OK] Started D-Bus System Message Bus.
[OK] Started inetc.busybox.service.
[OK] Finished Permit User Sessions.
[OK] Started Target Communication Framework agent.
[21.018992] macb ff0e0000.ethernet eth0: PHY [ff0e0000.ethernet-
ffffffffff:0c] driver [TI DP83867] (irq=_POLL)
```

```
[21.020606] macb ff0e0000.ethernet eth0: configuring for phy/rgmii-id
link mode
[21.029352] macb ff0e0000.ethernet eth0: unable to generate target
frequency: 125000000 Hz
[21.048562] macb ff0e0000.ethernet eth0: Link is Up - 1Gbps/Full -
flow control tx
[21.076591] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[21.471486] NFSD: Using /var/lib/nfs/v4recovery as the NFSv4 state
recovery directory
[21.472714] NFSD: Using legacy client tracking operations.
[21.472815] NFSD: starting 90-second grace period (net f0000000)
[FAILED] Failed to start LSB: Kernel NFS server support.
See 'systemctl status nfsserver.service' for details.
[OK] Started User Login Management.
[OK] Created slice Slice /system/systemd-fsck.
[OK] Reached target Sound Card.
[OK] Listening on Load/Save RF ...tch Status /dev/rfkill Watch.
[OK] Started Getty on tty1.
[OK] Started Serial Getty on ttys0.
[OK] Reached target Login Prompts.
[OK] Reached target Multi-User System.
Starting File System Check on /dev/mmcblk0...
Starting Record Runlevel Change in UTMP...
[OK] Finished Record Runlevel Change in UTMP.
[OK] Finished File System Check on /dev/mmcblk0.
Mounting /run/media/mmcblk0...
[27.970693] EXT4-fs (mmcblk0): mounted filesystem with ordered data
mode. Quota mode: none.
[OK] Mounted /run/media/mmcblk0.
[OK] Finished OpenSSH Key Generation.

