PetaLinux Tools Documentation

PetaLinux Command Line Reference

UG1157 (v2018.3) December 05, 2018

XILINX

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
12/05/2018 Version 2018.3	
General updates	• Modified Table 1-4, Table 1-5, Table 1-6, Table 1-9, Table 1-11, Table 1-13.
petalinux-boot	• Added usecases for petalinux-boot --jtag.
petalinux-package	• Added new option DOWNLOAD.BIT in petalinux-package -boot.
06/06/2018 Version 2018.2	
General updates	• Editorial updates only. No technical content updates.
04/18/2018 Version 2018.1	
General updates	• Modified Table 1-4
petalinux-build	• Added new options to the petalinux-build command line options
petalinux-boot	• Added new options to the petalinux-boot command line options
petalinux-package --sysroot	• Added a new section

Table of Contents

XILINX

Chapter 1

PetaLinux Tools

Introduction

PetaLinux is a development and build environment which automates many of the tasks required to boot embedded Linux on Xilinx SoC's and FPGA's. It uses 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 six independent tools that make up the PetaLinux design flow. They are:

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

In most cases, the individual PetaLinux tools are flexible such that the specific options passed to the tools present you with a unique usage model, compared to other options for the same tool.

For the purposes of this document, command line arguments that behave as a modifier for a workflow are referred to as "options". When options can accept user-specified values, these values are shown in italics. In some cases, omitting the user-specified value may result in a built-in default behavior. See the "Default Value" column in the tables for details about relevant default values.

Design Flow Overview

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

Table 1-1: **Design Flow Overview**

Design Flow Step	Tool / Workflow
Hardware Platform Creation	Vivado
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 Filesystem	`petalinux-config -c rootfs`
Build the System	`petalinux-build`
Test the System on qemu	`petalinux-boot --qemu`
Deploy the System	`petalinux-package --boot`

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.

Table 1-2 details the command line options that are common to all `petalinux-create` workflows.

Table 1-2: **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	None
`-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
`-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, then 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, this may cause build errors.

Options

Table 1-3 details options used specifically when creating a project. One of the below options is mandatory with the `petalinux-create` command.

Table 1-3: **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 • zynq • zynqMP	None
`-s,--source SOURCE`	Create project based on specified BSP file. SOURCE is the full path on disk to the BSP file. This is optional.	User-specified	None

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™ template.

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

By default, the directory structure created by this command with 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.

Options

The `petalinux-create -t apps` command allows you to customize how application components are initialized during creation. Table 1-4 details options that are common when creating applications within a PetaLinux project.

Table 1-4: **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 • install, for application which has prebuilt binary only.	c
`--enable`	Upon creating the component, automatically enable it in the projects' root filesystem. You can also enable using the `petalinux-config -c rootfs`. This is optional.	None	Disabled

Examples

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

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

```
$ petalinux-create -t apps -n <NAME> --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 module and enable it

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

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.

Table 1-5 details the available options for the `petalinux-config` tool.

Table 1-5: **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 working directory
`--get-hw-description <DIR containing HDF/DSA>`	Initializes or updates the hardware configuration for the PetaLinux project. Mutually exclusive with -c. This is required.	User-specified	None
`-c,--component COMPONENT`	Configures the specified system component. Mutually exclusive with `--get-hw-description`. This is required.	• kernel • rootfs • u-boot • bootloader • pmufw, for ZynqMP only • device-tree	None
`--defconfig DEFCONFIG`	Valid for Linux kernel and u-boot. Use the specified defconfig file to initialize the Linux kernel/u-boot configuration. This is optional.	User-specified. For example, for Linux kernel, the file name of a file in `<kernel_source>/arch/<ARCH>/configs/ XXX_defconfig`. For u-boot, the file name of a file in `<uboot_source> /configs`.	None
`--oldconfig`	Parse the config file on Kconfig and generate a new config file. The idea of it is to pull in dependencies if there are any.	None	None

Table 1-5: **petalinux-config Command Line Options *(Cont'd)***

Option	Functional Description	Value Range	Default Value
`-v,--verbose`	Displays additional output messages. This is optional.	None	None
`-h,--help`	Displays tool usage information. This is optional.	None	None

Note: In the previous PetaLinux releases (prior to 2016.3), `petalinux-config` generated the source code for all the embedded software applications such as fsbl, device-tree, pmufw and fs-boot. For 2016.4 and later releases, the source code can be generated explicitly with -c option, if required. Otherwise, it is automatically generated when `petalinux-build` is executed.

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 hardware project. The components affected by this process may 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 HDF/DSA file rather than the full path to the HDF/DSA file itself. This entire option can be omitted if run from the directory that contains the HDF file.

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 HDF/DSA.

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

- Initialize a PetaLinux project from within the directory containing an HDF/DSA.

