

PetaLinux SDK User Guide

Getting Started Guide

UG977 (v2013.10) November 25, 2013



Notice of Disclaimer

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 the Limited Warranties which can be viewed at <http://www.xilinx.com/warranty.htm>; 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 Critical Applications: <http://www.xilinx.com/warranty.htm#critapps>.

© Copyright 2013 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

Revision History

Date	Version	Notes
2009-11-19	1.1	Initial version for SDK 1.1 release
2011-11-26	1.3	Updated for PetaLinux SDK 1.3 release - PowerPC 440 support
2011-04-01	2.1	Updated for PetaLinux SDK 2.1 release - new procedure for rebuilding reference designs based on Xilinx 13.1
2012-08-03	3.1	Updated for PetaLinux SDK 3.1 release
2012-09-03	12.9	Updated for PetaLinux SDK 12.9 release
2012-12-17	2012.12	Updated for PetaLinux SDK 2012.12 release
2013-04-29	2013.04	Updated for PetaLinux SDK 2013.04 release
2013-11-25	2013.10	Updated for PetaLinux SDK 2013.10 release

Online Updates

Please refer to the PetaLinux v2013.10 Master Answer Record ([Xilinx Answer Record #55776](#)) for the latest updates on PetaLinux SDK usage and documentation.

Table of Contents

Revision History	1
Online Updates	2
Table of Contents	3
About this Guide	4
Prerequisites	4
Environment Setup	5
Test a Pre-built PetaLinux Image	6
Test Pre-Built PetaLinux Image with QEMU	6
Test Pre-Built PetaLinux Image on Hardware	7
Troubleshooting	9
Rebuilding the Reference Design Software Image	10
Compile PetaLinux Reference Design Software	10
Test New Software Image with QEMU	12
Test New Software Image on Hardware	12
Appendix A: IP Address Configuration	13
IP Address Configuration with ifconfig	13
Appendix B: PetaLinux Project Structure	14
Appendix C: PetaLinux SDK tools usage	18
petalinux-create	18
petalinux-config	20
petalinux-build	22
petalinux-boot	24
petalinux-boot with <code>-jtag</code> Option	24
petalinux-boot with <code>-qemu</code> Option	26
petalinux-package	28
petalinux-util	33
Additional Resources	35
References	35

About this Guide

This document provides basic information on how to start working with the PetaLinux SDK. PetaLinux is an Embedded Linux System Development Kit specifically targeting FPGA-based System-on-Chip designs. With PetaLinux, you can:

- Synchronise your hardware platform and software platform in one step
- Easily propagate your user application to MicroBlaze or Zynq Embedded Linux systems
- Test your MicroBlaze or Zynq Linux system in a virtual machine environment using QEMU.

The following sections will get you started with building and booting linux using PetaLinux tools.

Prerequisites

This getting started document assumes that the following prerequisites have been satisfied:

- You have PetaLinux SDK installed on your Linux workstation. If you haven't installed PetaLinux SDK, please refer to *PetaLinux SDK Installation Guide (UG976)* to install it.
- You have created at least one PetaLinux project from a PetaLinux reference BSP. If you haven't, please refer to section BSP Installation Procedure of *PetaLinux SDK Installation Guide (UG976)* to create a project with a PetaLinux reference BSP.
- A serial communication program such as minicom or kermit has been installed; the baud rate of the serial communication program has been set to 115200bps.
- If you wish to work on hardware designs, Xilinx tools and JTAG cable drivers must be installed. Please refer to Xilinx installation documentation and procedures.
- The reader of this document is assumed to have basic Linux knowledge.
- Unless otherwise indicated the PetaLinux tool command must be run from within a project directory ("`<project-root>`").

For some workflows you may need:

- A workstation with tftpd server running.
- A `/tftpboot` directory on your workstation and all users have read/write permissions to it.

Environment Setup

Setup the PetaLinux working environment by running the PetaLinux setup script as follows:

1. Source the set up script. For bash:

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

or for C shell:

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

WARNING:

- Only run one of these scripts - whichever is appropriate for your terminal shell
 - You must run the settings script each time you open a new terminal window or shell. The PetaLinux tools will fail otherwise.
-

2. Verify that the PetaLinux working environment has been set:

```
$ echo $PETALINUX  
/opt/petalinux-v2013.10-final
```

Environment variable "\$PETALINUX" should point to the path to the installed PetaLinux. Your echo output may be different from this example, it depends on where you installed PetaLinux.

Test a Pre-built PetaLinux Image

So far, you have successfully installed PetaLinux, one or more PetaLinux projects are created from PetaLinux reference BSP, and setup the PetaLinux working environment. Now, you can try one of the reference designs shipped with your BSP package. This is achieved with the `petalinux-boot` command, with the `--qemu` option to boot reference designs under software simulation (QEMU) and `--jtag` on a hardware board.

Test Pre-Built PetaLinux Image with QEMU

PetaLinux provides QEMU support such that the PetaLinux software image can be tested in a simulated environment, without any hardware.

To test the PetaLinux reference design with QEMU, follow these steps:

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

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

The `--qemu` option tells `petalinux-boot` to boot QEMU, instead of real hardware via JTAG, and the `--prebuilt 3` boots the linux kernel.

- The `--prebuilt 1` option performs a Level 1 boot, that is, only configure the FPGA.
- Level 2 is FPGA + u-boot.
- Level 3 is FPGA + pre-built Linux image.