PetaLinux 2023.2+release-S08141938 xilinx-zcu102-20232 ttys0
xilinx-zcu102-20232 login:
```

- Specify this option to disable GDB through QEMU boot

---

## petalinux-package

The `petalinux-package` tool packages a PetaLinux project into a format suitable for deployment. The tool provides several workflows whose operations vary depending on the target package format. The supported formats/workflows are `boot`, `bsp`, and `pre-built`. The `petalinux-package` tool is executed using the package type name to specify a specific workflow in the format `petalinux-package --PACKAGETYPE`.

- The `boot` package type creates a file (.BIN or .MCS) that allows the target device to boot.
- The `bsp` package type creates a .bsp file which includes the entire contents of the target PetaLinux project. This option allows you to export and re-use your bsp.
- The `pre-built` package type creates a new directory within the target PetaLinux project called "pre-built" and contains prebuilt content that is useful for booting directly on a physical board. This package type is commonly used as a precursor for creating a `bsp` package type.
- The `image` package type packages image for component with the specified format.

- The `sysroot` package type installs the sysroot for the AMD Vitis™ software platform. It can specify the installer path and also install directory path.

You are required to install AMD Vivado™ Design Suite on the same machine as PetaLinux to use `petalinux-boot` for the MCS format for MicroBlaze™ processor. By default, the `petalinux-package` tool loads default files from the `<plnx-proj-root>/images/linux/` directory.

## **petalinux-package Command Line Options**

The following table details the command line options that are common to all of the `petalinux-package` workflows.

**Table 52: petalinux-package Command Line Options**

| Option                             | Functional Description                              | Value Range    | Default Value     |
|------------------------------------|-----------------------------------------------------|----------------|-------------------|
| <code>-p, --project PROJECT</code> | PetaLinux project directory path. This is optional. | User-specified | Current Directory |
| <code>-h, --help</code>            | Display usage information. This is optional.        | None           | None              |

## **petalinux-package --boot**

The `petalinux-package --boot` command generates a bootable image that can be used directly with AMD Versal™ adaptive SoC, AMD Zynq™ UltraScale+™ MPSoC and Zynq 7000 devices, and also with MicroBlaze™-based FPGA designs. For devices in the Zynq series, bootable format is `BOOT.BIN` which can be booted from an SD card. For MicroBlaze-based designs, the default format is an `MCS` PROM file suitable for programming using AMD Vivado™ Design Suite or other PROM programmer.

For devices in the Zynq series, this workflow is a wrapper around the `bootgen` utility provided with the Vitis software platform. For MicroBlaze-based FPGA designs, this workflow is a wrapper around the corresponding Vivado Tcl commands and generates an `MCS` formatted programming file. This `MCS` file can be programmed directly to a target board and booted.

## **petalinux-package --boot Command Options**

The following table details the options that are valid when creating a bootable image with the `petalinux-package --boot` command:

**Table 53: petalinux-package --boot Command Options**

| Option                       | Functional Description                           | Value Range                                                                                    | Default Value |
|------------------------------|--------------------------------------------------|------------------------------------------------------------------------------------------------|---------------|
| <code>--format FORMAT</code> | Image file format to generate. This is optional. | <ul style="list-style-type: none"> <li>• BIN</li> <li>• MCS</li> <li>• DOWNLOAD.BIT</li> </ul> | BIN           |

Table 53: **petalinux-package --boot** Command Options (cont'd)

| Option                         | Functional Description                                                                                                                           | Value Range    | Default Value                                                                                                                                                                                                                                                          |
|--------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| --fsbl FSBL                    | Path on disk to FSBL elf binary. This is required. To skip loading FSBL, use --fsbl no or --fsbl none. This is optional.                         | User-specified | <ul style="list-style-type: none"> <li>• zynqmp_fsb.elf for AMD Zynq™ UltraScale+™ MPSoC</li> <li>• zynq_fsbl.elf for Zynq 7000 device</li> <li>• fs-boot.elf for MicroBlaze™ processor</li> </ul> <p>The default image is in &lt;plnx-proj-root&gt;/images/linux.</p> |
| --force                        | Overwrite existing files on disk. This is optional.                                                                                              | None           | None                                                                                                                                                                                                                                                                   |
| --fpga BITSTREAM <sup>1</sup>  | Path on disk to bitstream file. This is optional. To skip loading BITSTREAM , use --fpga no or -fpga none                                        | User-specified | <plnx-proj-root>/images/linux/system.bit                                                                                                                                                                                                                               |
| --atf TF-A-IMG                 | Path on disk to Arm® trusted firmware elf binary. This is optional. To skip loading TF-A, use --atf no or --atf none                             | User-specified | <plnx-proj-root>/images/linux/bl31.elf                                                                                                                                                                                                                                 |
| --u-boot UBOOT-IMG             | Path on disk to U-Boot binary. This is optional.                                                                                                 | User-specified | <ul style="list-style-type: none"> <li>• u-boot.elf for Zynq device</li> <li>• u-boot-s.bin for MicroBlaze CPUs</li> </ul> <p>The default image is in &lt;plnx-proj-root&gt;/images/linux</p>                                                                          |
| --kernel KERNEL-IMG            | Path on disk to Linux kernel image. This is optional.                                                                                            | User-specified | <plnx-proj-root>/images/linux/image.ub                                                                                                                                                                                                                                 |
| --boot-script BOOT-SCRIPT      | Path to the boot.scr file location. This is optional.                                                                                            | User-specified | <plnx-proj-root>/images/linux/boot.scr                                                                                                                                                                                                                                 |
| --qemu-rootfs ROOTFS-CPIO-FILE | Path to the rootfs file location to create qemu_boot.img (cpio.gz.u-boot). Only valid for the Versal adaptive SoC to generate the QEMU SD image. | User specified | <plnx-proj-root>/images/linux/rootfs.cpio.gz.u-boot                                                                                                                                                                                                                    |

Table 53: **petalinux-package --boot** Command Options (cont'd)

| Option                 | Functional Description                                                                                                                                                                                                                                              | Value Range                                                                                                                                                                                                | Default Value                                                                           |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|
| --pmufw PMUFW-ELF      | Optional and applicable only for AMD Zynq™ UltraScale+™ MPSoC. By default, prebuilt PMU firmware image is packed. Use this option to either specify a path for PMU firmware image or to skip packing of PMU firmware. To skip packing PMU firmware, use --pmufw no. | User-specified                                                                                                                                                                                             | <plnx-proj-root>/images/linux/pmufw.elf                                                 |
| --plm PLM-ELF          | Optional and applicable only for AMD Versal™ adaptive SoC. By default, prebuilt PLM image is packed. Use this option to either specify a path for PLM image or to skip packing of PLM. To skip packing PLM, use --plm no.                                           | User-specified                                                                                                                                                                                             | <plnx-proj-root>/images/linux/plm.elf                                                   |
| --psmfw PSMFW-ELF      | Optional and applicable only for Versal adaptive SoC. By default, prebuilt PSM firmware image is packed. Use this option to either specify a path for PSM firmware image or to skip packing of PSM firmware. To skip packing PSM firmware, use --psmfw no.          | User-specified                                                                                                                                                                                             | <plnx-proj-root>/images/linux/psmfw.elf                                                 |
| --addcdo CDOFILE       | Path on disk to add .cdo file pack into BOOT.BIN.                                                                                                                                                                                                                   | User-specified                                                                                                                                                                                             | None                                                                                    |
| --add DATAFILE         | Path on disk to arbitrary data to include. This is optional.                                                                                                                                                                                                        | User-specified                                                                                                                                                                                             | None                                                                                    |
| --offset OFFSET        | Offset at which to load the prior data file. Only the .elf files are parsed. This is optional.                                                                                                                                                                      | User-specified                                                                                                                                                                                             | None                                                                                    |
| --load <LOADADDR>      | Load address for specified data file. The RAM address where to load the specified data file.<br>Example: [ partition_type=raw, load=0x01000 ] <image>                                                                                                               | User-specified                                                                                                                                                                                             | None                                                                                    |
| --mmi MMIFILE          | Bitstream MMI file, valid for MicroBlaze CPUs only. It is used to generate the download.bit with boot loader in the block RAM. Default is the MMI file in the same directory as the FPGA bitstream. This is optional                                                | User-specified                                                                                                                                                                                             | MMI in directory with FPGA bitstream                                                    |
| --flash-size SIZE      | Flash size in MB. Must be a power-of-2. Valid for MicroBlaze processor only. Not needed for parallel flash types. Ensure you pass digit value to this option. Do not include MB in the value. This is optional.                                                     | User-specified                                                                                                                                                                                             | Auto-detect from system configuration. If it is not specified, the default value is 16. |
| --flash-intf INTERFACE | Valid for MicroBlaze processor only. This is optional.                                                                                                                                                                                                              | <ul style="list-style-type: none"> <li>• SERIALx1</li> <li>• SPIx1</li> <li>• SPIx2</li> <li>• SPIx4</li> <li>• BPix8</li> <li>• BPix16</li> <li>• SMAPx8</li> <li>• SMAPx16</li> <li>• SMAPx32</li> </ul> | Auto-detect                                                                             |

Table 53: **petalinux-package --boot** Command Options (*cont'd*)

| Option                                       | Functional Description                                                                                                                                                                                                                                                                                                                                                                                | Value Range                                                                                                  | Default Value |
|----------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------|---------------|
| <code>-o, --output OUTPUTFILE</code>         | Path on disk to write output image. This is optional.                                                                                                                                                                                                                                                                                                                                                 | User-specified                                                                                               | None          |
| <code>--cpu DESTINATION CPU</code>           | Zynq UltraScale+ MPSoC only. The destination CPU of the previous data file. This is optional.                                                                                                                                                                                                                                                                                                         | <ul style="list-style-type: none"> <li>• a53-0</li> <li>• a53-1</li> <li>• a53-2</li> <li>• a53-3</li> </ul> | None          |
| <code>--file-attribute DATA File ATTR</code> | Zynq 7000, AMD Zynq™ UltraScale+™ MPSoC, and Versal adaptive SoC only. Data file file-attribute. This is optional.<br><br>Example:<br><code>petalinux-package --boot --u-boot<br/>--kernel images/linux/Image --<br/>offset 0x01e40000 --file-<br/>attribute partition_owner=uboot<br/>--add images/linux/system.dtb --<br/>offset 0x3AD1200 --file-attribute<br/>partition_owner=uboot --fpga</code> | User-specified                                                                                               | None          |
| <code>--bif-attribute ATTRIBUTE</code>       | Zynq 7000, AMD Zynq™ UltraScale+™ MPSoC, and Versal adaptive SoC only.<br><br>Example:<br><code>petalinux-package --boot --bif-<br/>attribute fsbl_config --bif-<br/>attribute-value a53_x64 --u-boot</code>                                                                                                                                                                                          | User-specified                                                                                               | None          |
| <code>--bif-attribute-value VALUE</code>     | Zynq 7000, AMD Zynq™ UltraScale+™ MPSoC, and Versal adaptive SoC only. The value of the attribute specified by <code>--bif-attribute</code> argument. This is optional.<br><br>Example:<br><code>petalinux-package --boot --bif-<br/>attribute fsbl_config --bif-<br/>attribute-value a53_x64 --u-boot</code>                                                                                         | User-specified                                                                                               | None          |
| <code>--fsblconfig BIF_FSBL_CONFIG</code>    | AMD Zynq™ UltraScale+™ MPSoC only. BIF FSBL config value.<br><br>Example:<br><code>petalinux-package --boot --<br/>fsblconfig a53_x64 --u-boot</code>                                                                                                                                                                                                                                                 | User-specified                                                                                               | None          |

Table 53: **petalinux-package --boot** Command Options (cont'd)

| Option                    | Functional Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | Value Range                                                          | Default Value                                                                         |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| --bif BIF FILE            | Zynq 7000 devices, Zynq UltraScale+ MPSoC, and Versal adaptive SoC. BIF file. For Zynq 7000 devices and Zynq UltraScale + MPSoC, it overrides the following settings: <ul style="list-style-type: none"><li>• -fsbl</li><li>• -fpga</li><li>• -u-boot</li><li>• -add</li><li>• -fsblconfig</li><li>• -file-attribute</li><li>• -bif-attribute</li><li>• -bif-attribute-value</li></ul> For Versal adaptive SoC, it overrides the following settings: <ul style="list-style-type: none"><li>• -fpga</li><li>• -u-boot</li><li>• -add</li><li>• -file-attribute</li><li>• -bif-attribute</li><li>• -bit-attribute-value</li></ul> This is optional. | User-specified                                                       | None                                                                                  |
| --boot-device BOOT-DEV    | Zynq 7000, Zynq UltraScale+ MPSoC, and Versal adaptive SoC. The boot device is updated in bootargs to boot. This is optional.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | <ul style="list-style-type: none"><li>• sd</li><li>• flash</li></ul> | Default value is the one selected from the system select menu of boot image settings. |
| --bootgen-extra-args ARGS | Zynq 7000, Zynq UltraScale+ MPSoC, and Versal adaptive SoC only. Extra arguments to be passed while invoking bootgen command. This is optional.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | User-specified                                                       | None                                                                                  |

**Notes:**

- When the **FPGA manager** option is enabled in `petalinux-config`, the `--fpga` option cannot be used. Bitstream is not included in the `BOOT.BIN`.

## ***petalinux-package --boot Examples***

The following examples demonstrate the correct usage of the `petalinux-package --boot` command.

- Create a `BOOT.BIN` file for an AMD Versal™ device.

