See all versions of this document

PetaLinux Tools Documentation

Command Line Reference Guide

UG1157 (v2020.1) June 3, 2020

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
06/03/2020 Version 2020.1	
petalinux-package --wic Command Examples	Added new section
Adding Custom dtsi and bit Files to the FPGA Manager for Zynq-7000 Devices and Zynq UltraScale+ MPSoCs	Added new section
Building and Installing eSDK	Added new section
Packaging Sources and Licenses	Added new section

Table of Contents

Chapter 1

PetaLinux Tools

Introduction

PetaLinux is a development and build environment that automates many of the tasks required to boot embedded Linux on Zynq®-7000 SoCs and Xilinx® 7 series FPGAs. It uses the Yocto Project underneath for configuring and building various components. This document contains detailed information about the various tools that comprise the PetaLinux environment.

There are seven independent tools that make up the PetaLinux design flow. They are:

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

In most cases, the PetaLinux tools 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.

Design Flow Overview

Most PetaLinux tools follow a sequential workflow model. The table below provides an example design workflow to demonstrate the order in which tasks should be completed and the corresponding tool or workflow needed for that task.

Table 1: **Design Flow Overview**

Design Flow Step	Tool / Workflow
Hardware platform creation	Vivado® Design Suite
Create PetaLinux project	`petalinux-create -t project`
Initialize PetaLinux project	`petalinux-config --get-hw-description`
Configure system-level options	`petalinux-config`
Create user components	`petalinux-create -t COMPONENT`
Configure the Linux kernel	`petalinux-config -c kernel`
Configure the root file system	`petalinux-config -c rootfs`
Build the system	`petalinux-build`
Test the system on qemu	`petalinux-boot --qemu`
Deploy the system	`petalinux-package --boot`
Update the PetaLinux tool system software components	`petalinux-upgrade --url/--file`

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 2: **petalinux-create Command Line Options**

Option	Functional Description	Value Range	Default Value
`-t,--type TYPE`	Specify the TYPE of object to create. This is required.	• project • apps • modules	None
`-n,--name NAME`	Create object with the specified NAME. This is optional when creating a project from a BSP source. Otherwise, this is required.	User-specified	When creating a project from a BSP source, the project takes the name of the source BSP.
`-p,--project PROJECT`	PetaLinux project directory path for component creation in a project. This is optional.	User-specified	Current Directory

Table 2: **petalinux-create Command Line Options** *(cont'd)*

Option	Functional Description	Value Range	Default Value
`--force`	Overwrite existing files on disk. This is optional.	None	None
`-h,--help`	Display usage information. This is optional.	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 3: **petalinux-create -t project Options**

Option	Functional Description	Value Range	Default Value
`--template TEMPLATE`	Assumes the specified CPU architecture, and is only required when `--source` is not provided.	• microblaze • zynqMP • zynq	None
`-s,--source SOURCE`	Creates project based on specified BSP file. SOURCE is the full path on disk to the BSP file. This is optional.	User-specified	None

Note: For Xilinx® boards, the `-s`, `--source` BSP flows are suggested. For custom boards, the `--template` flow is required.

petalinux-create -t project Examples

The following examples demonstrate proper usage of the `petalinux-create -t project` command.

- Create a new project from a reference BSP file

```
$ petalinux-create -t project -s <PATH-TO-BSP>
```

- Create a new project based on the MicroBlaze™ processor template

```
$ petalinux-create -t project -n <NAME> --template microblaze
```

By default, the directory structure created by `--template` is minimal, and is not useful for building a complete system until initialized using the `petalinux-config --get-hw-description` command. Projects created using a BSP file as their source are suitable for building immediately.

petalinux-create -t COMPONENT

The `petalinux-create -t COMPONENT` command allows you to create various components within the specified PetaLinux project. These components can then be selectively included or excluded from the final system by toggling them using the `petalinux-config -c rootfs` workflow.

petalinux-create -t COMPONENT Options

The `petalinux-create -t apps` command allows you to customize how application components are created. The following table details options that are common when creating applications within a PetaLinux project

Table 4: **petalinux-create -t apps Options**

Option	Functional Description	Value Range	Default Value
`-s,--source SOURCE`	Create the component from pre-existing content on disk. Valid formats are .tar.gz, .tar.bz2, .tar, .zip, and source directory (uncompressed). This is optional.	User-specified	None
`--template TEMPLATE`	Create the component using a pre-defined application template. This is optional.	• c • c++ • autoconf, for GNU autoconfig • fpgamanager • install, for applications which have prebuilt binary only	c
`--enable`	Upon creating the component, enable it in the project's root file system. You can also enable using the `petalinux-config -c rootfs`. This is optional.	None	Disabled
--srcuri	Creates an application with local sources or from remote source.	None	None

petalinux-create -t COMPONENT Examples

The following examples demonstrate proper usage of the `petalinux-create -t COMPONENT` command.

- Create an application component that is enabled in the root file system.

  ```
  $ petalinux-create -t apps -n <NAME> --template <template> --enable
  ```

- Create a new install-only application component. In this flow, nothing is compiled.

  ```
  $ petalinux-create -t apps -n <NAME> --template install
  ```

- Create a new kernel module and enable it.

  ```
  $ petalinux-create -t modules -n <name> --template <template> --enable
  ```

- Create an application with multiple source files.

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

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

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

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

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

  Note: This is applicable for applications and modules.

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

This section provides the mechanism and infrastructure required to work with readily (hand-stitched) available dtsi files instead of relying on the XSA to generate them when the FPGA manager is enabled. This generates the dtbo and bin files and copies them into the `rootfs /lib/firmware` directory and loads them when the system boots.

