PetaLinux Tools Documentation

Reference Guide

UG1144 (v2016.2) June 8, 2016

XILINX®

Revision History

Date	Version	Notes
11/25/2014	2014.4	Initial public release for PetaLinux Tools 2014.4
08/18/2015	2015.2	Updated for PetaLinux Tools 2015.2 release
08/26/2015	2015.2.1	Updated for PetaLinux Tools 2015.2.1 release
11/23/2015	2015.4	Updated for PetaLinux Tools 2015.4 release
05/06/2016	2016.1	Updated for PetaLinux Tools 2016.1 release
06/08/2016	2016.2	Updated for PetaLinux Tools 2016.2 release

Online Updates

Please refer to the PetaLinux v2016.2 Master Answer Record (Xilinx Answer Record #55776) for the latest updates on PetaLinux Tools usage and documentation.

XILINX

Table of Contents

About this Guide

PetaLinux is an Embedded Linux System Development Kit specifically targeting FPGA-based System-on-Chip designs. This guide helps the reader to familiarize with the tool enabling overall usage of PetaLinux.

Please note: The reader of this document is assumed to have basic Linux knowledge such as how to run Linux commands. The reader should also be aware of OS and Host system features such as OS bit version, Linux Distribution and Security Privileges.

PetaLinux Tools Installation Requirements

This section lists the requirements for the PetaLinux Tools Installation:

- Minimum workstation requirements:
  - 4 GB RAM (recommended minimum for Xilinx tools)
  - Pentium 4 2GHz CPU clock or equivalent
  - 20 GB free HDD space
  - Supported OS:
    - RHEL 6.6/6.7/7.1/7.2 (64-bit)
    - CentOS 7.1 (64-bit)
    - SUSE Enterprise 12.0 (64-bit)
    - Ubuntu 14.04.3 (64 bit)
- You need to have root access to perform some operations.
- PetaLinux requires a number of standard development tools and libraries to be installed on your Linux host workstation. Please install the libraries and tools listed in the following table on your host Linux. You must install the appropriate 32-bit compatible libraries due to some tools such as toolchains are 32bit executables.

  The table below describes the required packages, and how to install them on different Linux workstation environments.

Tool / Library	CentOS7.1	RHEL6.6	RHEL6.7	RHEL7.1	RHEL7.2	SuSE 12.0	Ubuntu 14.04.3
dos2unix	dos2unix 6.0.3	dos2unix 3.1-37	dos2unix 3.1-37	dos2unix 6.0.3	dos2unix 6.0.3	dos2unix 6.0.4	tofrodos 1.7.13
ip	iproute 3.10.0	iproute 2.6.32	iproute 2.6.32	iproute 3.10.0	iproute 3.10.0	iproute2	iproute2
gawk	gawk 4.0.2	gawk 3.1.7	gawk 3.1.7	gawk 4.0.2	gawk 4.0.2	gawk 4.1.0	gawk 4.0.1
gcc	gcc 4.8.3	gcc 4.4.7	gcc 4.4.7	gcc 4.8.3	gcc 4.8.5	gcc 4.8	gcc 4.8
git	git 1.8.3	git 1.7.1	git 1.7.1	git 1.8.3	git 1.8.3	git 1.7.1 or above	git 1.7.1 or above
make	make 3.81	make 3.81	make 3.81	make 3.82	make 3.82	make 4.0	make 3.81

Tool / Library	CentOS7.1	RHEL6.6	RHEL6.7	RHEL7.1	RHEL7.2	SuSE 12.0	Ubuntu 14.04.3
netstat	net-tools 2.0	net-tools 1.60	net-tools 1.60	net-tools 2.0	net-tools 2.0	net-tools	net-tools
ncurses devel	ncurses -devel 5.9-13	ncurses -devel 5.7-3	ncurses -devel 5.7-4	ncurses -devel 5.9-13	ncurses -devel 5.9-13	ncurses -devel	libncurses5 -dev
tftp server	tftp-server	tftp-server	tftp-server	tftp-server	tftp-server	atftp or yast2 -tftp-server	tftpd
zlib devel	zlib-devel 1.2.7	zlib-devel 1.2.3	zlib-devel 1.2.3	zlib-devel 1.2.7	zlib-devel 1.2	zlib-devel	zlib1g-dev
openssl devel	openssl -devel 1.0	openssl -devel 1.0	openssl -devel 1.0	openssl -devel 1.0	openssl -devel 1.0	libopenssl -devel	libssl -dev
flex	flex 2.5.37	flex 2.5.35	flex 2.5.35	flex 2.5.37	flex 2.5.37	flex	flex
bison	bison-2.7	bison 2.4.1	bison 2.4.1	bison 2.7.4	bison 2.7.4	bison	bison
libselinux	libselinux 2.2.2	libselinux 2.0.94	libselinux 2.0.94	libselinux 2.2.2	libselinux 2.2.2	libselinux1 2.3.2	libselinux1