```
petalinux-package --boot --format BIN --plm --psmfw --u-boot --dtb -o
<PATH-TO-OUTPUT-WITH-FILE-NAME>
```

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

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

It generates a `BOOT.BIN` with a specified load address for DTB.

**Note:** The files `versal-qemu-multiarch-pmc.dtb` and `versal-qemu-multiarch-ps.dtb` are QEMU DTBs required to boot multi-arch QEMU. Use `system.dtb` for `--dtb` option, generated in `image/linux/` directory or simply use `--dtb` option.

- Create a `BOOT.BIN` file for an AMD Zynq™ device (including Zynq 7000 and AMD Zynq™ UltraScale+™ MPSoC).

```
petalinux-package --boot --format BIN --fsbl <PATH-TO-FSBL> --u-boot -o <PATH-TO-OUTPUT-WITH-FILE-NAME>
```

- Create a `BOOT.BIN` file for a Zynq device that includes a PL bitstream and FITImage.

```
petalinux-package --boot --format BIN --fsbl <PATH-TO-FSBL> --u-boot --fpga <PATH-TO-BITSTREAM> --kernel -o <PATH-TO-OUTPUT>
```

- Create a x8 SMAP PROM MCS file for a MicroBlaze™ CPU design.

```
petalinux-package --boot --format MCS --fsbl <PATH-TO-FSBL> --u-boot --fpga <PATH-TO-BITSTREAM> --flash-size <SIZE> --flash-intf SMAPx8 -o <PATH-TO-OUTPUT-WITH-FILE-NAME>
```

- Create a `BOOT.BIN` file for a Zynq UltraScale+ MPSoC that includes PMU firmware.

```
petalinux-package --boot --u-boot --kernel --pmufw <PATH_TO_PMUFW>
```

- Create bitstream file `download.bit` for a MicroBlaze CPU design.

```
petalinux-package --boot --format DOWNLOAD.BIT --fpga <BITSTREAM> --fsbl <FSBOOT_ELF>
```

## **`petalinux-package --bsp`**

The `petalinux-package --bsp` command compiles all contents of the specified PetaLinux project directory into a BSP file with the provided file name. This .bsp file can be distributed and later used as a source for creating a new PetaLinux project. This command is generally used as the last step in producing a project image that can be distributed to other users. All AMD reference BSPs for PetaLinux are packaged using this workflow.

### ***petalinux-package --bsp Command Options***

The following table details the options that are valid when packaging a PetaLinux BSP file with the `petalinux-package --bsp` command.

**Table 54: petalinux-package --bsp Command Options**

| Option                           | Functional Description                                                                                                                                                       | Value Range    | Default Value |
|----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|---------------|
| -o, --output BSPNAME             | Path on disk to store the BSP file. File name is of the form BSPNAME.bsp. This is required.                                                                                  | User-specified | None          |
| -p, --project PROJECT            | PetaLinux project directory path. In the BSP context, multiple project areas can be referenced and included in the output BSP file. This is optional.                        | User-specified | None          |
| --force                          | Overwrite existing files on disk. This is optional.                                                                                                                          | None           | None          |
| --clean                          | Clean the hardware implementation results to reduce package size. This is optional.                                                                                          | None           | None          |
| --hwsource HWPROJECT             | Path to an AMD Vivado™ design tools project to include in the BSP file. AMD Vivado™ hardware project is added to the hardware directory of the output BSP. This is optional. | None           | None          |
| --exclude-from-file EXCLUDE_FILE | Excludes the files mentioned in EXCLUDE_FILE from BSP.                                                                                                                       | User-specified | None          |

## ***petalinux-package --bsp Command Examples***

The following examples demonstrate the right usage of the `petalinux-package --bsp` command.

- Clean the project and generate the BSP installation image (.bsp file).

```
petalinux-package --bsp --clean -o <PATH-TO-BSP> -p <PATH-TO-PROJECT>
```

- Generate the BSP installation image that includes a reference hardware definition.