1. Create the FPGA manager template:

   ```
   $ petalinux-create -t apps --template fpgamanager -n can-interface --
   enable
               INFO: Create apps: can-interface
               INFO: New apps successfully created in <project-root-dir>/
   project-spec/meta-user/recipes-apps/can-interface
               INFO: Enabling created component...
               INFO: sourcing build environment
               INFO: silentconfig rootfs
               INFO: can-interface has been enabled
   ```

2. Replace default files with your own files:

   ```
   $ cp can.dtsi can.bit project-spec/meta-user/recipes-apps/can-interface/
   files/
   ```

3. Build the application:

   ```
   $ petalinux-build
   ```

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

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

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

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 5: **petalinux-config Command Line Options**

Option	Functional Description	Value Range	Default Value
`-p, --project <path to project directory>`	Specifies path to the project to be configured.	User-specified	Current Directory
`--get-hw-description <DIR containing XSA>`	Initializes or updates the hardware configuration for the PetaLinux project. Mutually exclusive with -c. This is required.	User-specified	Current Directory

Table 5: **petalinux-config Command Line Options** *(cont'd)*

Option	Functional Description	Value Range	Default Value
`-c,--component COMPONENT`	Configures the specified system component. Mutually exclusive with `--get-hw-description`. This is required.	• kernel • rootfs • u-boot • bootloader (for Zynq® UltraScale+™ MPSoC, Zynq architecture, and MicroBlaze™ CPU) • pmufw, for Zynq UltraScale+ MPSoC only • device-tree	None
`--defconfig DEFCONFIG`	Initializes the Linux kernel/U-Boot configuration using the specified `defconfig` file. Valid for Linux kernel and U-Boot. This is optional.	User-specified. For example, for Linux kernel, the file name of a file in `<kernel_source>/arch/<ARCH>/configs/` is `XXX_defconfig`. For U-Boot, the file name of a file in `<uboot_ source> / configs` is `XXX_defconfig`.	None
`--silentconfig`	Allows you to restore a prior configuration. Example: Execute the following command after enabling or disabling different configs by editing `<proj-root>/project-spec/configs/config` `$ petalinux-config --silentconfig`	None	None
`-v,--verbose`	Displays additional output messages. This is optional.	None	None
`-h,--help`	Displays tool usage information. This is optional.	None	None

petalinux-config --get-hw-description

The `petalinux-config --get-hw-description` command allows you to initialize or update a PetaLinux project with hardware-specific information from the specified Vivado® Design Suite hardware project. The components affected by this process can include FSBL configuration, U-Boot options, Linux kernel options, and the Linux device tree configuration. This workflow should be used carefully to prevent accidental and/or unintended changes to the hardware configuration for the PetaLinux project. The path used with this workflow is the directory that contains the XSA file rather than the full path to the XSA file itself. This entire option can be omitted if run from the directory that contains the XSA file.

petalinux-config --get-hw-description Examples

The following examples demonstrate proper usage of the `petalinux-config --get-hw-description` command.

- Initialize a PetaLinux project within the project directory with an external XSA.

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

- 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> -p
  <PATH-TO-PETALINUX-PROJECT>
  ```

petalinux-config -c COMPONENT

The `petalinux-config -c COMPONENT` command allows you to use a standard menuconfig interface to control how the embedded Linux system is built, and also generates the source code for embedded software applications. When `petalinux-config` is executed with no other options, it launches the system-level or "generic" menuconfig. This interface allows you to specify information such as the desired boot device or metadata about the system such as default hostname. The `petalinux-config -c kernel`, `petalinux-config -c u-boot`, and `petalinux-config -c rootfs` workflows launch the menuconfig interfaces for customizing the Linux kernel, U-Boot, and the root file system, respectively.

The `--silentconfig` option allows you to restore a prior configuration.

Example:

Execute the following command after enabling or disabling different configs by editing `<proj-root>/project-spec/configs/rootfs_config`

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

Use this command when you want to use the existing configurations without editing it. In this case, the menuconfig will not launch.

petalinux-config -c COMPONENT Examples

The following examples demonstrate proper usage of the `petalinux-config -c COMPONENT` command:

- Start the menuconfig for the system-level configuration.

  ```
  $ petalinux-config
  ```

- Enable different rootfs packages without opening the menuconfig. Execute below command after enabling or disabling different packages by editing `<proj-root>/project-spec/configs/rootfs_config`

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

- Load the Linux kernel configuration with a specific default configuration.

  ```
  $ petalinux-config -c kernel --defconfig xilinx_zynq_base_trd_defconfig
  ```

- Load the U-Boot configuration with a specific default configuration.

  ```
  $ petalinux-config -c u-boot --defconfig xilinx_zynqmp_zcu102_defconfig
  ```

- Generate the source code for FSBL/fs-boot.

  ```
  $ petalinux-config -c bootloader
  ```

The following warning message appears when `petalinux-config` or `petalinux-build` for components (for example: `petalinux-build -c u-boot`) is run. This message can be ignored.

WARNING! *SRC_URI is conditionally overridden in this recipe, thus several devtool-override-* branches have been created, one for each override that makes changes to SRC_URI. It is recommended that you make changes to the devtool branch first, then checkout and rebase each devtool-override-* branch and update any unique patches there (duplicates on those branches will be ignored by devtool finish/update-recipe).*

petalinux-build

The `petalinux-build` tool builds either the entire embedded Linux system or a specified component of the Linux system. This tool uses the Yocto Project underneath. Whenever `petalinux-build` is invoked, it internally calls `bitbake`. While the tool provides a single workflow, the specifics of its operation can be dictated using the `petalinux-build -c` and `petalinux-build -x` options.

petalinux-build Command Line Options

The following table outlines the valid options for the `petalinux-build` tool.

Table 6: **petalinux-build Command Line Options**

Option	Functional Description	Value Range	Default Value
`-p,--project PROJECT`	PetaLinux project directory path. This is optional.	User-specified	None

Table 6: **petalinux-build Command Line Options** *(cont'd)*

Option	Functional Description	Value Range	Default Value
`-c, --component COMPONENT`	Builds specified component. These are the default values which are supported. You can build against your own target (such as your application or module). This is optional.	• bootloader (Zynq® UltraScale+™ MPSoC, Zynq architecture, and MicroBlaze™ CPU) • kernel • u-boot • rootfs • pmufw, only for Zynq UltraScale+ MPSoC • arm-trusted-firmware, for Zynq UltraScale+ MPSoC. • device-tree • apps • modules	None
`-x, --execute STEP`	Executes specified build step. All Yocto tasks can be passed through this option. To get all tasks of a component, use "listtasks". This is optional.	• build • clean • cleanall • cleansstate • distclean • install • listtasks • populate_sysroot • package • mrproper	Build
`-v,--verbose`	Displays additional output messages. This is optional.	None	None
`-s, --sdk`	Builds Yocto SDK. This is optional.	None	None
--esdk	Builds Yocto e-SDK.This is optional.	None	Nnone
`-b`	Builds components ignoring dependencies. This is optional.	None	None
`-h`	Lists all the sub-components of a component. Valid only for rootfs. This is optional.	rootfs	None
`-f, --force`	Forces a specific task to run against a component, or a single task in the component, ignoring the stamps. This is optional.	None	None

petalinux-build --component

The `petalinux-build -c` option builds the specified component of the embedded system. When no components are specified, the `petalinux-build` tool operates on the project as a whole. User-created components for the root file system can be built by targeting those components by name (for example, with `-c <APP-NAME>`). This is equivalent to `bitbake <COMPONENT>`. Each recipe can be specified as a component for `petalinux-build -c <component>`. The component can be a user created app or package/package group in rootFS.

The `petalinux-build` command with no arguments runs `bitbake petalinux-user-image` internally. The default image target is `petalinux-user-image`. 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`.

Example to build base-files:

```
petalinux-build -c base-files
```

petalinux-build -c components

The following table summarizes the available components that can be targeted with this command:

Table 7: **petalinux-build -c components**

Component	Equivalent Bitbake Commands	Description
bootloader	`bitbake virtual/fsbl` `bitbake virtual/fsboot` (for MicroBlaze™ processor)	Build only the boot loader elf image and copy it into `<plnx-proj-root>/images/linux/`. For Zynq® UltraScale+™ MPSoC and Zynq-7000 devices, it is FSBL and for MicroBlaze™ processor, it is fs-boot.
device tree	`bitbake virtual/dtb`	Build only the device tree DTB file and copy it into `<plnx-proj-root>/images/linux/`. The device tree source is in `<plnx-proj-root>/components/plnx_workspace/device-tree/device-tree/`
arm-trusted-firmware	`bitbake virtual/arm-trusted-firmware`	Build only the ATF image and copy it into `<plnx-proj-root>/images/linux`
pmufw	`bitbake virtual/pmufw`	Build only the PMU firmware image and copy it into `<plnx-proj-root>/images/linux`
kernel	`bitbake virtual/kernel`	Build only the Linux kernel image and copy it into `<plnx-proj-root>/images/linux`
rootfs	`bitbake petalinux-user-image -c do_image_complete`	Build only the root file system. It generates the target rootfs in `${TMPDIR}/work/${MACHINE}/petalinux-user-image/1.0-r0/rootfs/` and the sysroot in `${TMPDIR}/tmp/sysroots/${MACHINE}`
u-boot	`bitbake virtual/bootloader`	Build only the U-Boot elf image and copy it into `<plnx-proj-root>/images/linux`

petalinux-build --execute

The `petalinux-build -x` option allows you to specify a build step to the `petalinux-build` tool to control how the specified components are manipulated. The Yocto task name has a `do_` prefixed to the `petalinux-build` step. All Yocto tasks can be passed through this option. To get all tasks of a component, use `listtasks`.

Commands for petalinux-build -x

The following table summarizes some of the available commands that can be used with this option:

Table 8: **petalinux-build -x options**

Component	Description
`clean`	Cleans build data for the target component.
`cleansstate`/ `distclean`	Removes the shared state cache of the corresponding component.
`cleanall`	Removes the downloads and shared state cache. Cleans the work directory of a component.
`mrproper`	Cleans the build area. This removes the `<plnx-proj-root>/build/`, `<TMPDIR>`, and `<plnx-proj-root>/images/` directories. This is the recommended way of cleaning the entire project.
`build`	Builds the target component.
`install`	Installs the target component. For bootloader, ATF, Linux kernel, U-Boot, and device tree, it copies the generated binary into `<plnx-proj-root>/images/linux/`. For the root file system and root file system component, it copies the generated binary to target the root file system host copy `${TMPDIR}/work/${MACHINE}/petalinux-user-image/1.0-r0/rootfs/`.
`package`	Generates FIT image `image.ub` from build area and copies it into `<plnx-proj-root>/images/linux/`. Valid for -c all or when no component is specified only.
`listtasks`	Gets all tasks of a specific component.

petalinux-build Examples

The following examples demonstrate proper usage of the `petalinux-build` command.

- Clear the build area of the PetaLinux project for archiving as a BSP or for revision control. This example retains the images directory of the project.

```
$ petalinux-build -x distclean
```

- Clean all build collateral from the U-Boot component of the PetaLinux project.

```
$ petalinux-build -c u-boot -x cleansstate
```

- Clean all build collateral. It removes build/, ${TMPDIR} and images. This brings the project to its initial state.

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

- Create an updated FIT image from the current contents of the deploy area.

  ```
  $ petalinux-build -x package
  ```

- Build the entire PetaLinux project.

  ```
  $ petalinux-build
  ```

- Build the kernel forcefully by ignoring the stamps (output of tasks from last successful build).

  ```
  $ petalinux-build -c kernel -f
  ```

- Compile kernel forcefully by ignoring do_compile task stamp.