You should see the following kernel boot log on the console:

```

INIT: version 2.88 booting
Starting Bootlog daemon: bootlogd.
Creating /dev/flash/* device nodes
Configuring network interfaces... udhcpc (v1.20.2) started
Sending discover...
Sending select for 10.0.2.15...
Lease of 10.0.2.15 obtained, lease time 86400
/etc/udhcpc.d/50default: Adding DNS 10.0.2.3
done.
starting Busybox inet Daemon: inetd... xemacps e000b000.ps7-ethernet: Set clk to 124999998 Hz
xemacps e000b000.ps7-ethernet: link up (1000/FULL)
done.
Starting uWeb server:
INIT: Entering runlevel: 5
Stopping Bootlog daemon: bootlogd.

-----
|  --- \   | |   | |   (-)
| | / / --- | | - - - | |   - - - - - - - - -
|  -- / / - \ | -- | / - ' | |   | | | ' - \ | | | \ \ / /
| | | | -- / | | - | (- | | | | --- | | | | | | | | | | | > <
\ - |   \ --- | \ - | \ - , - \ --- / | - | - | | - | \ - , - | / - \ \ \

PetaLinux v2013.10 (Yocto 1.4) Xilinx-ZC702-2013_3 ttyPS0
Xilinx-ZC702-2013_3 login:

```

Figure 1: Serial console output of successful petalinux - boot prebuilt

2. Login to PetaLinux with the default user name root and password root.



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

Test Pre-Built PetaLinux Image on Hardware

PetaLinux BSPs include pre-built FPGA bitstreams for each reference design, allowing you to quickly boot linux on your hardware. Here are the steps to test a pre-built linux image with hardware:

1. Power off the board.
2. Connect the JTAG port on the board with the JTAG cable to your workstation.
3. Connect the serial port on the board to your workstation.
4. Connect the Ethernet port on the board to the local network via a network switch.
5. For Zynq boards, ensure the mode switches are set to JTAG mode. Refer to the board documentation for details.
6. Power on the board.

7. Open a console on your workstation and then start your preferred serial communication program (e.g. kermit, minicom) with the baud rate set to 115200 on that console.
8. Run the `petalinux - boot` command as follows on your workstation:

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

The `--jtag` option tells `petalinux - boot` to boot on hardware via JTAG, and the `--prebuilt 3` boots the linux kernel. This command will take some time to finish, please wait until you see the shell prompt again on the command console.

The figures below are examples of the messages on the workstation command console and on the serial console:

```
$ petalinux-boot --jtag --prebuilt 3
INFO: The image provided is a zImage and no addition options were provided
INFO: Append dtb - /home/user/Xilinx-ZC702-2013.3/pre-built/linux/images/system.dtb
and other options to boot zImage
INFO: Configuring the FPGA...
INFO: FPGA configuration completed.
INFO: Downloading FSBL
INFO: FSBL download completed.
INFO: Launching XMD for file download and boot.
INFO: This may take a few minutes, depending on the size of your image.
```

Figure 2: Workstation console output for successful `petalinux - boot`

```

INIT: version 2.88 booting
Starting Bootlog daemon: bootlogd.
Creating /dev/flash/* device nodes
Configuring network interfaces... udhcpc (v1.20.2) started
Sending discover...
Sending select for 10.0.2.15...
Lease of 10.0.2.15 obtained, lease time 86400
/etc/udhcpc.d/50default: Adding DNS 10.0.2.3
done.
starting Busybox inet Daemon: inetd... xemacps e000b000.ps7-ethernet: Set clk to 124999998 Hz
xemacps e000b000.ps7-ethernet: link up (1000/FULL)
done.
Starting uWeb server:
INIT: Entering runlevel: 5
Stopping Bootlog daemon: bootlogd.

-----
|  --- \    | |    | |    (-)
| | / / --- | | -  -- - | |    -  - - - -  - - - -
|  -- / / - \ | -- | / - ' | | |    | | | ' - \ | | | | \ \ / /
| | | |  -- / | | - | (- | | | | ---- | | | | | | | | - | | | > <
\ - |    \ --- | \ - | \ - , - \ ---- / | | | - | | | \ - , - | / - \ - \

PetaLinux v2013.10 (Yocto 1.4) Xilinx-ZC702-2013_3 ttyPS0

Xilinx-ZC702-2013_3 login:

```

Figure 3: Serial console output of petalinux - boot

By default, network settings for PetaLinux reference designs are configured using DHCP. The output you see may be slightly different from the above example, depending upon which PetaLinux reference design you test.

- 9. Type user name root and password root on the serial console to log into the PetaLinux system.
- 10. Determine the IP address of the PetaLinux by running ifconfig on the system console.

Troubleshooting

If your local network does not have a DHCP server, the system will fail to acquire an IP address. If so, refer to Appendix A which describes how to manually specify the address.

If the petalinux - boot for jtag command fails, it is typically from a JTAG connectivity failure. Please ensure the board is powered on and your JTAG cable is properly connected. Please refer to the Xilinx JTAG cable and tools documentation for more detailed troubleshooting.

Rebuilding the Reference Design Software Image

So far, you have tested the PetaLinux reference design pre-built software image both with QEMU and on hardware. You can also rebuild the reference design. The following subsections describe how to do it and how to test the resulting image.

Compile PetaLinux Reference Design Software

First of all, let's look at how to rebuild the PetaLinux reference design.

1. Run `petalinux-build` to compile the software images:

```
$ petalinux-build
```

2. The compilation progress will show on the console. Wait until the compilation finishes.



TIP:

- A detailed compilation log will be in "`<project-root>/build/build.log`" file.
-

When the build finishes, the generated images will be within the "`<project-root>/images`" and "`/tftpboot`" directories.