```
petalinux-package --bsp -p <PATH-TO-PROJECT> --hwsource <PATH-TO-HW-EXPORT> -o <PATH-TO-BSP>
```

- Generate the BSP installation image from a neutral location.

```
petalinux-package --bsp -p <PATH-TO-PROJECT> -o <PATH-TO-BSP>
```

- Generate the BSP installation image excluding some files.

```
petalinux-package --bsp -p <path_to_project> -o <path_to_bsp> --exclude-from-file <EXCLUDE_FILE>
```

## ***petalinux-package --prebuilt***

The `petalinux-package --prebuilt` command creates a new directory named “pre-built” inside the directory hierarchy of the specified PetaLinux project. This directory contains the required files to facilitate booting a board immediately without completely rebuilding the project. This workflow is intended for those who later create a PetaLinux BSP file for distribution using the `petalinux-package --bsp` workflow. All AMD reference PetaLinux BSPs contain a prebuilt directory.

## ***petalinux-package --prebuilt Command Options***

The following table details the options that are valid when including prebuilt data in the project with the `petalinux-package --prebuilt` workflow.

*Table 55: petalinux-package --prebuilt Command Options*

| Options                            | Functional Description                                                                                                                                 | Value Range    | Default Value                                     |
|------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|---------------------------------------------------|
| <code>-p, --project PROJECT</code> | PetaLinux project directory path. This is optional.                                                                                                    | User-specified | Current Directory                                 |
| <code>--force</code>               | Overwrite existing files on disk. This is optional.                                                                                                    | None           | None                                              |
| <code>--clean</code>               | Remove all files from the <plnx-proj-root>/prebuilt directory. This is optional.                                                                       | None           | None                                              |
| <code>--fpga BITSTREAM</code>      | Include the BITSTREAM file in the prebuilt directory. This is optional.                                                                                | User-specified | <project>/images/linux/*.bit                      |
| <code>-a, --add src:dest</code>    | Add the file/directory specified by src to the directory specified by dest in the prebuilt directory. This is optional and can be used multiple times. | User-specified | The default dest path is <project>/prebuilt/linux |

## ***petalinux-package --prebuilt Command Examples***

The following examples demonstrate proper usage of the `petalinux-package --prebuilt` command.

- Include a specific bitstream in the prebuilt area.

```
petalinux-package --prebuilt --fpga <BITSTREAM>
```

- Include a specific data file in the prebuilt area. For example, add a custom readme to the prebuilt directory.

```
petalinux-package --prebuilt -a <Path to readme>:images/<custom readme>
```

## ***petalinux-package --sysroot***

The `petalinux-package --sysroot` command installs an SDK to a specified directory in publish mode. This directory can be used as sysroot for application development.

## ***petalinux-package --sysroot Command Options***

The following table details the options that are valid when installing an SDK with the `petalinux-package --sysroot` workflow. The SDK must be published previously using the `petalinux-build --sdk` command.

**Table 56: petalinux-package --sysroot Command Options**

| Options               | Functional Description                                   | Value Range    | Default Value                        |
|-----------------------|----------------------------------------------------------|----------------|--------------------------------------|
| -p, --project PROJECT | PetaLinux project directory path. This is optional.      | User-specified | Current Directory                    |
| -s, --sdk SDK         | SDK path on disk to SDK .sh file. This is optional.      | None           | <plnx-proj-root>/images/linux/sdk.sh |
| -d, --dir DIRECTORY   | Directory path on disk to install SDK. This is optional. | None           | <plnx-proj-root>/images/linux/sdk    |

## ***petalinux-package --sysroot Command Examples***

The following examples demonstrate the proper usage of the `petalinux-package --sysroot` command.

- Install default SDK to default directory.

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

- Install specified SDK to default directory.

```
petalinux-package --sysroot -s <PATH-TO-SDK>
```

- Install specified SDK to specified directory.

```
petalinux-package --sysroot -s <PATH-to-SDK> -d <PATH-TO-INSTALL-DIR>
```

## ***petalinux-package --wic***

The following command generates partitioned images from the `images/linux` directory. Image generation is driven by partitioning commands contained in the kickstart file (`.wks`). The default `.wks` file is FAT32 with 2 GB and EXT4 with 4 GB. You can find the default kickstart file in `<project-root>/build/rootfs.wks` after the `petalinux-package --wic` command is executed.

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

## ***petalinux-package --wic Command Options***

**Table 57: petalinux-package --wic Command Options**

| Options                   | Functional Description                                                        | Value Range | Default Value                                     |
|---------------------------|-------------------------------------------------------------------------------|-------------|---------------------------------------------------|
| -w, --wks <WKS_FILE>      | Specify the <code>wic</code> file to be used to create partitions of SD card. | None        | None                                              |
| -o, --outdir <OUTPUT_DIR> | Specify the output directory to create <code>petalinux-sdimage.wic</code> .   | None        | Default: <code>&lt;root&gt;/images/linux</code> . |

**Table 57: petalinux-package --wic Command Options (cont'd)**

| <b>Options</b>                                         | <b>Functional Description</b>                                                                                       | <b>Value Range</b> | <b>Default Value</b>                                                                                                                                                                                                                      |
|--------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>-i, --images-dir &lt;IMAGES_DIR&gt;</code>       | Specify the images directory the boot files were located.                                                           | None               | Default: <proot>/images/linux.                                                                                                                                                                                                            |
| <code>-c, --rootfs-file &lt;ROOTFS_FILE&gt;</code>     | Specify the compressed root filesystem file that is extracted into the /rootfs directory.                           | None               | Default: <proot>/images/linux/rootfs.tar.gz<br>Supports only the tar.gz format.                                                                                                                                                           |
| <code>-r, --rootfs-dir &lt;ROOTFS_DIR&gt;</code>       | Specify the extracted root filesystem directory that is copied to /rootfs directory.                                | None               | None                                                                                                                                                                                                                                      |
| <code>-b, --bootfiles &lt;BOOT_FILES&gt;</code>        | Specify boot files which should be copied into /boot directory.                                                     | None               | Default boot files:<br>For Zynq 7000 devices: BOOT.BIN, uImage, boot.scr.<br>For Zynq UltraScale+ MPSoC: BOOT.BIN, Image, boot.scr, and ramdisk.cpio.gz<br>For Versal adaptive SoC: BOOT.BIN, Image, boot.scr, and ramdisk.cpio.gz.u-boot |
| <code>-e, --extra-bootfiles &lt;EXTRA_FILES&gt;</code> | Specify extra boot files which should be copied into /boot directory. Make sure these are part of images directory. | None               | None                                                                                                                                                                                                                                      |

## ***petalinux-package --wic Command Examples***

### **Packaging the WIC Image using Default Images**

The following command generates the wic image, `petalinux-sdimage.wic`, in the `images/linux` folder with the default images from the `images/linux` directory.

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

### **Packaging the WIC Image in a Specific Folder**

The following command generates the wic image, `petalinux-sdimage.wic`, in the `wicimage/` folder.