```
$ 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-HDF/DSA> -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 embeddedSW apps. 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 filesystem, respectively.

The `--oldconfig` option allows you to parse the modified config file onto the Kconfig, so that it can pull the dependencies.

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 --oldconfig
  ```

- 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
  ```

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 via the `petalinux-build -c` and `petalinux-build -x` options.

Table 1-6 outlines the valid options for the `petalinux-build` tool.

Table 1-6: **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
`-c, --component COMPONENT`	Build 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 • kernel • u-boot • rootfs • pmufw, only for ZynqMP • arm-trusted-firmware, only for ZynqMP • device-tree	None
`-x, --execute STEP`(1)	Execute 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	None
`-v,--verbose`	Displays additional output messages. This is optional.	None	None
`-s, --sdk`	Builds Yocto e-SDK. This is optional.	None	None
`-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

Table 1-6: **petalinux-build Command Line Options *(Cont'd)***

Option	Functional Description	Value Range	Default Value
`-f, --force`	Force runs a specific task against a component, or a single task in the component, ignoring the stamps. This is optional.	None	None

Notes:
1. Only a subset of the available tasks are mentioned in the value range. List of available tasks can be found by using the `listtasks` switch.

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 filesystem can be built by targeting those components by name (For example, with `-c <APP-NAME>`). This is equivalent to bitbake <COMPONENT>.

Options

Table 1-7 summarizes the available components that can be targeted with this command.

Table 1-7: **petalinux-build -c Components**

Component	Equivalent Bitbake Commands	Description
`bootloader`	`bitbake virtual/fsbl`	Build only the bootloader elf image and copy it into `<plnx-proj-root>/images/linux/`. For Zynq® and Zynq UltraScale+™ MPSoC devices it is FSBL and for MicroBlaze CPUs 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`

Table 1-7: **petalinux-build -c Components *(Cont'd)***

Component	Equivalent Bitbake Commands	Description
`rootfs`	`bitbake petalinux-user-image -c do_image_complete`	Build only the root filesystem. 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`

The `petalinux-build` command runs `bitbake petalinux-user-image` internally. The default image target is `petalinux-user-image`. There is no restriction on the components, you can build your own packages. For the names of the packages, please search in `petalinux-config -c rootfs`.

Example to build base-files:

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

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. All Yocto tasks can be passed through this option. To get all tasks of a component, use `listtasks`.

Options

Table 1-8 summarizes some of the available commands that can be used with this option.

Table 1-8: **petalinux-build -x Components**

Component	Description
`clean`	Cleans build data for the target component.
`cleansstate`	This removes the sstate cache of the corresponding component.
`distclean`	This removes the sstate cache of the corresponding component.
`cleanall`	This removes the downloads, sstate cache and cleans the work directory of a component.

Table 1-8: **petalinux-build -x Components *(Cont'd)***

Component	Description
`mrproper`	Cleans the build area. This removes the `<plnx-proj-root>/build/`, <TMPDIR> and `<plnx-proj-root>/images/` directories. This the recommended way of cleaning the entire project.
`build`	Build the target component.
`install`	Install 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 rootfs and rootfs component, it copies the generated binary to target rootfs host copy `${TMPDIR}/work/${MACHINE}/petalinux-user-image/1.0-r0/rootfs/.`
`package`	Valid for `-c all` or when no component is specified only. Generate FIT image `image.ub` from build area and copy into `<plnx-proj-root>/images/linux/`.
`listtasks`	To get all tasks of a specific component use listtasks.

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 collaterals from the U-Boot component of the PetaLinux project.

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

- Clean all build collaterals. It removes build/, ${TMPDIR} and images. This will bring 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

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

- Compile a kernel forcefully

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

petalinux-boot

The `petalinux-boot` command boots MicroBlaze, Zynq and Zynq UltraScale+ systems with the 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 via a JTAG cable connection. In `petalinux-boot --qemu` workflow, images are loaded and booted via 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.

Table 1-9 details the command line options that are common to all `petalinux-boot` workflows.