Here is an example of the compilation progress output:

```
INFO: Checking component...
INFO: Generating make files and build linux
INFO: Generating make files for the subcomponents of linux
INFO: Building linux
[INFO ] pre-build linux/rootfs/fwupgrade
[INFO ] pre-build linux/rootfs/peekpoke
[INFO ] pre-build linux/rootfs/uWeb
[INFO ] build system.dtb
[INFO ] build linux/kernel
[INFO ] update linux/u-boot source
[INFO ] generate linux/u-boot configuration files
[INFO ] build linux/u-boot
[INFO ] Setting up stage config
[INFO ] Setting up rootfs config
[INFO ] Updating for armv7a-vfp-neon
[INFO ] Updating package manager
[INFO ] Expanding stagefs
[INFO ] build linux/rootfs/fwupgrade
[INFO ] build linux/rootfs/peekpoke
[INFO ] build linux/rootfs/uWeb
[INFO ] build kernel in-tree modules
[INFO ] modules linux/kernel
[INFO ] post-build linux/rootfs/fwupgrade
[INFO ] post-build linux/rootfs/peekpoke
[INFO ] post-build linux/rootfs/uWeb
[INFO ] pre-install linux/rootfs/fwupgrade
[INFO ] pre-install linux/rootfs/peekpoke
[INFO ] pre-install linux/rootfs/uWeb
[INFO ] install linux/kernel
[INFO ] install linux/u-boot
[INFO ] Setting up rootfs config
[INFO ] Setting up stage config
[INFO ] Updating for armv7a-vfp-neon
[INFO ] Updating package manager
[INFO ] Expanding rootfs
[INFO ] install sys_init
[INFO ] install linux/rootfs/fwupgrade
[INFO ] install linux/rootfs/peekpoke
[INFO ] install linux/rootfs/uWeb
[INFO ] install kernel in-tree modules
[INFO ] modules_install linux/kernel
[INFO ] post-install linux/rootfs/fwupgrade
[INFO ] post-install linux/rootfs/peekpoke
[INFO ] post-install linux/rootfs/uWeb
[INFO ] package rootfs.cpio to /home/user/ZC702/images/linux
[INFO ] Update and install vmlinux image
[INFO ] vmlinux linux/kernel
[INFO ] install linux/kernel
[INFO ] package zImage
[INFO ] zImage linux/kernel
[INFO ] install linux/kernel
[INFO ] package FIT image
```

Figure 4: Compilation progress output

The final kernel image is the "zImage" for Zynq or "image.elf" for MicroBlaze, living in the "<project-root>/images/linux" folder. A copy is also placed in the "/tftpboot" directory or your development workstation, to support network-based kernel boot.

Test New Software Image with QEMU

Now you have successfully rebuilt the software system image, it is time to test it out.

1. Use `petalinux-boot --qemu` command to test the newly built software image:

```
$ petalinux-boot --qemu --image --kernel
```

The system boot messages will be shown on the console where QEMU is running.

2. When you see the login prompt on the QEMU console, login as `root` with password `root`.

TIP:

- To exit QEMU, press `Ctrl+A` together, release and then press `X`
-

Test New Software Image on Hardware

Next, let's test the rebuilt software image on the real hardware. Follow the instructions from the previous Test Pre-built PetaLinux Image on Hardware section, to connect the board, serial and JTAG correctly.

1. Use `petalinux-boot` to program the FPGA with the reference design pre-built bitstream:

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

This command will take a few moments, please wait until you see the shell prompt shows again on the command console.

2. Use `petalinux-boot` to download the built Linux image to the board and boot it:

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

This command will take a few minutes, downloading the entire kernel image over the JTAG link. Please wait until the shell prompt displays again on the serial console.

3. Watch the serial console, you should see the Linux booting messages shown on the serial console.

You can now repeat the previous steps for connecting to the board via the serial console and the network demo.

Appendix A: IP Address Configuration

IP Address Configuration with ifconfig

After the PetaLinux system boots, you can set or change its IP address manually.

1. First determine which network the PetaLinux system is connected to.
 - If you are booting on real hardware, and the board is connected to a local network, the IP address of the system should be in the subnet of the local network.
 - If you are booting in QEMU, the IP address of the system should be in the QEMU subnet. Refer to the *PetaLinux SDK QEMU System Simulation Guide (UG982)* for more details.
2. Use ifconfig command to set the systems IP address on the login console:

```
# ifconfig eth0 <IP>
```

e.g:

```
# ifconfig eth0 192.168.10.10
```

3. Use ifconfig on the system login console again to confirm whether the IP address has been successfully set:

```
# ifconfig
```

You should be able to see the IP has been set to the interface eth0.

Appendix B: PetaLinux Project Structure

This section provides a brief introduction on a PetaLinux project:

Here is an example of a PetaLinux project:

```

<project-root>
|-.petalinux/
|-hw-description/
|-config.project
|-subsystems/
|  |-linux/
|  |  |-config
|  |  |-hw-description/
|  |  |  |-system.dts
|  |  |  |-xparameters.h
|  |  |  |-config.mk
|  |  |-configs/
|  |  |  |-kernel/
|  |  |  |  |-config
|  |  |  |-u-boot/
|  |  |  |  |-petalinux-user-config.h.template
|  |  |  |-rootfs/
|  |  |  |  |-config
|-components/
|  |-apps/
|  |  |-myapp/

```

File/Directory in a PetaLinux Project	Description
"<project-root>/ .petalinux/"	directory to hold tools usage and webtalk data
"<project-root>/hw-description/"	project level hardware description, NOT USED for this release
"<project-root>/config.project"	project configuration file it defines the external components search path and the subsystem in the project
"<project-root>/subsystems/"	subsystems of the project
"<project-root>/subsystems/linux/"	Linux subsystem. This is the only subsystem supported in this release
"<project-root>/subsystems/linux/config"	Linux subsystem configuration file used when building the subsystem
"<project-root>/subsystems/linux/hw-description/"	subsystem hardware description
"<project-root>/subsystems/linux/hw-description/system.dts"	DTS file used when building the subsystem