```
petalinux-package --wic --outdir wicimage/
```

### **Packaging the WIC Image with Specified Images Path**

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

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

## Packaging Custom Bootfiles into the /boot Directory

- To copy boot.bin userfile1 userfile2 files from the <plnx-proj-root>/images/linux directory to the /boot of media, use the following command:

```
petalinux-package --wic --bootfiles "boot.bin userfile1 userfile2"
```

This generates the wic image with specified files copied into the /boot directory.

**Note:** Ensure that these files are part of the images directory.

- To copy the uImage file named kernel to the /boot directory, use the following command:

```
petalinux-package --wic --extra-bootfiles "uImage:kernel"
```

- To copy the default bootfiles and specified bootfiles by user files into the /boot directory, use the following command:

```
petalinux-package --wic --bootfiles "userfiles/*"
```

- To copy all the files in the userfiles/ directory to the /boot/user\_boot directory, use the following command:

```
petalinux-package --wic --extra-bootfiles "userfiles/*:user_boot"
```

**Note:** Ensure that these files are part of the images directory.

## Packaging Custom Root File System

The following command unpacks your custom-rootfs.tar.gz file and copies it to the /rootfs directory.

```
petalinux-package --wic --rootfs-file custom-rootfs.tar.gz
```

## Customizing WIC Partitions

PetaLinux uses the kickstart (.wks) file to define the partitions to create the WIC image. When the petalinux-package --wic command is executed, the default .wks file is placed in <plnx-proj-root>/build/wic/rootfs.wks. This can be modified and provided as an input to create the WIC images as per the requirement. Following is the default rootfs.wks file:

```
part /boot --source bootimg-partition --ondisk mmcblk0 --fstype=vfat --
label boot --active --align 4 --size 800
part / --source rootfs --ondisk mmcblk0 --fstype=ext4 --label root --align
4 --size 2400
```

You can refer to <https://www.yoctoproject.org/docs/latest/ref-manual/ref-manual.html#ref-kickstart> for each argument specified in the wks file.

Once you have the updated .wks file, use the following command to create WIC image:

```
petalinux-package --wic --wks <path to the wks file>
```

---

## **petalinux-util**

The `petalinux-util` tool provides various support services to the other PetaLinux workflows. The tool itself provides several workflows depending on the support function needed.

### **petalinux-util --gdb**

The `petalinux-util --gdb` command is a wrapper around the standard GNU GDB debugger and simply launches the GDB debugger in the current terminal. Executing `petalinux-util --gdb --help` at the terminal prompt provides verbose GDB options that can be used.

For GDB GUI-based debugging, use the AMD Vitis™ software platform. For more information regarding GDB, see *Vitis Unified Software Platform Documentation: Embedded Software Development* ([UG1400](#)).

#### ***petalinux-util --gdb command Examples***

The following example demonstrates proper usage of the `petalinux-util --gdb` command. To launch the GNU GDB debugger, use the following command:

```
petalinux-util --gdb
```

### **petalinux-util --dfu-util**

The `petalinux-util --dfu-util` command is a wrapper around the standard `dfu-util`, and launches `dfu-util` in the current terminal. Executing `petalinux-util --dfu-util --help` at the terminal prompt provides verbose `dfu-util` options that can be used.

#### ***petalinux-util --dfu-util Command Examples***

The following example demonstrates proper usage of the `petalinux-util --dfu-util` command. To launch the `dfu-util`, use the following command:

```
petalinux-util --dfu-util
```

## ***petalinux-util --xsdb-connect***

The `petalinux-util --xsdb-connect` command provides XSDB connection to QEMU. This is for AMD Zynq™ UltraScale+™ MPSoC and Zynq 7000 devices only.

For more information regarding XSDB, see *Vitis Unified Software Platform Documentation: Embedded Software Development* ([UG1400](#)).

### ***petalinux-util --xsdb-connect Options***

The following table details the options that are valid when using the `petalinux-util --xsdb-connect` command.

**Table 58: petalinux-util --xsdb-connect Options**

| Option                                | Functional Description                                                                                                                                                                                                                                         | Value Range    | Default Value |
|---------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|---------------|
| <code>--xsdb-connect HOST:PORT</code> | Host and the port XSDB should connect to. This should be the host and port that QEMU has opened for GDB connections. It can be found in the QEMU command line arguments from: <code>--gdb tcp: &lt;QEMU_HOST&gt;: &lt;QEMU_PORT&gt;</code> . This is required. | User-specified | None          |

## ***petalinux-util --jtag-logbuf***

The `petalinux-util --jtag-logbuf` command logs the Linux kernel printk output buffer that occurs when booting a Linux kernel image using JTAG. This workflow is intended for debugging the Linux kernel for review and debug. This workflow can be useful when the Linux kernel is not producing output using a serial terminal. For details on how to boot a system using JTAG, see the `petalinux-boot --jtag` command. For MicroBlaze™ CPUs, the image that can be debugged is `<plnx-proj-root>/image/linux/image.elf`. For Arm® cores, the image that can be debugged is `<plnx-proj-root>/image/linux/vmlinu`.

### ***petalinux-util --jtag-logbuf Options***

The following table details the options that are valid when using the `petalinux-util --jtag-logbuf` command.

**Table 59: petalinux-util --jtag-logbuf Options**

| Option                             | Functional Description                                | Value Range    | Default Value     |
|------------------------------------|-------------------------------------------------------|----------------|-------------------|
| <code>-i, --image IMAGEPATH</code> | Linux kernel ELF image. This is required.             | User-specified | None              |
| <code>--hw_server-url URL</code>   | URL of the hw_server to connect to. This is optional. | User-specified | None              |
| <code>-p, --project PROJECT</code> | PetaLinux project directory path. This is optional.   | User-specified | Current Directory |

**Table 59: petalinux-util --jtag-logbuf Options (cont'd)**

| Option        | Functional Description                                                              | Value Range | Default Value |
|---------------|-------------------------------------------------------------------------------------|-------------|---------------|
| --noless      | Do not pipe output to the less command. This is optional.                           | None        | None          |
| -v, --verbose | Displays additional output messages. This is optional.                              | None        | None          |
| -h, --help    | Displays tool usage information. This is optional.                                  | None        | None          |
| --dryrun      | Prints the commands required to extract the kernel log buffer, but do not run them. | None        | None          |

## ***petalinux-util --jtag-logbuf Examples***

The following examples demonstrate proper usage of the `petalinux-util --jtag-logbuf` command.

- Launch a specific Linux kernel image