  ```
  $ petalinux-build -c kernel -x compile -f
  ```

Building and Installing eSDK

Building eSDK

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

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

The following is the equivalent BitBake command.

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

Installing eSDK

To install the eSDK, follow these steps:

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

Packaging Sources and Licenses

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

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

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

4. To pack only the sysroot components, use the following command.

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

Note: You can find the archiver tar in `<plnx-proj-root>/images/linux`.

petalinux-boot

The `petalinux-boot` command boots MicroBlaze™ CPU, Zynq® devices, and Zynq® UltraScale+™ MPSoC with PetaLinux images through JTAG/QEMU. This tool provides two distinct workflows:

- In `petalinux-boot --jtag` workflow, images are downloaded and booted on a physical board using a JTAG cable connection.
- In `petalinux-boot --qemu` workflow, images are loaded and booted using the QEMU software emulator.

Either the `--jtag` or the `--qemu` is mandatory for the `petalinux-boot` tool. By default, the `petalinux-boot` tool loads binaries from the `<plnx-proj-root>/images/linux/` directory.

petalinux-boot Command Line Options

The following table details the command line options that are common to all `petalinux-boot` workflows.

Table 9: **petalinux-boot Command Line Options**

Option	Functional Description	Value Range	Default Value
`--jtag`	Use the JTAG workflow. Mutually exclusive with the QEMU workflow. One of the two, `--jtag` or `--qemu` is required.	None	None
`--qemu`	Use the QEMU workflow. Mutually exclusive with the JTAG workflow. One of the two, `--jtag` or `--qemu` is required.	None	None
`--prebuilt`	Boot a prebuilt image. This is optional.	• 1 (bitstream /FSBL) (1) • 2 (U-Boot) • 3 (Linux kernel)	None
`--boot-addr BOOT_ADDR`	Boot address. This is optional.	None	None

Table 9: **petalinux-boot Command Line Options** *(cont'd)*

Option	Functional Description	Value Range	Default Value
`-i,--image IMAGEPATH`	Image to boot. This is optional. To specify U-Boot/Kernel image from an external path, use this option. Example: `$ petalinux-boot --qemu --image ./images/linux/zImage --dtb ./images/linux/system.dtb`	User-specified	None
`--u-boot`	This option can be use to download specified U-Boot binary along with dependent files to boot into the U-Boot. It will select an U-Boot ELF image from `<plnx-root>/images/linux/`. This is optional.	User-specified	`<plnx-projroot>/images/linux/uboot.elf`
`--kernel`	This option can be use to download specified kernel binary along with dependent files to boot kernel. This option will pick kernel image from `<plnx-root>/images/linux/`. This is optional.	User-specified	• zImage for Zynq®-7000 devices • Image for Zynq® UltraScale+™ MPSoC • image.elf for MicroBlaze™ CPU The default image is in `<plnx-projroot>/images/linux`.
`-v,--verbose`	Displays additional output messages. This is optional.	None	None
`-h,--help`	Displays tool usage information. This is optional.	None	None

Notes:

1. `--prebuilt` 1 is not a valid option for the QEMU workflow.

petalinux-boot --jtag

The `petalinux-boot --jtag` command boots the MicroBlaze™ CPUs, the Zynq® UltraScale+™ MPSoCs, or Zynq-7000 devices with a PetaLinux image using a JTAG connection.

Note: The `petalinux-boot --jtag` command might not work as expected when executed within a virtual machine since virtual machines often have problems with JTAG cable drivers.

petalinux-boot --jtag Options

The following table contains details of options specific to the JTAG boot workflow.

Table 10: **petalinux-boot --jtag Options**

Option	Functional Description	Value Range	Default Value
`--xsdb-conn COMMAND`	Customised XSDB connection command to run prior to boot. This is optional.	User-specified	None

Table 10: **petalinux-boot --jtag Options** *(cont'd)*

Option	Functional Description	Value Range	Default Value
`--hw_server-url URL`	URL of the hw_server to connect to. This is optional.	User-specified	None
`--tcl OUTPUTFILE`	Log JTAG Tcl commands used for boot. This is optional.	User-specified	None
`--fpga` [1]	Program FPGA bitstream. This is optional.	None	If no bitstream is specified with the `--bitstream` option, it uses the bitstream from one of the following locations: • If you are using build images to boot, it will pick the bitstream from `<plnx-proj>/images/linux/system.bit` • If you are using prebuilt images to boot, it will pick the bitstream pick `<plnx-proj>/prebuilt/linux/implementation/download.bit`
`--bitstream BITSTREAM`	Specify a bitstream. This is optional.	User-specified	None
`--pmufw PMUFW-ELF`	PMU firmware image. This is optional and applicable for Zynq® UltraScale+™ MPSoC. PMU firmware image is loaded by default, unless it is specified otherwise. To skip loading PMU firmware, use `--pmufw no`.	None	`<plnx-projroot>/images/linux/pmufw.elf`
`before-connect <CMD>`	Extra command to run before XSDB connect command. Ensure the command is properly quoted in your shell. This is optional and can be used multiple times.	None	None
`after-connect <CMD>`	Extra commands to run after XSDB connect command. Ensure the command is properly quoted in your shell. This is optional and can be used multiple times.	None	None

Notes:

1. The `--fpga` option looks for download.bit in `<plnx-proj-root>/pre-built/linux/implementation` by default.

petalinux-boot --jtag Examples

Images for loading on target can be selected from the following:

1. Prebuilt directory: `<PROJECT>/pre-built/linux/images`. These are prebuilt images packed along with the BSP.
2. Images directory: `<PROJECT>/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 `<PROJECT>/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 `<PROJECT>/pre-built/linux/images` directory.

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

  Note: Images are taken from `<PROJECT>/images/linux` directory.

  - For MicroBlaze™ processors, the above commands download the bitstream to the target board, and then boot the U-Boot on the target board.
  - For Zynq-7000 devices, they download the bitstream and FSBL to the target board, and then boot the U-Boot on the target board.
  - For Zynq UltraScale+ MPSoC, they download the bitstream, PMU firmware, and FSBL, and then boot the U-Boot on the target board.

- Boot prebuilt kernel on target board after loading bitstream, boot loader, and U-Boot.

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

  Note: Images are taken from `<PROJECT>/pre-built/linux/images` directory.

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

  Note: Images are taken from `<PROJECT>/images/linux` directory.

  - For MicroBlaze processors, the above commands download the bitstream to the target board, and then boot the kernel image on the target board.
  - For Zynq-7000 devices, they download the bitstream and FSBL to the target board, and then boot the U-Boot and then the kernel on the target board.
  - For Zynq UltraScale+ MPSoC, they download the bitstream, PMU firmware, and FSBL, and then boot the kernel with help of `linux-boot.elf` to set kernel start and DTB addresses.

petalinux-boot --qemu

The `petalinux-boot --qemu` command boots the MicroBlaze™ CPU, Zynq® UltraScale+™ MPSoC, 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.

petalinux-boot --qemu Options

The following table contains details of options specific to the QEMU boot workflow:

Table 11: **petalinux-boot --qemu Options**

Otion	Functional Description	Value Range	Default Value
`--root`	Boot in root mode	None	None
`--dtb DTBFILE`	Use a specified device tree file. This is optional.	User-specified	system.dtb
`--iptables-allowed`	Whether to allow to implement iptables commands. This is optional and applicable only in root mode.	None	None
`--net-intf`	Network interface on the host to bridge with the QEMU subnet. This option applies for root mode only.	User-specified	eth0
`--qemu-args`	Extra arguments to QEMU command. This is optional.	None	None
`--subnet SUBNET`	Specifies subnet gateway IP and the number of valid bit of network mask. This option applies for root mode only.	User-specified	192.168.10.1/24
`--dhcpd`	Enable or disable dhcpd. This is optional and applicable only for root mode.	Enable Disable	Enable
`--tftp`	Path to tftp boot directory	User-specified	None
`--pmu-qemu-args`	Extra arguments for PMU instance of QEMU. This is optional.	User-specified	None

petalinux-boot --qemu Examples

The following examples demonstrate proper usage of the `petalinux-boot --qemu` command.

- Load and boot a prebuilt U-Boot elf using QEMU.