"<project-root>/subsystems/linux/hw-description/xparameters.h"	U-Boot "xparameter.h" file
"<project-root>/subsystems/linux/hw-description/config.mk"	U-Boot "config.mk" file
"<project-root>/subsystems/linux/configs/"	configuration files of the components of the subsystem
"<project-root>/subsystems/linux/configs/kernel/config"	configuration file used to build the Linux kernel
"<project-root>/subsystems/linux/configs/u-boot/petalinux-user-config.h.template"	file for users to define or undefine U-Boot configuration macros
"<project-root>/subsystems/linux/configs/rootfs/config"	configuration file used to build the rootfs
"<project-root>/components/"	<p>directory for local components. If you don't have local components, this directory is not required.</p> <p>Components created by <code>petalinux-create</code> will be placed into this directory.</p> <p>You can also manually copy components into this directory.</p> <p>Here is the rule to place a local component:</p> <p>"<project-root>/components/<COMPONENT_TYPE>/<COMPONENT>"</p>

When the project is built, two directories will be auto generated:

- "<project-root>/build" for the files generated for build
- "<project-root>/images" for the bootable images

Here is an example:

```

<project-root>
|-.petalinux/
|-hw-description/
|-config.project
|-subsystems/
|  |-linux/
|    |-config
|    |-hw-description/
|    |  |-system.dts
|    |  |-xparameters.h
|    |  |-config.mk
|    |-configs/
|    |  |-kernel/
|    |    |-config
|    |  |-u-boot/
|    |    |-petalinux-user-config.h.template
|    |-rootfs/
|    |  |-config
|-components/
|  |-apps/
|  |  |-myapp/
|-build/
|  |-build.log
|  |-linux/
|    |-rootfs/
|    |  |-targetroot/
|    |  |-stage/
|    |  |-apps/
|    |    |-myapp/
|    |-kernel/
|    |-u-boot/
|-images/
|  |-linux/

```



WARNING: "`<project-root>/build/`" is automatically generated. Don't manually edit. Contents in this directory will get updated when you run `petalinux-config` or `petalinux-build`.

"`<project-root>/images/`" is automatically generated. Files in this directory will get updated when you run `petalinux-build`.

Build Directory in a PetaLinux Project	Description
" <code><project-root>/build/build.log</code> "	logfile of the build
" <code><project-root>/build/linux/</code> "	directory to hold files related to the linux subsystem build
" <code><project-root>/build/linux/rootfs/</code> "	directory to hold files related to the rootfs build
" <code><project-root>/build/linux/rootfs/targetroot/</code> "	target rootfs host copy

"<project-root>/build/linux/rootfs/stage/"	stage directory to hold the libs and header files required to build user apps/libs
"<project-root>/build/linux/kernel/"	directory to hold files related to the kernel build
"<project-root>/build/linux/u-boot/"	directory to hold files related to the u-boot build

Image Directory in a PetaLinux Project	Description
"<project-root>/images/linux/"	directory to hold the bootable images for Linux subsystem

Appendix C: PetaLinux SDK tools usage

This section provides usage information on PetaLinux SDK tools.

petalinux-create

```

petalinux-create          (c) 2005-2013 Xilinx, Inc.

This command creates a new PetaLinux Project or component

Usage:
  petalinux-create [options] -t|--type <TYPE> -n|--name <COMPONENT_NAME>

Required:
  -t, --type <TYPE>          Available type:
                              * project
                              * apps
                              * libs
                              * modules
                              * generic
  -n, --name <COMPONENT_NAME> specify a name for the component or
                              project. It is OPTIONAL to create a
                              PROJECT. If you specify source BSP when
                              you create a project, you are not
                              required to specify the name.

Options:
  -p, --project <PROJECT>    specify full path to a PetaLinux project
                              this option is NOT USED for PROJECT CREATION.
                              default is the working project.
  --force                    force overwriting an existing component
                              directory.
  -h, --help                 show function usage
  --enable                   this option applies to all types except
                              project.
                              enable the created component

Options for project:
  --template <TEMPLATE>      zynq|microblaze
                              default is zynq.
  -s|--source <SOURCE>      specify a PetaLinux BSP as a project
                              source.
  --out <OUTPUT_DIR>        directory to place your project default
                              is your working directory

Options for apps:
  --template <TEMPLATE>      <c|c++|autoconf|install>
                              c   : c user application(default)
                              c++ : c++ user application
                              autoconf: autoconf user application
                              install: install data only
  -s, --source <SOURCE>      valid source name format:
                              XXX.tar.gz, XXX.tar.bz2, XXX.tar,
                              XXX.zip, app source directory

```

```
Options for libs:
--template <TEMPLATE>          <c|c++|autoconf|install-only>
                                c   : c user library(default)
                                c++ : c++ user library
                                autoconf: autoconf user library
--priority                      Library priority (1 to 11, Default is 7)
-s, --source <SOURCE>         valid source name format:
                                XXX.tar.gz, XXX.tar.bz2, XXX.tar,
                                XXX.zip,lib source directory
```

Options for modules: (No specific options for modules)

Options for generic: (No specific options for generic)

Example to create projects:

From PetaLinux Project BSP:

```
$ petalinux-create -t project -s <PATH_TO_PETALINUX_PROJECT_BSP>
```

From template:

```
$ petalinux-create -t project -n <PROJECT> --template zynq
```

Example to create apps:

Create an app and enable it:

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

The application "myapp" will be created with c template in:

```
<PROJECT>/components/apps/myapp/
```

Example to create libs:

Create an lib and enable it:

```
$ petalinux-create -t libs -n mylib --enable
```

The library "mylib" will be created with c template in:

```
<PROJECT>/components/libs/mylib/
```

Example to create modules:

Create an lib and enable it:

```
$ petalinux-create -t modules -n mymodule --enable
```