```
$ petalinux-util --jtag-logbuf -i <PATH-TO-IMAGE>
```

- Launch the JTAG logger from a neutral location. This workflow is for AMD Zynq™ 7000 devices only

```
$ petalinux-util --jtag-logbuf -i <PATH-TO-IMAGE> -p <PATH-TO-PROJECT>
```

## ***petalinux-util --find-xsa-bitstream***

The `petalinux-util --find-xsa-bitstream` gives the name of bitstream packed in the XSA file.

### ***petalinux-util --find-xsa-bitstream Options***

The following table details the options that are valid when using the `petalinux-util --find-xsa-bitstream` command.

**Table 60: petalinux-util --find-xsa-bitstream Options**

| Option                              | Functional Description                                     | Value Range | Default Value                                                          |
|-------------------------------------|------------------------------------------------------------|-------------|------------------------------------------------------------------------|
| <code>--xsa-file &lt;XSA&gt;</code> | Argument to specify the XSA file to use. This is optional. | None        | system.xsa file in the <project>/project-spec/hw-description directory |

## ***petalinux-util -- find-xsa-bitstream Examples***

The following examples demonstrate proper usage of the `petalinux-util --find-xsa-bitstream` command.

- To find the default bitstream of a project

```
petalinux-util --find-xsa-bitstream
```

- To find the bitstream of a xsa

```
petalinux-util --find-xsa-bitstream --xsa-file <path to xsa file>
```

## **petalinux-upgrade**

To upgrade the workspace, use the `petalinux-upgrade` command.

### **petalinux-upgrade Options**

*Table 61: petalinux-upgrade Options*

| Options                      | Functional description                           | Value Range                                                                                                                           | Default Range |
|------------------------------|--------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|---------------|
| <code>-h --help</code>       | Displays usage information.                      | None                                                                                                                                  | None          |
| <code>-f --file</code>       | Local path to target system software components. | User-specified.                                                                                                                       | None          |
| <code>-u --url</code>        | URL to target system software components.        | User-specified.                                                                                                                       | None          |
| <code>-w, --wget-args</code> | Passes additional wget arguments to the command. | Additional wget options.                                                                                                              | None          |
| <code>-pl--platform</code>   | Specifies the architecture name to upgrade.      | aarch64: sources for Zynq UltraScale+ MPSoC and Versal<br>arm: sources for Zynq devices<br>microblaze: sources for microblaze devices | None          |

## **petalinux-devtool**

The `petalinux-devtool` is a utility that uses the Yocto devtool to enable you to build, test, and package software. The following table details the available options for the `petalinux-devtool` command.

### **petalinux-devtool Command Line Options**

*Table 62: petalinux-devtool Command Line Options*

| Option           | Functional Description |
|------------------|------------------------|
| <code>add</code> | Add a new recipe       |

**Table 62: petalinux-devtool Command Line Options (cont'd)**

| Option               | Functional Description                              |
|----------------------|-----------------------------------------------------|
| modify               | Modify the source for an existing recipe            |
| upgrade              | Upgrade an existing recipe                          |
| status               | Show workspace status                               |
| search               | Search available recipes                            |
| latest-version       | Report the latest version of an existing recipe     |
| check-upgrade-status | Report upgradability for multiple (or all) recipes  |
| build                | Build a recipe                                      |
| rename               | Rename a recipe file in the workspace               |
| edit-recipe          | Edit a recipe file                                  |
| find-recipe          | Find a recipe file                                  |
| configure-help       | Get help on configure script options                |
| update-recipe        | Apply changes from external source tree to recipe   |
| reset                | Remove a recipe from your workspace                 |
| finish               | Finish working on a recipe in your workspace        |
| deploy-target        | Deploy recipe output files to live target machine   |
| undeploy-target      | Undeploy recipe output files in live target machine |
| build-image          | Build image including workspace recipe packages     |
| create-workspace     | Set up workspace in an alternative location         |
| configure            | Runs configure command                              |
| export               | Export workspace into a tar archive                 |
| extract              | Extract the source for an existing recipe           |
| sync                 | Synchronize the source tree for an existing recipe  |
| import               | Import exported tar archive into workspace          |
| menuconfig           | Alter build-time configuration for a recipe         |

## petalinux-devtool Examples

### Adding a New Recipe to the Workspace Layer

To add a new recipe to the workspace layer, use the `petalinux-devtool add` command. For `petalinux-devtool add bbexample https://github.com/whbruce/bbexample.git` This command fetches the source from the specified URL and creates the recipe `bbexample` in the Devtool Workspace directory.

`petalinux-devtool add`

```
petalinux-devtool add bbexample https://github.com/whbruce/bbexample.git
[INFO] Sourcing buildtools
[INFO] Silentconfig project
[INFO] Silentconfig rootfs
[INFO] Generating workspace directory
[INFO] devtool add bbexample https://github.com/whbruce/bbexample.git
```

```
NOTE: Starting bitbake server...
NOTE: Started PRServer with DBfile: xilinx-zcu102-2023.2/build/cache/
prserv.sqlite3, Address: 127.0.0.1:40721, PID: 21540
NOTE: Starting bitbake server...
NOTE: Started PRServer with DBfile: xilinx-zcu102-2023.2/build/cache/
prserv.sqlite3, Address: 127.0.0.1:44081, PID: 21572
INFO: Fetching git://github.com/whbruce/
bbexample.git;protocol=https;branch=master...
Loading cache: 100% |
#####
| Time: 0:00:02
Loaded 6336 entries from dependency cache.
Parsing recipes: 100% |
#####
| Time: 0:00:02
Parsing of 4413 .bb files complete (4399 cached, 14 parsed). 6350 targets,
344 skipped, 1 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
Initialising tasks: 100% |
#####
| Time: 0:00:01
Sstate summary: Wanted 0 Local 0 Mirrors 0 Missed 0 Current 0 (0% match, 0%
complete)
NOTE: No setscene tasks
NOTE: Executing Tasks
NOTE: Tasks Summary: Attempted 2 tasks of which 0 didn't need to be rerun
and all succeeded.
INFO: Using default source tree path xilinx-zcu102-2023.2/components/yocto/
workspace/sources/bbexample
NOTE: Reconnecting to bitbake server...
NOTE: Previous bitbake instance shutting down?, waiting to retry...
NOTE: Retrying server connection (#1)...
NOTE: Reconnecting to bitbake server...
NOTE: Reconnecting to bitbake server...
NOTE: Previous bitbake instance shutting down?, waiting to retry...
NOTE: Previous bitbake instance shutting down?, waiting to retry...
NOTE: Retrying server connection (#1)...
NOTE: Retrying server connection (#1)...
NOTE: Starting bitbake server...
NOTE: Started PRServer with DBfile: xilinx-zcu102-2023.2/build/cache/
prserv.sqlite3, Address: 127.0.0.1:37295, PID: 22750
INFO: Recipe xilinx-zcu102-2023.2/components/yocto/workspace/recipes/
bbexample/bbexample_git.bb has been automatically created; further editing
may be required to make it fully functional
```

## Getting the Status of the Recipes in Your Workspace

Use the `petalinux-devtool status` command to list the recipes currently in your workspace. Information includes the paths to their respective external source trees.

`petalinux-devtool status`