```
$ petalinux-boot --qemu --prebuilt 2
```

- Load and boot a prebuilt U-Boot elf using QEMU in root mode.

```
$ petalinux-boot --qemu --root --prebuilt 2
```

petalinux-package

The `petalinux-package` tool packages a PetaLinux project into a format suitable for deployment. The tool provides several workflows whose operations vary depending on the target package format. The supported formats/workflows are `boot`, `bsp`, and `pre-built`. The `petalinux-package` tool is executed using the package type name to specify a specific workflow in the format `petalinux-package --PACKAGETYPE`.

- The `boot` package type creates a file (.BIN or .MCS) that allows the target device to boot.
- The `bsp` package type creates a .bsp file which includes the entire contents of the target PetaLinux project. This option allows you to export and re-use your bsp.
- The `pre-built` package type creates a new directory within the target PetaLinux project called "pre-built" and contains prebuilt content that is useful for booting directly on a physical board. This package type is commonly used as a precursor for creating a `bsp` package type.
- The `image` package type packages image for component with the specified format.
- The `sysroot` package type installs the sysroot for the Vitis™ software platform. It can specify the installer path and also install directory path.

You are required to install Vivado® Design Suite on the same machine as PetaLinux to use `petalinux-boot` for the MCS format for MicroBlaze™ processor. By default, the `petalinux-package` tool loads default files from the `<plnx-proj-root>/images/linux/` directory.

petalinux-package Command Line Options

The following table details the command line options that are common to all of the `petalinux-package` workflows.

Table 12: **petalinux-package Command Line Options**

Option	Functional Description	Value Range	Default Value
`-p, --project PROJECT`	PetaLinux project directory path. This is optional.	User-specified	Current Directory
`-h, --help`	Display usage information. This is optional.	None	None

petalinux-package --boot

The `petalinux-package --boot` command generates a bootable image that can be used directly with Zynq® UltraScale+™ MPSoC and Zynq-7000 devices, and also with MicroBlaze™-based FPGA designs. For devices in the Zynq series, bootable format is BOOT.BIN which can be booted from an SD card. For MicroBlaze-based designs, the default format is an MCS PROM file suitable for programming using Vivado® Design Suite or other PROM programmer.

For devices in the Zynq series, this workflow is a wrapper around the bootgen utility provided with the Vitis software platform. For MicroBlaze-based FPGA designs, this workflow is a wrapper around the corresponding Vivado Tcl commands and generates an MCS formatted programming file. This MCS file can be programmed directly to a target board and then booted.

petalinux-package --boot Command Options

The following table details the options that are valid when creating a bootable image with the `petalinux-package --boot` command:

Table 13: **petalinux-package --boot Command Options**

Option	Functional Description	Value Range	Default Value
`--format FORMAT`	Image file format to generate. This is optional.	• BIN • MCS • DOWNLOAD.BIT	BIN
`--fsbl FSBL`	Path on disk to FSBL elf binary. This is required. To skip loading FSBL, use `--fsbl no` or `--fsbl none`. This is optional.	User-specified	• zynqmp_fsbl.elf for Zynq® UltraScale+™ MPSoC • zynq_fsbl.elf for Zynq-7000 device • fs-boot.elf for MicroBlaze™ processor The default image is in `<plnx-proj -root>/images/linux`.
`--force`	Overwrite existing files on disk. This is optional.	None	None
`--fpga BITSTREAM`[1]	Path on disk to bitstream file. This is optional.	User-specified	`<project>/images/linux/system.bit`
`--atf ATF-IMG`	Path on disk to Arm® trusted firmware elf binary. This is optional. To skip loading ATF, use `--atf no` or `--atf none`	User-specified	`<plnx-projroot>/images/linux/bl31.elf`
`--u-boot UBOOT-IMG`	Path on disk to U-Boot binary. This is optional.	User-specified	• u-boot.elf for Zynq device • u-boot-s.bin for MicroBlaze CPUs The default image is in `<plnx-proj-root>/images/linux`

Table 13: **petalinux-package --boot Command Options** *(cont'd)*

Option	Functional Description	Value Range	Default Value
`--kernel KERNEL-IMG`	Path on disk to Linux kernel image. This is optional.	User-specified	`<plnx-projroot>/images/linux/image.ub`
`--pmufw PMUFW-ELF`	Optional and applicable only for Zynq® UltraScale+™ MPSoC. By default, prebuilt PMU firmware image is packed. Use this option to either specify a path for PMU firmware image or to skip packing of PMU firmware. To skip packing PMU firmware, use `--pmufw no`.	User-specified	`<plnx-proj-root>/images/linux/pmufw.elf`
`--add DATAFILE`	Path on disk to arbitrary data to include. This is optional.	User-specified	None
`--offset OFFSET`	Offset at which to load the prior data file. Only the `.elf` files are parsed. This is optional.	User-specified	None
`--load <LOADADDR>`	Load address for specified data file. The RAM address where to load the specified data file. Example: `[ partition_type=raw, load=0x01000 ] <image>`	User-specified	None
`--mmi MMIFILE`	Bitstream MMI file, valid for MicroBlaze CPUs only. It will be used to generate the `download.bit` with boot loader in the block RAM. Default will be the MMI file in the same directory as the FPGA bitstream. This is optional	User-specified	MMI in directory with FPGA bitstream
`--flash-size SIZE`	Flash size in MB. Must be a power-of-2. Valid for MicroBlaze processor only. Not needed for parallel flash types. Ensure you just pass digit value to this option. Do not include MB in the value. This is optional.	User-specified	Auto-detect from system configuration. If it is not specified, the default value is 16.
`--flash-intf INTERFACE`	Valid for MicroBlaze processor only. This is optional.	• SERIALx1 • SPIx1 • SPIx2 • SPIx4 • BPIx8 • BPIx16 • SMAPx8 • SMAPx16 • SMAPx32	Auto-detect
`-o, --output OUTPUTFILE`	Path on disk to write output image. This is optional.	User-specified	None
`--cpu DESTINATION CPU`	Zynq UltraScale+ MPSoC only. The destination CPU of the previous data file. This is optional.	• a53-0 • a53-1 • a53-2 • a53-3	None

Table 13: **petalinux-package --boot Command Options** *(cont'd)*

Option	Functional Description	Value Range	Default Value
`--file-attribute DATA File ATTR`	Zynq-7000 or Zynq® UltraScale+™ MPSoC only. Data file file-attribute. This is optional. Example: `petalinux-package --boot --u-boot --kernel images/linux/Image --offset 0x01e40000 --file-attribute partition_owner=uboot --add images/linux/system.dtb --offset 0x3AD1200 --file-attribute partition_owner=uboot --fpga`	User-specified	None
`--bif-attribute ATTRIBUTE`	Zynq-7000 or Zynq® UltraScale+™ MPSoC only. Example: `petalinux-package --boot --bif-attribute fsbl_config --bif-attribute-value a53_x64 --u-boot`	User-specified	None
`--bif-attribute-value VALUE`	Zynq-7000 or Zynq® UltraScale+™ MPSoC only. The value of the attribute specified by `--bif-attribute` argument. This is optional. Example: `petalinux-package --boot --bif-attribute fsbl_config --bif-attribute-value a53_x64 --u-boot`	User-specified	None
`--fsblconfig BIF_FSBL_CONFIG`	Zynq® UltraScale+™ MPSoC only. BIF FSBL config value. Example: `petalinux-package --boot --fsblconfig a53_x64 --u-boot`	User-specified	None
`--bif BIF FILE`	Zynq-7000 or Zynq UltraScale+ MPSoC only. BIF file. For Zynq-7000 devices and Zynq UltraScale + MPSoC, it overrides the following settings: • –fsbl • –fpga • –u-boot • –add • –fsblconfig • –file-attribute • –bif-attribute • –bif-attribute-value This is optional.	User-specified	None
`--boot-device BOOT-DEV`	Zynq-7000 or Zynq UltraScale+ MPSoC only. The boot device is updated in bootargs to boot. This is optional.	• sd • flash	Default value is the one selected from the system select menu of boot image settings.

Table 13: **petalinux-package --boot Command Options** *(cont'd)*

Option	Functional Description	Value Range	Default Value
`--bootgen-extra-args ARGS`	Zynq-7000 or Zynq UltraScale+ MPSoC only. Extra arguments to be passed while invoking bootgen command. This is optional.	User-specified	None

Notes:

1. When the FPGA Manager `petalinux-config` option is enabled, the `--fpga` option cannot be used. Bitstream will not be included in the BOOT.BIN.

petalinux-package --boot Examples

The following examples demonstrate proper usage of the `petalinux-package --boot` command.

- Create a BOOT.BIN file for a Zynq® device (including Zynq-7000 and Zynq® UltraScale+™ MPSoC).