The library "mymodule" will be created with template in:

```
<PROJECT>/components/modules/mymodule/
```

petalinux-config

```

petalinux-config          (c) 2005-2013 Xilinx, Inc.

INFO: Checking component...
Configures the project or the specified component with menuconfig.

Usage:
  petalinux-config [options] [--component <COMPONENT> |\
  --get-hw-description[=SRC] |--searchpath <--ACTION> [VALUE]]

Options:
  -h, --help                show function usage
  -p, --project <PROJECT>  path to PetaLinux SDK project.
                           default is the working project
  --oldconfig              takes the working configuration
  -c, --component <COMPONENT>
                           Specify the component
                           If no component is specified, it will do
                           top level subsystem configuration only
                           all: to configure the whole project
                           If you specify other component, it will
                           configure that component
                           E.g. -c rootfs
                           If you use ?, it will show you subcomponents
                           E.g. -c ? shows subcomponents of the subsystem
  --get-hw-description[=SRC]
                           get hardware description.
                           if [SRC] is specified, look in that
                           location for an XSDK BSP project.
                           Otherwise, this MUST be run from
                           WITHIN an XSDK PetaLinux BSP project.
  --searchpath              edit project search path

Available project user searchpath actions:
  --prepend <SEARCHPATH>  prepend <SEARCHPATH> to project external searchpath
  --append <SEARCHPATH>   append <SEARCHPATH> to project external searchpath
  --replace <SEARCHPATH>  replace project user searchpath with <SEARCHPATH>
  --print                  print full project searchpath
  --delete                 delete project external searchpath

Available Components of linux for this command:
  * kernel                # is of linux-kernel type
  * rootfs                 # is of rootfs type

Examples to edit searchpath:
  Default PetaLinux tools will look into <PROJECT>/components/ first and then
  ${PETALINUX}/components/ for components
  Prepend external searchpath:
  $ petalinux-config --searchpath --prepend <EXTERN_SEARCHPATH0>
  the components searchpath will become:
  <PROJECT>/components:<EXTERN_SEARCHPATH0>:${PETALINUX}/components/
  Append external searchpath:
  $ petalinux-config --searchpath --append <EXTERN_SEARCHPATH1>
  the components searchpath will become:
  <PROJECT>/components:<EXTERN_SEARCHPATH0>:<EXTERN_SEARCHPATH1>:${PETALINUX}/components/
  Delete external searchpath:
  $ petalinux-config --searchpath --delete
  the components searchpath will become:
  <PROJECT>/components:${PETALINUX}/components/

```

Examples to sync hardware description:

Sync hardware description from XSDK PetaLinux BSP project:

```
$ cd <XSDK_PLNX_BSP>
```

```
$ petalinux-config --get-hw-description
```

It will sync up the DTS, the xparameters.h and the config.mk from
<XSDK_PLNX_BSP> to subsystems/linux/hw-description/ directory.

Sync hardware description inside PetaLinux project but outside XSDK PetaLinux BSP project:

```
$ petalinux-config --get-hw-description=<XSDK_PLNX_BSP>
```

Examples to configure PetaLinux project:

Configure subsystem level configuration:

```
$ petalinux-config
```

Configure kernel:

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

Configure rootfs:

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

petalinux-build

```

petalinux-build          (c) 2005-2013 Xilinx, Inc.

INFO: Checking component...
Builds the project or the specified components.

Usage:
  petalinux-build [options]

Required:

Options:
  -h, --help                show function usage
  -p, --project <PROJECT>  path to PetaLinux SDK project.
                           Default is working project.
  -c, --component <COMPONENT>
                           Specify the component
                           all: to build the whole project
                           If you specify other component, it will
                           build that component
                           E.g. -c rootfs
                           E.g. -c rootfs/myapp
                           If you use ?, it will show you subcomponents
                           E.g. -c rootfs/? shows subcomponents of rootfs
  -x, --execute <GNU_MAKE_TARGET>
                           Specify a GNU make command of the component
  --makeenv <MAKE ENV>    Pass GNU make environment variables
  -v, --verbose            Show compile messages verbose mode

Available Components for linux:
  * kernel      # is of linux-kernel type
  * u-boot      # is of u-boot type
  * rootfs      # is of rootfs type

Available make target for linux:

Quick reference for various supported build targets for linux.
-----
clean          clean out build objects
distclean     clean out build
all           build subsystem and generate final image
build         build subsystem
build_hw-description
build_hw-description
install_hw-description
install       install dtb to subsystem images directory
install       install built objects to target subsystem host copy
package       combine target file system and kernel into final image

Examples:
Build the project:
$ petalinux-build
It is the same as "petalinux-build -c all"
the bootable images are in <PROJECT>/images/linux/.
Build kernel only:
$ petalinux-build -c kernel
Build kernel and update the bootable images:
$ petalinux-build -c kernel
$ petalinux-build -x package

```

```
Build rootfs only:
$ petalinux-build -c rootfs
Build myapp of rootfs only:
$ petalinux-build -c rootfs/myapp
Clean up u-boot and build again:
$ petalinux-build -c u-boot -x distclean
## above command will remove the <PROJECT>/build/linux/u-boot/ directory.
$ petalinux-build -c u-boot
Clean up the project build and build again:
$ petalinux-build -x distclean
## above command will remove the <PROJECT>/build/ directory.
$ petalinux-build
Clean up the project build and the generated bootable images:
$ petalinux-build -x mrproper
## above command will remove <PROJECT>/images/ and <PROJECT>/build/ directories
```