```
petalinux-devtool status
[INFO] Sourcing buildtools
[INFO] Silentconfig project
[INFO] Silentconfig rootfs
[INFO] Generating workspace directory
[INFO] devtool status
NOTE: Starting bitbake server...
```

```
NOTE: Started PRServer with DBfile: xilinx-zcu102-2023.2/build/cache/prserv.sqlite3, Address: 127.0.0.1:39421, PID: 23789
bbexample: xilinx-zcu102-2023.2/components/yocto/workspace/sources/
bbexample (xilinx-zcu102-2023.2/components/yocto/workspace/recipes/
bbexample/bbexample_git.bb)
```

## Modifying an Existing Recipe

Use the `petalinux-devtool modify` command to begin modifying the source of an existing recipe. This command extracts the source for a recipe to the Devtool Workspace directory, checks out a branch for development, and applies the patches, if any, from the recipe as commits on top.

Use the following command to modify the `linux-xlnx` recipe:

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

```
petalinux-devtool modify linux-xlnx
[INFO] Sourcing buildtools
[INFO] Silentconfig project
[INFO] Silentconfig rootfs
[INFO] Generating workspace directory
[INFO] devtool modify linux-xlnx
NOTE: Starting bitbake server...
NOTE: Started PRServer with DBfile: xilinx-zcu102-2023.2/build/cache/prserv.sqlite3, Address: 127.0.0.1:42543, PID: 24530
NOTE: Reconnecting to bitbake server...
NOTE: Previous bitbake instance shutting down?, waiting to retry...
NOTE: Retrying server connection (#1)...
NOTE: Started PRServer with DBfile: xilinx-zcu102-2023.2/build/cache/prserv.sqlite3, Address: 127.0.0.1:42809, PID: 24558
Loading cache: 100% | #####
| Time: 0:00:01
Loaded 6336 entries from dependency cache.
Parsing recipes: 100% | #####
Time: 0:00:02
Parsing of 4413 .bb files complete (4399 cached, 14 parsed). 6350 targets, 344 skipped, 1 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
Initialising tasks: 100% | #####
Time: 0:00:02
Sstate summary: Wanted 74 Local 0 Mirrors 51 Missed 23 Current 72 (68%
match, 84% complete)
NOTE: Executing Tasks
NOTE: Tasks Summary: Attempted 656 tasks of which 656 didn't need to be
rerun and all succeeded.
INFO: Copying kernel config to workspace
INFO: Recipe linux-xlnx now set up to build from xilinx-zcu102-2023.2/
components/yocto/workspace/sources/linux-xlnx
```

## Building the Recipe

Use the `petalinux-devtool build` command to build your recipe. This command is equivalent to the `bitbake -c populate_sysroot` command. You must supply the root name of the recipe (that is, do not provide versions, paths, or extensions), as shown:

```
petalinux-devtool build linux-xlnx

petalinux-devtool build linux-xlnx
[INFO] Sourcing buildtools
[INFO] Silentconfig project
[INFO] Silentconfig rootfs
[INFO] Generating workspace directory
[INFO] devtool build linux-xlnx
NOTE: Starting bitbake server...
NOTE: Started PRServer with DBfile: xilinx-zcu102-2023.2/build/cache/
prserv.sqlite3, Address: 127.0.0.1:37513, PID: 27000
NOTE: Started PRServer with DBfile: xilinx-zcu102-2023.2/build/cache/
prserv.sqlite3, Address: 127.0.0.1:35987, PID: 27025
Loading cache: 100% |
#####
| Time: 0:00:01
Loaded 6336 entries from dependency cache.
Parsing recipes: 100% |
#####
| Time: 0:01:23
Parsing of 4413 .bb files complete (4398 cached, 15 parsed). 6350 targets,
344 skipped, 1 masked, 0 errors.
Removing 1 recipes from the xilinx_zcu102 sysroot: 100% |
#####
| Time: 0:00:00
NOTE: Started PRServer with DBfile: xilinx-zcu102-2023.2/build/cache/
prserv.sqlite3, Address: 127.0.0.1:36161, PID: 28745
Loading cache: 100% |
#####
| Time: 0:00:03
Loaded 6336 entries from dependency cache.
Parsing recipes: 100% |
#####
| Time: 0:01:14
Parsing of 4413 .bb files complete (4398 cached, 15 parsed). 6350 targets,
344 skipped, 1 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
Initialising tasks: 100% |
#####
| Time: 0:00:05
Checking sstate mirror object availability: 100% |
#####
| Time: 0:00:22
Sstate summary: Wanted 401 Local 22 Mirrors 177 Missed 202 Current 880 (49%
match, 84% complete)
WARNING: The qemu-xilinx-system-native:do_configure sig is computed to be
6d36811060421830430787c675634127effc326c1300c2f998eb2b2d9e8719f0, but the
sig is locked to
376b1b71f07fe035fc71b594319efed81fe9521d167a63eb53eda66db904b045 in
SIGGEN_LOCKEDSIGS_t-x86-64
Removing 3 stale sstate objects for arch xilinx_zcu102: 100% |
#####
| Time: 0:00:00
NOTE: Executing Tasks
NOTE: linux-xlnx: compiling from external source tree xilinx-zcu102-2023.2/
```

```
components/yocto/workspace/sources/linux-xlnx
NOTE: Tasks Summary: Attempted 3410 tasks of which 3361 didn't need to be
rerun and all succeeded.