```
$ petalinux-package --boot --format BIN --fsbl <PATH-TO-FSBL> --u-boot -o
<PATH-TO-OUTPUT-WITH-FILE-NAME>
```

- Create a BOOT.BIN file for a Zynq device that includes a PL bitstream and FITimage.

```
$ petalinux-package --boot --format BIN --fsbl <PATH-TO-FSBL> --u-boot --
fpga <PATH-TO-BITSTREAM> --kernel -o <PATH-TO-OUTPUT>
```

- Create a x8 SMAP PROM MCS file for a MicroBlaze™ CPU design.

```
$ petalinux-package --boot --format MCS --fsbl <PATH-TO-FSBL> --u-boot --
fpga <PATH-TO-BITSTREAM> --flash-size <SIZE> --flash-intf SMAPx8 -o
<PATH-TO-OUTPUT-WITH-FILE-NAME>
```

- Create a BOOT.BIN file for a Zynq UltraScale+ MPSoC that includes PMU firmware.

```
$ petalinux-package --boot --u-boot --kernel --pmufw <PATH_TO_PMUFW>
```

- Create bitstream file download.bit for a MicroBlaze CPU design.

```
$ petalinux-package --boot --format DOWNLOAD.BIT --fpga <BITSTREAM> --fsbl
<FSBOOT_ELF>
```

petalinux-package --bsp

The `petalinux-package --bsp` command compiles all contents of the specified PetaLinux project directory into a BSP file with the provided file name. This .bsp file can be distributed and later used as a source for creating a new PetaLinux project. This command is generally used as the last step in producing a project image that can be distributed to other users. All Xilinx® reference BSPs for PetaLinux are packaged using this workflow.

petalinux-package --bsp Command Options

The following table details the options that are valid when packaging a PetaLinux BSP file with the `petalinux-package --bsp` command.

Table 14: **petalinux-package --bsp Command Options**

Option	Functional Description	Value Range	Default Value
-o, --output BSPNAME	Path on disk to store the BSP file. File name is of the form BSPNAME.bsp. This is required.	User-specified	None
-p,--project PROJECT	PetaLinux project directory path. In the BSP context, multiple project areas can be referenced and included in the output BSP file. This is optional.	User-specified	None
--force	Overwrite existing files on disk. This is optional.	None	None
--clean	Clean the hardware implementation results to reduce package size. This is optional.	None	None
--hwsource HWPROJECT	Path to a Vivado® design tools project to include in the BSP file. Vivado® hardware project will be added to the hardware directory of the output BSP. This is optional.	None	None
--exclude-from-file EXCLUDE_FILE	Excludes the files mentioned in EXCLUDE_FILE from BSP.	User-specified	None

petalinux-package --bsp Command Examples

The following examples demonstrate the proper usage of the `petalinux-package --bsp` command.

- Clean the project and then generate the BSP installation image (.bsp file).

```
$ petalinux-package --bsp --clean -o <PATH-TO-BSP> -p <PATH-TO-PROJECT>
```

- Generate the BSP installation image that includes a reference hardware definition.

```
$ petalinux-package --bsp --hwsource <PATH-TO-HW-EXPORT> -o <PATH-TO-BSP>
-p <PATH-TO-PROJECT>
```

- Generate the BSP installation image from a neutral location.

```
$ petalinux-package --bsp -p <PATH-TO-PROJECT> -o <PATH-TO-BSP>
```

- Generate the BSP installation image excluding some files.