petalinux-boot

petalinux-boot with -jtag Option

```

petalinux-boot          (c) 2005-2013 Xilinx, Inc.

This command boots the MicroBlaze/Zynq systems with Petalinux images
through JTAG/QEMU.
Usage:
  petalinux-boot --qemu|--jtag -c|--component <COMPONENT> [options]
Required:
  --jtag|--qemu          JTAG/QEMU boot mode

Options:
  --prebuilt <BOOT_LEVEL>  Boot prebuilt images (override all settings).
                           supported boot level 1 to 3
                           1 - download FPGA bitstream (and FSBL for Zynq)
                           2 - Boot U-Boot only
                           3 - Boot Linux Kernel only

  --boot-addr <BOOT_ADDR>  boot address
  -i, --image <IMAGE>     image to boot
  --uboot                  boot images/linux/u-boot.elf image
                           if --kernel is specified, --uboot will not take
                           effect.

  --kernel                 boot images/linux/zImage for Zynq
                           boot images/linux/image.elf for MicroBlaze
                           if --kernel is specified, --uboot will not take
                           effect.

  -v, --verbose            output debug messages
  -h|--help                Display help messages

JTAG available options:
  --load-addr <LOADADDR>  address to load the image
  --regdata <REGDATA>     register data
  --extra-xmd "EXTRA_CMD" extra XMD command to run before loading the
                           image, can be repeated
                           E.g. -x "debugconfig -reset_on_run disable"
  --xmd-conn "CONNECT_CMD" customised XMD connect command, can be repeated
                           E.g. --xmd-connect "connect mb mdm"
  --tcl TCL_OUTPUT        dump XMD commands to the specified file
  --targetcpu <TARGET_CPU> specify target CPUID (0 to N-1)
  --fpga                  Programs the hardware with the specified
                           bitstream, If not specified, it will use
                           the pre-built bitstream.

  --bitstream [BITSTREAM] Programs the hardware with the specified
                           bitstream.

Example to download prebuilt bitstream (and FSBL for zynq) to target board:
  $ petalinux-boot --jtag --prebuilt 1
Example to boot prebuilt u-boot on target board:
  $ petalinux-boot --jtag --prebuilt 2
It will download the prebuilt bitstream (and FSBL for zynq) to target board,
and then boot prebuilt u-boot on target board.

```

Example to boot prebuilt kernel on target board:

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

For microblaze, it will download the prebuilt bitstream to target board, and then boot the prebuilt kernel image on target board.

For Zynq, it will download the prebuilt bitstream and FSBL to target board, and then boot the prebuilt u-boot and then the prebuilt kernel on target board.

Example to download a bitstream to target board:

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

Example to download newly built u-boot to target board:

```
$ petalinux-boot --jtag --uboot
```

It will download <PROJECT>/images/linux/u-boot.elf on target board.

Example to download newly built kernel to target board:

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

For MicroBlaze, it will download <PROJECT>/images/linux/image.elf on target board.

For Zynq, it will download <PROJECT>/images/linux/system.dtb and <PROJECT>/images/linux/zImage on target board.

petalinux-boot with `--qemu` Option

```

petalinux-boot          (c) 2005-2013 Xilinx, Inc.

This command boots the MicroBlaze/Zynq systems with Petalinux images
through JTAG/QEMU.
Usage:
  petalinux-boot --qemu|--jtag -c|--component <COMPONENT> [options]
Required:
  --jtag|--qemu          JTAG/QEMU boot mode

Options:
  --prebuilt <BOOT_LEVEL>  Boot prebuilt images (override all settings).
                           supported boot level 1 to 3
                           1 - download FPGA bitstream (and FSBL for Zynq)
                           2 - Boot U-Boot only
                           3 - Boot Linux Kernel only

  --boot-addr <BOOT_ADDR>  boot address
  -i, --image <IMAGE>      image to boot
  --uboot                   boot images/linux/u-boot.elf image
                           if --kernel is specified, --uboot will not take
                           effect.

  --kernel                  boot images/linux/zImage for Zynq
                           boot images/linux/image.elf for MicroBlaze
                           if --kernel is specified, --uboot will not take
                           effect.

  -v, --verbose            output debug messages
  -h|--help                Display help messages

QEMU available options:
  --dtb DTB                force use of a particular device tree file.
                           if not specified, QEMU uses
                           <PROJECT>/images/linux/system.dtb
  --dhcpd enable|disable  enable or disable dhcpd. This option applies
                           for ROOT MODE ONLY.
                           default is to enable dhcpd.
  --iptables-allowed      whether to allow to implement iptables commands.
                           This option applies for ROOT MODE ONLY
                           Default is not allowed.
  --net-intf NET_INTERFACE network interface on the host to bridge with
                           the QEMU subnet. This option applies for ROOT
                           MODE ONLY. Default is eth0.
  --subnet SUBNET          subnet_gateway_ip/num_bits_of_subnet_mask
                           subnet gateway IP and the number of valid bits
                           of network mask. This option applies for ROOT
                           MODE ONLY. Default is 192.168.10.1/24
  --root                   QEMU as root (ROOT MODE).
  --qemu-args "QEMU_ARGUMENTS" extra arguments to QEMU command

Example to boot prebuilt u-boot with QEMU:
  $ petalinux-boot --jtag --prebuilt 2
Example to boot prebuilt kernel with QEMU:
  $ petalinux-boot --jtag --prebuilt 3
Example to download newly built u-boot with QEMU:
  $ petalinux-boot --jtag --uboot
  It will boot <PROJECT>/images/linux/u-boot.elf with QEMU.

```

Example to download newly built kernel to target board:

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

For MicroBlaze, it will boot <PROJECT>/images/linux/image.elf with QEMU.