WARNING: *Consult your system administrator if you are unsure about correct procedures for host system package management.*

IMPORTANT: *PetaLinux tools require your host system "`/bin/sh`" is `bash`. If you are using Ubuntu distribution and your "`/bin/sh`" is dash, you can consult your system administrator to change your default with `sudo dpkg-reconfigure dash` command.*

IMPORTANT: *PetaLinux v2016.2 works with Vivado 2016.2.*

PetaLinux Tools Installation Steps

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- PetaLinux Tools Installation Requirements is completed. Please refer to section PetaLinux Tools Installation Requirements.
- PetaLinux release package is downloaded. You can download PetaLinux installer from PetaLinux Downloads.

Run PetaLinux Tools Installer

PetaLinux Tools installation is very straight-forward.

Without any options, PetaLinux Tools will be installed into a subdirectory of the current working directory. Alternatively, an installation path may be specified.

E.g. To install PetaLinux Tools under `"/opt/pkg"`:

```
$ mkdir /opt/pkg
$ ./petalinux-v2016.2-final-installer.run /opt/pkg
```

This will install the PetaLinux Tools into `"/opt/pkg/petalinux-v2016.2-final"` directory.

Reading and agreeing to the PetaLinux EULA (End User License Agreement) is a required and integral part of the PetaLinux Tools installation process. Users can read the license agreement prior to running the installation. If users wish to keep the license for their records, the licenses are available in plain ASCII text in the following files:

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

> **NOTE:** *PetaLinux tools do not require a license to install or run.*

By default, the webtalk option is enabled to send tools usage statistics back to Xilinx. You can turn off the webtalk feature by running the `petalinux-util --webtalk` command:

> **IMPORTANT:** *Before running the PetaLinux command, you will need to source PetaLinux settings first. Please refer to section PetaLinux Working Environment Setup.*

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

Troubleshooting

This section describes some common issues you may experience while installing PetaLinux Tools.

If the PetaLinux Tools installation fails, the file `"$PETALINUX/post-install.log"` will be generated in your PetaLinux installation directory.

Problem/Error Message	Description and Solution
WARNING: You have less than 1Gbyte free space on the installation drive	***Problem Description:*** This warning message indicates that installation drive is almost full. You may not have enough free space to develop your hardware project and/or software project after the installation. ***Solution:*** • Clean up the installation drive to clear some more free space. Alternatively, • Move PetaLinux installation to another hard disk drive.
WARNING: No tftp server found	***Problem Description:*** This warning message indicates that you don't have a TFTP service running on your workstation. Without a TFTP service, you cannot download Linux system images to your target system using u-boot's network/TFTP capabilities. ***Solution:*** Enable the TFTP service on your workstation. If you are unsure how to enable this service, please contact your system administrator.
ERROR: GCC is not installed - unable to continue. Please install and retry	***Problem Description:*** This error message indicates that you don't have gcc installed on your workstation. ***Solution:*** Please install gcc using your Linux work-station's package management system. If you are unsure how to do this, please contact your system administrator.

Problem/Error Message	Description and Solution
ERROR: You are missing the following system tools required by PetaLinux: *missing-tools-list* **OR** ERROR: You are missing these development libraries required by PetaLinux: *missing-library-list*	***Problem Description:*** This error message indicates that you don't have the required tools or libraries listed in the "*missing-tools-list*" or "*missing-library-list*". ***Solution:*** Please install the packages of the missing tools. Refer to section PetaLinux Tools Installation Requirements for details.