Table 1-9: **petalinux-boot Command Line Options**

Option	Functional Description	Value Range	Default Value
`--jtag`	Use the JTAG workflow. Mutually exclusive with the QEMU workflow. This is required.	None	None
`--qemu`	Use the QEMU workflow. Mutually exclusive with the JTAG workflow. This is required.	None	None
`--prebuilt`	Boot a prebuilt image. This is optional.	• 1 (bitstream /FSBL)(1) • 2 (U-Boot) • 3 (Linux Kernel)	None
`--boot-addr, BOOT_ADDR`	Boot address. This is optional.	None	None
`-i,--image IMAGEPATH`	Image to boot. This is optional.	User-specified	None
`--u-boot`	Specify U-Boot elf binary. Optionally, you can specify u-boot binary path.This option can be use to download specified u-boot binary along with dependent files to boot till u-boot. This is optional.	User-specified	`<plnx-projroot>/images/ linux/uboot.elf`

Table 1-9: **petalinux-boot Command Line Options *(Cont'd)***

Option	Functional Description	Value Range	Default Value
`--kernel`	Specify Linux kernel binary. Optionally, you can specify kernel binary path. This option can be use to download specified kernel binary along with dependent files to boot kernel. This is optional.	User-specified	• zImage for Zynq-7000 • Image for Zynq UltraScale+ MPSoC • `image.elf` for MicroBlaze. 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 or Zynq®-7000 or Zynq UltraScale+ MPSoC system with a PetaLinux image via a JTAG connection.

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

Options

Table 1-10 contains details of options specific to the JTAG boot workflow.

Table 1-10: **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
`--hw_server-url URL`	URL of the `hw_server` to connect to. This is optional.	User-specified	None
`--tcl OUTPUTFILE`	Log JTAG Tcl commands used for boot. This is optional.	User-specified	None

Table 1-10: **petalinux-boot --jtag Options *(Cont'd)***

Option	Functional Description	Value Range	Default Value
`--fpga`[1]	Program FPGA bitstream. This is optional.	User-specified	If no bitstream is specified with the `--bitstream` option, it uses the bitstream found in `<plnxproj-root>/images/linux` directory.
`--bitstream BITSTREAM`	Specify a bitstream. This is optional.	User-specified	None
`--pmufw PMUFW-ELF`	PMU Firmware image. This is optional and applicable for ZynqMP. PMU Firmware image is loaded by default, unless it is specified otherwise. To skip loading pmufw use "`--pmufw no`".	None	`<plnx-projroot>/images/linux/pmufw.elf`
`before-connect <CMD>`	Extra command to run before XSDB connect command. This is optional and can be used multiple times.	None	None
`after-connect <CMD>`	Extra commands to run after XSDB connect command. 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.

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 additionally FSBL for Zynq, FSBL and PMUFW for ZynqMP boards:

  ```
  $ petalinux-boot --jtag --prebuilt 1
  ```

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

- Boot U-Boot on target board:

  ```
  $ 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, the above commands will download the bitstream to target board, and then boot the U-Boot on target board.
  - For Zynq, it will download the bitstream and FSBL to target board, and then boot the U-Boot on target board.
  - For Zynq UltraScale+, it will download the bitstream, PMUFW and FSBL, and then boot the U-Boot on target board.

- Boot prebuilt kernel on target board:

  ```
  $ 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, it will download the bitstream to target board, and then boot the kernel image on target board.
  - For Zynq, it will download the bitstream and FSBL to target board, and then boot the U-Boot and then the kernel on target board.
  - For Zynq UltraScale+, it will download the bitstream, PMUFW and FSBL, and then boot the kernel with help of `linux-boot.elf` to set kernel start and DTB addresses.

- Boot prebuilt kernel on target board:

  ```
  $ 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, it will download the bitstream to target board, and then boot the kernel image on target board.
  - For Zynq, it will download the bitstream and FSBL to target board, and then boot the U-Boot and then the kernel on target board.
  - For Zynq UltraScale+, it will download the bitstream, PMUFW and FSBL, and then boot the kernel with help of `linux-boot.elf` to set kernel start and DTB addresses.

- Generate XSDB tcl using `petalinux-boot` command:

  ```
  $ petalinux-boot --jtag --kernel --tcl mytcl
  ```

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

  This is similar to previous use-case, but instead of loading images on target a tcl (mytcl) is generated. This script can be modified further by users and used directly with XSDB to load images. For example, xsdb mytcl.

- Generate debug messages while loading images:

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

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

  This is similar to previous use-case of loading kernel onto target. Through this use-case additionally user would be able to see debug information from XSDB.

petalinux-boot --qemu

The `petalinux-boot --qemu` command boots the MicroBlaze or Zynq-7000 or Zynq UltraScale+ MPSoC system with a PetaLinux image via 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.

Options

Table 1-11 contains details of options specific to the QEMU boot workflow.

Table 1-11: **petalinux-boot --qemu Options**

Option	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

Examples

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

- Load and boot a pre-built U-Boot elf via QEMU.

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

- Load and boot a pre-built U-Boot elf via 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 operation varies 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.
- The `pre-built` package type creates a new directory within the target PetaLinux project called "pre-built" and contains pre-built 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 SDK. It can specify the SDK installer path and also install directory path.

You are required to install Vivado to use `petalinux-boot` for the MCS format for MicroBlaze architecture. By default, the `petalinux-package` tool loads default files from the `<plnx-proj-root>/images/linux/` directory.

Table 1-12 details the command line options that are common to all of the `petalinux-package` workflows.

Table 1-12: **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 a Zynq family device (including both Zynq-7000 and Zynq UltraScale+ MPSoC) or MicroBlaze-based FPGA design. For Zynq family devices, 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 via Vivado or other PROM programmer.

For Zynq family devices, this workflow is a wrapper around the bootgen utility provided with Xilinx SDK. 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.

Options

Table 1-13 details the options that are valid when creating a bootable image with the `petalinux-package --boot` command.

Table 1-13: **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 • `fs-boot.elf` for MicroBlaze. The default image is in `<plnx-proj-root>/images/linux`.
`--force`	Overwrite existing files on disk. This is optional.	None	None
`--fpga BITSTREAM`	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. It is U-Boot ELF for Zynq family device and `u-boot-s.bin` for MicroBlaze. This is optional.	User-specified	• `u-boot.elf` for Zynq family device • `u-boot-s.bin` for MicroBlaze. The default image is in `<plnx-proj-root>/images/linux`
`--kernel KERNEL-IMG`	Path on disk to Linux Kernel image. This is optional.	User-specified	`<plnx-projroot>/images/linux/image.ub`

Table 1-13: **petalinux-package --boot Command Options *(Cont'd)***

Option	Functional Description	Value Range	Default Value
`--pmufw PMUFW-ELF`	Optional and applicable only for Zynq UltraScale+ MPSoC. By default, pre-built pmufw image is packed. Use this option to either specify a path for pmufw image or to skip packing of pmufw. To skip packing pmufw 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
`--mmi MMIFILE`	Valid for MicroBlaze Only. 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 only. Not needed for parallel flash types. Ensure you just pass digit value to this option. Please 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 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	Current Directory
`--cpu DESTINATION CPU`	Zynq UltraScale+ MPSoC only. The destination CPU of the data file. This is optional.	a53-0 a53-1 a53-2 a53-3	None
`--file-attribute DATA File ATTR`	Zynq-7000 or Zynq UltraScale+ MPSoC only. Data file file-attribute. This is optional.	User-specified	None

Table 1-13: **petalinux-package --boot Command Options *(Cont'd)***

Option	Functional Description	Value Range	Default Value
`--bif-attribute ATTRIBUTE`	Zynq-7000 or Zynq Ultrascale +MPSoC Only. Name of BIF attribute.	User-specified	None
`--bif-attribute-value VALUE`	Zynq-7000 or Zynq UltraScale+ MPSoC only. The value of the attribute specified by `--file-attribute` argument. This is optional.	User-specified	None
`--fsblconfig BIF FSBL CONFIG`	Zynq Ultrascale+MPSoC only. BIF FSBL config value.	User-specified	None
`--bif BIF FILE`	Zynq-7000 or Zynq UltraScale+ MPSoC only. BIF file. It overrides all other 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. This is optional.	• sd • flash	Default value is the one selected from the system select menu of boot image settings.
`--bootgen-extra-args ARGS`	Zynq-7000 or Zynq Ultrascale+ MPSoC only. Extra arguments to be passed while invoking bootgen command. This is optional.	User-specified	None

Examples

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

- Create a `BOOT.BIN` file for a Zynq family 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 family device that includes a PL bitstream and FIT image.

```
$ 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 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 device that includes a PMU firmware.