Summary: There was 1 WARNING message.
```

## Building Your Image

Use the `petalinux-devtool build-image` command to build an image using the devtool flow. This includes the recipes which are the workspace directory. When running this command you must specify the image name to be built.

```
petalinux-devtool build-image petalinux-image-minimal
```

*Figure 45: petalinux-devtool build-image*

```
:petalinux-devtool build-image petalinux-image-minimal
INFO: Sourcing build tools
[INFO] Sourcing build environment
[INFO] Generating workspace directory
[INFO] devtool build-image petalinux-image-minimal
NOTE: Starting bitbake server...
...
NOTE: Executing Tasks
NOTE: Setscene tasks completed
NOTE: bbexample: compiling from external source tree <plnx-proj-dir>/components/yocto/workspace/sources/bbexample
NOTE: Tasks Summary: Attempted 3623 tasks of which 2805 didn't need to be rerun and all succeeded.
INFO: Successfully built petalinux-image-minimal. You can find output files in <plnx-proj-dir>/build/tmp/deploy/images/zynqmp-generic
```

## Resetting the Recipe

Use the `petalinux-devtool reset` command to remove the recipe and its configurations from Devtool Workspace directory. This command does not add/append changes in the Devtool Workspace to any layers, you need to update the recipe append file before running the reset command.

```
petalinux-devtool reset linux-xlnx
```

*Figure 46: petalinux-devtool reset*

```
:petalinux-devtool reset linux-xlnx
INFO: Sourcing build tools
[INFO] Sourcing build environment
[INFO] Generating workspace directory
[INFO] devtool reset linux-xlnx
NOTE: Starting bitbake server...
INFO: Cleaning sysroot for recipe linux-xlnx...
INFO: Leaving source tree <plnx-proj-dir>/components/yocto/workspace/sources/linux-xlnx as-is; if you no longer need it then please delete it manually
```

**Note:** You can find a list of examples at <https://www.yoctoproject.org/docs/latest/ref-manual/ref-manual.html#ref-devtool-reference>. Make sure to use `petalinux-devtool` in place of `devtool` in the examples.

# Additional Resources and Legal Notices

## Finding Additional Documentation

### Documentation Portal

The AMD Adaptive Computing Documentation Portal is an online tool that provides robust search and navigation for documentation using your web browser. To access the Documentation Portal, go to <https://docs.xilinx.com>.

### Documentation Navigator

Documentation Navigator (DocNav) is an installed tool that provides access to AMD Adaptive Computing documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the AMD Vivado™ IDE, select **Help → Documentation and Tutorials**.
- On Windows, click the **Start** button and select **Xilinx Design Tools → DocNav**.
- At the Linux command prompt, enter `docnav`.

**Note:** For more information on DocNav, refer to the *Documentation Navigator User Guide (UG968)*.

### Design Hubs

AMD 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.
- Go to the [Design Hubs](#) web page.

# Support Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see [Support](#).

# Revision History

The following table shows the revision history for this document.

| Section                                                                               | Revision Summary                                                                                                                                                                                                                |
|---------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>10/18/2023 Version 2023.2</b>                                                      |                                                                                                                                                                                                                                 |
| Troubleshooting                                                                       | Added new topic <a href="#">Password Recovery</a> .                                                                                                                                                                             |
| Accessing BitBake/Devtool in a Project                                                | Updated <a href="#">Steps to Access the BitBake Utility</a> .                                                                                                                                                                   |
| FPGA Manager Configuration and Usage for Zynq 7000 Devices and Zynq UltraScale+ MPSoC | Updated <a href="#">DFX Support</a> .                                                                                                                                                                                           |
| <b>05/16/2023 Version 2023.1</b>                                                      |                                                                                                                                                                                                                                 |
| Installation Requirements                                                             | Updated supported operating systems.                                                                                                                                                                                            |
| PMU Firmware Configuration for Zynq UltraScale+ MPSoC                                 | Added extra compiler flags.                                                                                                                                                                                                     |
| DTG Settings                                                                          | Added new configurations.                                                                                                                                                                                                       |
| Yocto Settings                                                                        | Added new configuration.                                                                                                                                                                                                        |
| Appendix A: Migration                                                                 | <a href="#">Removed Auto Login</a><br><a href="#">Removed platform-auto.h Changes in Machine Conf</a><br><a href="#">Removed config.project Usage of uenv.txt</a><br><a href="#">Renamed ARM Trusted Firmware Configuration</a> |
| Subsystem AUTO Hardware Settings                                                      | Added ethernet submask changes.                                                                                                                                                                                                 |
| FPGA Manager Changes                                                                  | Updated FPGA manager configuration.                                                                                                                                                                                             |
| Chapter 5: Packaging and Booting                                                      | <a href="#">Boot from HBM/Higher DDR/LP DDR</a>                                                                                                                                                                                 |
| PetaLinux Menuconfig System                                                           | Custom dtsi for DFX                                                                                                                                                                                                             |

# References

These documents provide supplemental material useful with this guide:

1. PetaLinux Documentation ([www.xilinx.com/petalinux](http://www.xilinx.com/petalinux))
2. [Answer Record 55776](#)
3. [Zynq UltraScale+ MPSoC: Software Developers Guide \(UG1137\)](#)
4. [Versal Adaptive SoC System Software Developers Guide \(UG1304\)](#)
5. [Bootgen User Guide \(UG1283\)](#)
6. Xilinx Quick Emulator User Guide (QEMU) ([UG1169](#))

7. Libmetal and OpenAMP for Zynq Devices User Guide ([UG1186](#))
  8. [www.wiki.xilinx.com/Linux](http://www.wiki.xilinx.com/Linux)
  9. [PetaLinux Yocto Tips](#)
  10. [Yocto Project Technical FAQ](#)
  11. *Vitis Unified Software Platform Documentation: Embedded Software Development* ([UG1400](#))
- 

## Please Read: Important Legal Notices

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes. THIS INFORMATION IS PROVIDED "AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

### 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

© Copyright 2014-2023 Advanced Micro Devices, Inc. AMD, the AMD Arrow logo, Vivado, Versal, Vitis, Zynq, and combinations thereof are trademarks of Advanced Micro Devices, Inc. OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos. PCI, PCIe, and PCI Express are trademarks of PCI-SIG and used under license. AMBA, AMBA Designer, Arm, ARM1176JZ-S, CoreSight, Cortex, PrimeCell, Mali, and MPCore are trademarks of Arm Limited in the US and/or elsewhere. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.