Failed to open PetaLinux lib.	***Problem Description:***
	This error message indicates that a PetaLinux library failed to load. The possible reasons are:
	• The PetaLinux `"settings.sh"` has not been loaded.
	• The Linux Kernel you are running has SELinux configured. This can cause issues with regards to security context and loading libraries.
	
	Solution:
	1. Source the `"settings.sh"` script from the top-level PetaLinux directory. Please refer to section PetaLinux Working Environment Setup for more details.
	2. If you have SELinux enabled, determine if SELinux is in 'enforcing mode'.
	
	If SELinux is configured in 'enforcing mode', either reconfigure SELinux to 'permissive mode' (refer to SELinux manual), or change the security context of the libraries to allow access (see below for details).
	
	`$ cd $PETALINUX/tools/common/petalinux/lib`
	`$ chcon -R -t textrel_shlib_t lib`

PetaLinux Working Environment Setup

After the installation, the remainder of the setup is completed automatically by sourcing the provided "settings" scripts.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- PetaLinux Tools Installation is completed. Please refer to section PetaLinux Tools Installation Steps.
- "/bin/sh" is bash.

Steps to Setup PetaLinux Working Environment

1. Source the appropriate settings script:
   - For Bash as user login shell:

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

   - for C shell as user login shell:

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

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

   ```
   $ source /opt/petalinux-v2016.2-final/settings.sh
   PetaLinux environment set to '/opt/petalinux-v2016.2-final'
   INFO: Finalising PetaLinux installation
   INFO: Checking free disk space
   INFO: Checking installed tools
   INFO: Checking installed development libraries
   INFO: Checking network and other services
   ```

2. Verify that the working environment has been set:

   ```
   $ echo $PETALINUX
   /opt/petalinux-v2016.2-final
   ```

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

Troubleshooting

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

Problem/Error Message	Description and Solution
WARNING: /bin/sh is not bash	***Problem Description:*** This warning message indicates that your default shell is linked to dash. ***Solution:*** Use the following command to make bash as the default shell `$ chsh -s /bin/bash` `$ sudo dpkg-reconfigure dash` (alternatively for ubuntu)

PetaLinux BSP Installation

PetaLinux Reference BSPs are reference designs for you to start working with and customize for your own projects. These are provided in the form of installable BSP (Board Support Package) files, and includes all necessary design and configuration files, pre-built and tested hardware and software images, ready for downloading on your board or for booting in the QEMU system emulation environment.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- PetaLinux BSP (Board Support Package) is downloaded. You can download PetaLinux BSP from PetaLinux Downloads.
- PetaLinux Working Environment Setup is completed. Please refer to section PetaLinux Working Environment Setup.

PetaLinux BSP Installation Steps

1. Change to the directory under which you want PetaLinux projects to be created. E.g.: I want to create projects under /home/user:

```
$ cd /home/user
```

2. Run petalinux-create command on the command console:

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

You will see output similar to the following (board being referenced is based on BSP installed):

```
INFO: Create project:

INFO: Projects:
INFO:   * Xilinx-ZC702-2016.2
INFO: has been successfully installed to /home/user
INFO: New project successfully created in /home/user
```

In the above example, upon execution of petalinux-create command, the projects extracted from BSP and being installed are listed on the display console. If you run `ls` from `"/home/user"`, you will see the installed projects.

Troubleshooting

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

Problem/Error Message	Description and Solution
petalinux-create: command not found	***Problem Description:*** This message indicates that it is unable to find "petalinux-create" command, hence it can't proceed with BSP installation. ***Solution:*** You have to setup your environment for PetaLinux Tools. Please refer to section PetaLinux Working Environment Setup to set it up.

Create Hardware Platform with Vivado

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

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- Vivado Design Suite is installed. You can download Vivado Design Suite from Vivado Design Tool Downloads.
- You have setup Vivado Tools Working Environment. If you have not, source the appropriate settings scripts as follows.

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

- You know how to use Xilinx Vivado and SDK tools.

Configure a Hardware Platform for Linux

You can create a hardware platform with Vivado. Regardless of how the hardware platform is created and configured, there are a small number of hardware IP and software platform configuration changes required to make the hardware platform Linux ready. These are described below.