For Zynq, it will boot <PROJECT>/images/linux/zImage with QEMU.

petalinux-package

```
petalinux-package          (c) 2005-2013 Xilinx, Inc.

This command packages various image format, firmware, prebuilt
and bsp
Usage:
  petalinux-package --boot|--bsp|--firmware|--image|--prebuilt [options]

Required:
  --boot|--bsp|--firmware|--image|--prebuilt
                                     Various package mode.
                                     boot: packages a boot.bin for Zynq
                                     bsp: packages a bsp
                                     firmware: creates a firmware package used
                                     by PetaLinux firmware upgrade demo app to
                                     upgrade firmwares.
                                     image: package various image type
                                     prebuilt: package images to prebuilt

Options:
  -h|--help                          Display help messages
Please specify a package mode option for the detailed options
Show package boot options:
  $ petalinux-package --boot --help
Show package bsp options:
  $ petalinux-package --bsp --help
Show package firmware options:
  $ petalinux-package --firmware --help
Show package image options:
  $ petalinux-package --image --help
Show package prebuilt options:
  $ petalinux-package --prebuilt --help
```

Required option for boot image package:

--fsbl <FSBL_ELF> Path to FSBL ELF image location

Options for boot image package:

--force Force overwrite the boot binary image
--fpga <BITSTREAM> Path to FPGA bitstream image location
--uboot[=<UBOOT_IMG>] Path to the u-boot elf image location
(default <PROJECT>/images/linux/u-boot.elf)
-o, --output <PKGNAME> Generated boot image name
-p, --project <PROJECT> PetaLinux SDK project location.
Default is the working project.

Example to package BOOT.BIN for Zynq:

```
$ petalinux-package --boot --fsbl <FSBL_ELF> --fpga <BITSTREAM> --uboot
```

It will generate a BOOT.BIN in your working directory with:

- * specified <BITSTREAM>
- * specified <FSBL_ELF>
- * newly built u-boot image which is <PROJECT>/images/linux/u-boot.elf

Required options for BSP packaging:

-o, --output <BSP_NAME> BSP package name - <BSP_NAME>.bsp
-p, --project <PROJECT> PetaLinux projects path to be included in
BSP (allow multiple).

Options for BSP packaging:

--force Force overwrite the BSP
--clean Force clean hardware project
--hwsources <PATH_TO_HW> Include a hardware source
--no-extern Exclude external components
If this option is enabled, you can only
rebuild the BSP if the BSP is installed
in machine which can see the external
component search path.
--no-local Exclude local components
If the option is enabled, you may not be
able to build the BSP since the components
placed in your project will not be included
in the BSP. You may want this option if
you don't want to expose your local
components.

Example to package BSP with a PetaLinux project:

```
$ petalinux-package --bsp -p <PATH_TO_PROJECT> --output MY.BSP
```

It will generate MY.BSP including:

- * <PROJECT>/hw-description/
- * <PROJECT>/config.project
- * <PROJECT>/petalinux/
- * <PROJECT>/subsystems/
- * <PROJECT>/pre-built/
- * all selected components

from the specified project.

Example to package BSP with hardware source:

```
$ petalinux-package --bsp -p <PATH_TO_PROJECT> \  
--hwsources <PATH_TO_HARDWARE_PROJECT> --output MY.BSP
```

It will not modify the specified PetaLinux project <PATH_TO_PROJECT>. It will put the specified hardware project source to <PROJECT>/hardware/ inside MY.BSP archive.

Example to package BSP excluding local components:

```
$ petalinux-package --bsp -p <PATH_TO_PROJECT> --output MY.BSP --no-local
```

It will not include any local component in <PROJECT>/components directory, however, it will not change the configuration file, that is, it is possible that you may fail to rebuild the project from MY.BSP.

Example to package BSP excluding extern components:

```
$ petalinux-package --bsp -p <PATH_TO_PROJECT> --output MY.BSP --no-extern
```

It will not include any extern component in user specified components searchpaths. However, it will not change the configuration file, that is, it is possible that you may fail to rebuild the project from MY.BSP.

Required options for firmware packaging:

Options for firmware packaging:

```

-o, --output <PKGNAME>      Output firmware package name
                             (default firmware.tar.gz)
-p, --project <PROJECT>    Path to PetaLinux project.
                             (default current project)
--linux[=<UBIMAGE>]        Update linux kernel image partition with
                             UBIMAGE. (default images/image.ub)
--dtb[=<DTBFILE>]          Update DTB partition with specified DTBFILE
                             (default system.dtb)
--fpga <BITSTREAM>         Update FPGA image partition with bitstream
--uboot[=<UBOOT_S>]        Update u-boot partition with UBOOT_S
                             (default images/u-boot-s.bin)
--bootbin[=<BOOT.BIN>]     Update boot partition with BOOT.BIN
                             (default images/BOOT.BIN) (Arm only)
--jffs2[=<JFFS2IMG>]      Update JFFS2 partition with JFFS2IMG
                             (default images/jffs2.img)
-a, --add dev:file         Update flash partition "dev" with "file", can be repeated.
                             e.g. -a /dev/flash/fpga:<path-to-fpga-bin>
--flash FLASH_TYPE        Flash type(spi, parallel. default is "parallel")
--little-endian            Specify the system is a little endian system.
                             E.g. AXI system is little endian.
                             It can be 8 bits, 16 bits or 32 bits.
--big-endian               Specify the system is a big endian system.
                             E.g. PLB system is big endian.
--data-width <8|16|32>    Specify the data width of the Parallel Flash.
                             Only valid if the target system is little endian.
--product <PRODUCT_STRING> Specify additional compatible product strings
--pre SCRIPT               Run SCRIPT on target prior to firmware upgrade
-v, --verbose              Verbose mode

```

The --image, --dtb, --uboot and --jffs2 options allow to override the default filenames and partitions using the partition:file syntax of the --add option. For example:

Install the image.ub file into the flash partition safe-image:

```
$ petalinux-package --firmware -a /dev/flash/safe-image:/path/to/image.ub
```

Install the uboot-s.bin file into the flash partition safe-boot:

```
$ petalinux-package --firmware -a /dev/flash/safe-boot:/path/to/u-boot-s.bin
```

Example to package firmware with BOOT.BIN and kernel image for Zynq:

```
$ petalinux-package --firmware --bootbin=<BOOT_BIN> --linux
It will create firmware.tar.gz archive in your working directory
including the specified <BOOT_BIN> and <PROJECT>/images/linux/image.ub.
```