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

- Create bitstream file download.bit for a Microblaze design

```
$ petalinux-package --boot --format DOWNLOAD.BIT --fpga <BITSTREAM> --fsbl
<FSBL_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 BSP's for PetaLinux are packaged using this workflow.

Options

Table 1-14 details the options that are valid when packaging a PetaLinux BSP file with the `petalinux-package --bsp` command.

Table 1-14: **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	Current Directory
`-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	Current Directory
`--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

Table 1-14: **petalinux-package --bsp Command Options *(Cont'd)***

Option	Functional Description	Value Range	Default Value
`--hwsource HWPROJECT`	Path to a Vivado project to include in the BSP file. This is optional.	None	None
`--exclude-from-file EXCLUDE_FILE`	Excludes the files mentioned in EXCLUDE_FILE from bsp.	User-specified	None

Examples

The following examples demonstrate 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 --image

The `petalinux-package --image` command packages an image for a component. You can use it to generate uImage for kernel.

Options

Table 1-15 details the options that are valid when packaging an image with the `petalinux-package -- image` workflow.

Table 1-15: **PetaLinux-package --image Command Options**

Option	Functional Description	Value Range	Default Value
`-p,--project PROJECT`	PetaLinux project directory path. This is optional.	User-specified	Current Directory
`-c,--component COMPONENT`	PetaLinux project component. This is optional.	User-specified	• kernel
`--format FORMAT`	Image format. It relies on the component. This is optional.	User-specified	kernel: • uImage • Image for Zynq UltraScale+ MPSoC • zImage for Zynq-7000

Example

The following example demonstrate proper usage of the `petalinux-package --image` command.

- Generate uImage.

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

  The uImage is in `<plnx-proj-root>/images/linux` directory.

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 BSP's contain a pre-built directory.

Options

Table 1-16 details the options that are valid when including pre-built data in the project with the `petalinux-package --prebuilt` workflow.

Table 1-16: **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 pre-built directory. This is optional and can be used multiple times.	User-specified	None

Examples

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

- Include a specific bitstream in the pre-built area.

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

- Include a specific data file in the pre-built 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 sdk to a specified directory in publish mode. This directory can be used as sysroot for application development.

Options

Table 1-16 details the options that are valid when installing an SDK with the `petalinux-package --sysroot` workflow.

Table 1-17: **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

Examples

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

- Install default SDK to default directory

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

- Install specified sdk to default directoy

```
petalinux-package --sysroot -s <PATH-TO-SDK>
```

- Install specified sdk to specified sdk to specified directory

```
petalinux-package --sysroot -s <PATH-to-SDK> -d <PATH-TO-INSTALL-DIR>
```

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 Xilinx SDK. For more information regarding GDB, see Using Xilinx SDK.

Example

The following example demonstrates proper usage of the `petalinux-util --gdb` command.

- Launch the GNU GDB debugger.

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

petalinux-util --dfu-util

The `petalinux-util --dfu-util` command is a wrapper around the standard `dfu-util`, and simply launches the `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.

Example

The following example demonstrates proper usage of the `petalinux-util --dfu-util` command.

- Launch the `dfu-util`

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

petalinux-util --xsdb-connect

The `petalinux-util --xsdb-connect` command provides XSDB connection to QEMU, this is for Zynq-7000 and Zynq UltraScale+ MPSoC only.

For more information regarding XSDB, see Using Xilinx SDK.

Options

Table 1-18 details the options that are valid when using the `petalinux-util --xsdb-connect` command.

Table 1-18: **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 via JTAG. This workflow is intended for debugging the Linux kernel for review and debug. This workflow may be useful for users when the Linux kernel is not producing output via a serial terminal. For details on how to boot a system via JTAG, see the petalinux-boot --jtag command. For MicroBlaze, the image is `<plnx-proj-root>/image/linux/image.elf`. For Arm, the image is `<plnx-proj-root>/image/linux/vmlinux`.

Options

Table 1-19 details the options that are valid when using the `petalinux-util --jtag-logbuf` command.

Table 1-19: **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

Table 1-19: **petalinux-util --jtag-logbuf Options *(Cont'd)***

Option	Functional Description	Value Range	Default Value
`-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

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` extracts bitstream from hdf.

Options

Table 1-20 details the options that are valid when using the `petalinux-util --find-hdf-bitstream` command.

Table 1-20: **petalinux-util --find-hdf-bitstream Options**

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

Example

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

- To find the name of the default bitstream of a project:

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

- To find the bitstream of a hdf:

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

petalinux-util --webtalk

The `petalinux-util --webtalk` command toggles 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 turn on.

Options

Table 1-21 details the options that are valid when using the `petalinux-util --webtalk` command.

Table 1-21: **petalinux-util --webtalk Options**

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

Examples

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

- Toggle the WebTalk feature off.

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

- Toggle the WebTalk feature on.

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

Appendix A

Additional Resources and Legal Notices

Xilinx Resources

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

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (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 the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

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

References

1. *PetaLinux Tools Documentation (*UG1144*).*
2. Xilinx Answer Record 55776

Please Read: Important Legal Notices