```
$ petalinux-package --bsp -p <path_to_project> -o <path_to_bsp> --exclude-
from-file <EXCLUDE_FILE>
```

petalinux-package --prebuilt

The `petalinux-package --prebuilt` command creates a new directory named “pre-built” inside the directory hierarchy of the specified PetaLinux project. This directory contains the required files to facilitate booting a board immediately without completely rebuilding the project. This workflow is intended for those who will later create a PetaLinux BSP file for distribution using the `petalinux-package --bsp` workflow. All Xilinx® reference PetaLinux BSPs contain a prebuilt directory.

petalinux-package --prebuilt Command Options

The following table details the options that are valid when including prebuilt data in the project with the `petalinux-package --prebuilt` workflow.

Table 15: **petalinux-package --prebuilt Command Options**

Options	Functional Description	Value Range	Default Value
`-p,--project PROJECT`	PetaLinux project directory path. This is optional.	User-specified	Current Directory
`--force`	Overwrite existing files on disk. This is optional.	None	None
`--clean`	Remove all files from the `<plnx-proj-root>/prebuilt` directory. This is optional.	None	None
`--fpga BITSTREAM`	Include the BITSTREAM file in the prebuilt directory. This is optional.	User-specified	`<project>/images/linux/*.bit`
`-a,--add src:dest`	Add the file/directory specified by src to the directory specified by dest in the prebuilt directory. This is optional and can be used multiple times.	User-specified	`The default dest path is <project>/prebuilt/linux`

petalinux-package --prebuilt Command Examples

The following examples demonstrate proper usage of the `petalinux-package --prebuilt` command.

- Include a specific bitstream in the prebuilt area.

```
$ petalinux-package --prebuilt --fpga <BITSTREAM>
```

- Include a specific data file in the prebuilt area. For example, add a custom readme to the prebuilt directory.

```
$ petalinux-package --prebuilt -a <Path to readme>:images/<custom readme>
```

petalinux-package --sysroot

The `petalinux-package --sysroot` command installs an SDK to a specified directory in publish mode. This directory can be used as sysroot for application development.

petalinux-package --sysroot Command Options

The following table details the options that are valid when installing an SDK with the `petalinux-package --sysroot` workflow. The SDK must previously have been published using the `petalinux-build --sdk` command.

Table 16: **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 Command Examples

The following command generates partitioned images from the `images/linux` directory. Image generation is driven by partitioning commands contained in the kickstart file (.wks). The default .wks file is FAT32 with 1G and EXT4 with 3 GB. You can find the default kickstart file in `<project-proot>/build/wic/rootfs.wks` after the `petalinux-package --wic` command is executed.

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

Package wic Image using Default Images

The following command generates the wic image, `petalinux-sdimage.wic`, in the `images/linux` folder with the default images from the `images/linux` directory.

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

Package wic Image in a Specific Folder

The following command generates the wic image, `petalinux-sdimage.wic`, in the `wicimage/` folder.

```
$ petalinux-package --wic --outdir wicimage/
```

Package wic Image with Specified Images Path

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

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

Package Custom Bootfiles into the /boot Directory

- To copy `boot.bin userfile1 userfile2` files from the `<project-dir>/images/linux` directory to the `/boot` of media, use the following command:

  ```
  $ petalinux-package --wic --bootfiles "boot.bin userfile1 userfile2"
  ```

  This generates the wic image with specified files copied into the `/boot` directory.

  Note: Ensure that these files are part of the `images` directory.

- To copy the `uImage` file named kernel to the `/boot` directory, use the following command:

  ```
  $ petalinux-package --wic --extra-bootfiles "uImage:kernel"
  ```

- To copy the default bootfiles and specified bootfiles by user files into the `/boot` directory, use the following command:

  ```
  $ petalinux-package --wic --bootfiles "userfiles/*"
  ```

- To copy all the files in the `userfiles/` directory to the `/boot/user_boot` directory, use the following command:

  ```
  $ petalinux-package --wic --extra-bootfiles "userfiles/*:user_boot"
  ```

  Note: Ensure that these files are part of the `images` directory.

Package Custom Root File System

The following command unpacks your `custom-rootfs.tar.gz` file and copies it to the `/rootfs` directory.

```
$ petalinux-package --wic --rootfs-file custom-rootfs.tar.gz
```

Copy the Image SD Card

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

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

petalinux-util

The `petalinux-util` tool provides various support services to the other PetaLinux workflows. The tool itself provides several workflows depending on the support function needed.

petalinux-util --gdb

The `petalinux-util --gdb` command is a wrapper around the standard GNU GDB debugger and simply launches the GDB debugger in the current terminal. Executing `petalinux-util --gdb --help` at the terminal prompt provides verbose GDB options that can be used.

For GDB GUI-based debugging, use the Vitis™ software platform. For more information regarding GDB, see *Vitis Unified Software Platform Documentation: Embedded Software Development* (UG1400).

petalinux-util --gdb command Examples

The following example demonstrates proper usage of the `petalinux-util --gdb` command. To launch the GNU GDB debugger, use the following command:

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

petalinux-util --dfu-util

The `petalinux-util --dfu-util` command is a wrapper around the standard `dfu-util`, and launches `dfu-util` in the current terminal. Executing `petalinux-util --dfu-util --help` at the terminal prompt provides verbose dfu-util options that can be used.

petalinux-util --dfu-util Command Examples

The following example demonstrates proper usage of the `petalinux-util --dfu-util` command. To launch the `dfu-util`, use the following command:

```
$ petalinux-util --dfu-util
```

petalinux-util --xsdb-connect

The `petalinux-util --xsdb-connect` command provides XSDB connection to QEMU. This is for Zynq® UltraScale+™ MPSoC and Zynq-7000 devices only.

For more information regarding XSDB, see *Vitis Unified Software Platform Documentation: Embedded Software Development* (UG1400).

petalinux-util --xsdb-connect Options

The following table details the options that are valid when using the `petalinux-util --xsdb-connect` command.

Table 17: **petalinux-util --xsdb-connect Options**

Option	Functional Description	Value Range	Default Value
`--xsdb-connect HOST:PORT`	Host and the port XSDB should connect to. This should be the host and port that QEMU has opened for GDB connections. It can be found in the QEMU command line arguments from: --gdb tcp: <QEMU_HOST>: <QEMU_PORT>. This is required.	User-specified	None

petalinux-util --jtag-logbuf

The `petalinux-util --jtag-logbuf` command logs the Linux kernel printk output buffer that occurs when booting a Linux kernel image using JTAG. This workflow is intended for debugging the Linux kernel for review and debug. This workflow can be useful when the Linux kernel is not producing output using a serial terminal. For details on how to boot a system using JTAG, see the `petalinux-boot --jtag` command. For MicroBlaze™ CPUs, the image that can be debugged is `<plnx-proj-root>/image/linux/image.elf`. For Arm® cores, the image that can be debugged is `<plnx-proj-root>/image/linux/vmlinux`.

petalinux-util --jtag-logbuf Options

The following table details the options that are valid when using the `petalinux-util --jtag-logbuf` command.

Table 18: **petalinux-util --jtag-logbuf Options**

Option	Functional Description	Value Range	Default Value
`-i,--image IMAGEPATH`	Linux kernel ELF image. This is required.	User-specified	None
`--hw_server-url URL`	URL of the hw_server to connect to. This is optional.	User-specified	None
`-p,--project PROJECT`	PetaLinux project directory path. This is optional.	User-specified	Current Directory
`--noless`	Do not pipe output to the less command. This is optional.	None	None
`-v,--verbose`	Displays additional output messages. This is optional.	None	None
`-h,--help`	Displays tool usage information. This is optional.	None	None
`--dryrun`	Prints the commands required to extract the kernel log buffer, but do not run them.	None	None

petalinux-util --jtag-logbuf Examples

The following examples demonstrate proper usage of the `petalinux-util --jtag-logbuf` command.

- Launch a specific Linux kernel image

```
$ petalinux-util --jtag-logbuf -i <PATH-TO-IMAGE>
```

- Launch the JTAG logger from a neutral location. This workflow is for Zynq®-7000 devices only

```
$ petalinux-util --jtag-logbuf -i <PATH-TO-IMAGE> -p <PATH-TO-PROJECT>
```

petalinux-util --find-hdf-bitstream

The `petalinux-util --find-hdf-bitstream` gives the name of bitstream packed in the hdf bitstream from xsa.

petalinux-util --find-hdf-bitstream Options

The following table details the options that are valid when using the `petalinux-util --find-hdf-bitstream` command.

Table 19: **petalinux-util --find-hdf-bitstream Options**

Option	Functional Description	Value Range	Default Value
`--hdf-file <XSA>`	Argument to specify the XSA file to use. This is optional.	None	system.xsa file in the `<project>/project-spec/hw-description` directory

petalinux-util --find-hdf-bitstream Examples

The following examples demonstrate proper usage of the `petalinux-util --find-hdf-bitstream` command.

- To find the default bitstream of a project

```
petalinux-util --find-hdf-bitstream
```

- To find the bitstream of a xsa

```
petalinux-util --find-hdf-bitstream --hdf-file <path to xsa file>
```

petalinux-util --webtalk

The `petalinux-util --webtalk` command sets the Xilinx® WebTalk feature ON or OFF. Xilinx WebTalk provides anonymous usage data about the various PetaLinux tools to Xilinx. A working internet connection is required for this feature to work when enabled.

petalinux-util --webtalk Options

The following table details the options that are valid when using the `petalinux-util --webtalk` command.

Table 20: **petalinux-util --webtalk Options**

Option	Functional Description	Value Range	Default Value
`--webtalk`	Toggle WebTalk. This is required.	• On • Off	Off
`-h,--help`	Display usage information. This is optional.	None	None

petalinux-util --webtalk Examples

The following examples demonstrate proper usage of the `petalinux-util --webtalk` command.

- Turn the WebTalk feature off

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

- Turn the WebTalk feature on

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

petalinux-upgrade

To upgrade the workspace, use the `petalinux-upgrade` command.

petalinux-upgrade Options

Table 21: **petalinux-upgrade Options**

Options	Functional description	Value Range	Default Range
`-h --help`	Displays usage information.	None	None
`-f --file`	Local path to target system software components.	User-specified.	None
`-u --url`	URL to target system software components.	User-specified.	None
`-w, --wget-args`	Passes additional wget arguments to the command.	Additional wget options.	None
`-p|--platform`	Specifies the architecture name to upgrade.	aarch64: sources for Zynq UltraScale+ MPSoC arm: sources for Zynq devices microblaze_lite: sources for microblaze_lite microblaze_full: sources for microblaze_full	None

Upgrading Between Minor Releases (2020.1 Tool with 2020.2 Tool)

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

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

Upgrade PetaLinux Tool

Upgrade from Local File

Download the target system software components content from the server URL http://petalinux.xilinx.com/sswreleases/rel-v2020/sdkupdate.

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

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

   ***Note*:** Ensure the install area is writable.

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

Example:

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

Upgrade from Remote Server

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

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

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

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

Example:

```
petalinux-upgrade -u “http://petalinux.xilinx.com/sswreleases/rel-v2020/
sdkupdate/”
```

☆ **IMPORTANT!** *Only minor version upgrades are supported.*

Upgrading only Preferred Platforms in Tool

- **To upgrade all platforms:**

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

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

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

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

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

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

- **To upgrade eSDKs for microblaze_lite:**

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

Upgrade PetaLinux Project

Upgrade an Existing Project with the Upgraded Tool

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

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

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

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

Upgrading the Installed Tool with More Platforms

Initially you installed PetaLinux tool with only the arm platform. To install the aarch64 platform, follow these steps.

1. Go to the installed tool.
2. Source `settings.sh` file.
3. Run: `petalinux-upgrade -u http://petalinux.xilinx.com/sswreleases/rel-v2020/sdkupdate/ -p aarch64`

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

Use Cases

- To get the Zynq platform only:

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

- To get Zynq and Zynq UltraScale+ MPSoC platforms:

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

- To get the MicroBlaze platforms:

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

Upgrading the Installed Tool with your Customized Platform

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

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

   The tool will be upgraded with your new platform changes.

***Note*:** These procedures work only between minor releases.

Appendix A

Additional Resources and Legal Notices

Xilinx Resources

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

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the Vivado® IDE, select **Help → Documentation and Tutorials**.
- On Windows, select **Start → All Programs → Xilinx Design Tools → DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

***Note*:** For more information on DocNav, see the Documentation Navigator page on the Xilinx website.

References

These documents provide supplemental material useful with this guide:

1. *PetaLinux Tools Documentation: Reference Guide* (UG1144)
2. Xilinx Answer 55776
3. *Vitis Unified Software Platform Documentation: Embedded Software Development* (UG1400)

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

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