Example to package firmware with bistream, u-boot and kernel image for microBlaze:

```
$ petalinux-package --firmware --fpga <BITSTREAM> --uboot --linux
It will create firmware.tar.gz archive in your working directory
including:
* specified <BITSTREAM>
* <PROJECT>/images/linux/u-boot-s.bin
* <PROJECT>/images/linux/image.ub
```


Options for prebuilt:

```
-p, --project <PROJECT>      PetaLinux SDK project.  
                               Default is the working project.  
--force                       Force update the pre-built folder  
--clean                       Clean pre-built directory (remove any files).  
--fpga <BITSTREAM>          FPGA bitstream  
-a, --add src:dest           Add file/folder to prebuilt directory  
                               "src" with "dest"
```

Example to package prebuilt images:

```
$ petalinux-package --prebuilt
```

It will create a pre-built/ directory in <PROJECT>/, and copy the following files from <PROJECT>/images to <PROJECT>/pre-built/linux/images/ directory:

```
* images.ub  
* system.dtb  
* u-boot.elf  
* image.elf  
* System.map.linux  
* u-boot-s.bin  
* zImage (For zynq only)  
* zynq_fsbl.elf (If it is there for zynq only)  
* BOOT.BIN (If it is there for zynq only)
```

Example to package prebuilt images and specified bitstream:

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

Besides copying the images, it will copy the bitstream to <PROJECT>/pre-built/linux/implentation/

Example to package prebuilt images and add myfile to prebuilt:

```
$ petalinux-package --prebuilt --prebuilt -a myfile:images/myfile
```

Besides copying the images, it will copy myfile to <PROJECT>/pre-built/linux/images/myfile

petalinux-util

```

petalinux-util          (c) 2005-2013 Xilinx, Inc.

This command provides the misc utilities.
Usage:
  petalinux-util --gdb | --jtag-logbuf | --update-sdcard |--webtalk <on|off> [options]

Required:
  --gdb | --jtag-logbuf | --update-sdcard |--webtalk
      Various utilities.
      gdb: petalinux gdb debug wrapper
      jtag-logbuf: prints kernel early printk
                  with JTAG.
      update-sdcard: updates the contents of
                    SD card.
      webtalk: Enable or disable webtalk

Options:
  -h|--help          Display help messages
Please specify a package mode option for the detailed options
Show gdb util options:
  $ petalinux-util --gdb --help
Show jtag-logbuf util options:
  $ petalinux-util --jtag-logbuf --help
Show update-sdcard util options:
  $ petalinux-util --update-sdcard --help
Show webtalk util options:
  $ petalinux-util --webtalk --help

Required options for JTAG logbuf:
  -i, --image <IMAGE>      kernel ELF image
                          For MicroBlaze/ARM - PetaLinux image
                          (vmlinux or image.elf).

Available options for JTAG logbuf:
  -p, --project            specify a PetaLinux project (ARM only)
  -v, --verbose            show all the outputs
  --noless                do not pipe the output to less when in a terminal

Required options for update-sdcard :
  -d , --dir <boot:rootfs> SD device mount point. First argument is
                          the boot partition mount point and the
                          rootfs partition mount point for rootfs
                          at least one mount point must be provided
                          Note: ROOT mode is required for rootfs setup

Available option for update-sdcard:
  -p, --project PROJECT    specify a PetaLinux reference project

```

Example to update SD card:

```
$ petalinux-util --update-sdcard --dir /mnt/vfat
It will copy BOOT.BIN and image.ub from <PROJECT>/images/linux/ to
/mnt/vfat. The /mnt/vfat is the SD device mount point.
```

Example to update boot partition and rootfs partition in SD card:

```
$ petalinux-util --update-sdcard --dir /mnt/vfat:/mnt/rootfs
It will require sudo permission.
It will copy BOOT.BIN and image.ub from <PROJECT>/images/linux/ to
/mnt/vfat and <PROJECT>/build/linux/rootfs/targetroot/ to /mnt/rootfs.
/mnt/rootfs is the SD device mount point. It needs to be ext2 and
above for rootfs.
```

Required options for update-sdcard :

```
-d , --dir <boot:rootfs> SD device mount point. First argument is
the boot partition mount point and the
rootfs partition mount point for rootfs
at least one mount point must be provided
Note: ROOT mode is required for rootfs setup
```

Available option for update-sdcard:

```
-p, --project PROJECT specify a PetaLinux reference project
```

Example to update SD card:

```
$ petalinux-util --update-sdcard --dir /mnt/vfat
It will copy BOOT.BIN and image.ub from <PROJECT>/images/linux/ to
/mnt/vfat. The /mnt/vfat is the SD device mount point.
```

Example to update boot partition and rootfs partition in SD card:

```
$ petalinux-util --update-sdcard --dir /mnt/vfat:/mnt/rootfs
It will require sudo permission.
It will copy BOOT.BIN and image.ub from <PROJECT>/images/linux/ to
/mnt/vfat and <PROJECT>/build/linux/rootfs/targetroot/ to /mnt/rootfs.
/mnt/rootfs is the SD device mount point. It needs to be ext2 and
above for rootfs.
```

Available options for Weblink:

```
petalinux-util --weblink on
enable weblink feature
petalinux-util --weblink off
disable weblink feature
```

Additional Resources

References

- PetaLinux SDK Application Development Guide (UG981)
- PetaLinux SDK Board Bringup Guide (UG980)
- PetaLinux SDK Firmware Upgrade Guide (UG983)
- PetaLinux SDK Getting Started Guide (UG977)
- PetaLinux SDK Installation Guide (UG976)
- PetaLinux SDK QEMU System Simulation Guide (UG982)

PetaLinux SDK Documentation is available at <http://www.xilinx.com/petalinux>.