Zynq UltraScale+ MPSoC

The following is a list of requirements for a Zynq UltraScale+ MPSoC hardware project to boot Linux:

1. External memory controller with at least 64MB of memory (Required)
2. UART (Optional, but required for serial console)

IMPORTANT: *If soft IP is used, ensure the interrupt signal is connected*

3. Non-volatile memory (Optional) e.g. QSPI Flash, SD/MMC
4. Ethernet (Optional, essential for network access)

IMPORTANT: *If soft IP with interrupt or external PHY device with interrupt is used, ensure the interrupt signal is connected*

Zynq-7000

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

1. One Triple Timer Counter (TTC) (Required)

IMPORTANT:

- *If multiple TTCs are enabled, the Zynq-7000 Linux kernel uses the first TTC block from the device tree.*
- *Please make sure the TTC is not used by others.*

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

IMPORTANT: *If soft IP is used, ensure the interrupt signal is connected*

4. Non-volatile memory (Optional) e.g. QSPI Flash, SD/MMC
5. Ethernet (Optional, essential for network access)

IMPORTANT: *If soft IP with interrupt or external PHY device with interrupt is used, ensure the interrupt signal is connected*

MicroBlaze AXI

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

1. IP core check list:
   - External memory controller with at least 32MB of memory (Required)
   - Dual channel timer with interrupt connected (Required)
   - UART with interrupt connected (Optional, but required for serial console)
   - Non-volatile memory such as Linear Flash or SPI Flash (Optional)
   - Ethernet with interrupt connected (Optional, but required for network access)
2. MicroBlaze CPU configuration:
   - MicroBlaze with MMU support by selecting either **Linux with MMU** or **Low-end Linux with MMU** configuration template in the MicroBlaze configuration wizard.

IMPORTANT: *Do not disable any instruction set related options that are enabled by the template, unless you understand the implications of such a change.*

- The MicroBlaze initial bootloader, called FS-BOOT, has a minimum BRAM requirement. 4KByte is required for Parallel flash and 8KByte for SPI flash when the system boots from non-volatile memory.

Export Hardware Platform to PetaLinux Project

This section describes how to export hardware platform to PetaLinux Project.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- A hardware platform is created with Vivado. Please refer to section Create Hardware Platform with Vivado for more information.

Exporting Hardware Platform

After you have configured your hardware project, build the hardware bitstream. The PetaLinux project requires a hardware description file (`.hdf` file) with information about the processing system. You can get the hardware description file by running `"Export Hardware"` from Vivado.

PetaLinux tools can generate a device tree source file, u-boot config header files, and enable some Xilinx IP kernel drivers based on the hardware description file. This will be described in later sections.

For Zynq UltraScale+ MPSoC platform, if you want to boot with PMU(Power Mangement Unit) firmware, you will also need to build the PMU firmware with XSDK. Please refer to `"MPSoC Software Development Guide"` for the details on how to build the PMU firmware with XSDK.

Create a New PetaLinux Project

This section describes how to create a new PetaLinux project.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- PetaLinux Working Environment Setup is completed. Please refer to section PetaLinux Working Environment Setup.

Create New Project

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

```
$ petalinux-create --type project --template <CPU_TYPE> --name <PROJECT_NAME>
```

The parameters are as follows:

- `--template <CPU_TYPE>` - The supported CPU types are `zynqMP`, `zynq` and `microblaze`
- `--name <PROJECT_NAME>` - The name of the project you are building.

This command will create a new PetaLinux project folder from a default template. Later steps customize these settings to match the hardware project created previously.

TIP: *For details of PetaLinux project structure, please refer to Appendix A PetaLinux Project Structure.*

Version Control

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

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have created a new PetaLinux project or have an existing PetaLinux project. Please refer to section Create New PetaLinux Project for more information on creating the PetaLinux project.

Version Control

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

- `"<plnx-proj-root>/.petalinux"`
- `"<plnx-proj-root>/build/"`
- `"<plnx-proj-root>/images/"`

Import Hardware Configuration

This Section explains how to update an existing/newly created PetaLinux project with a new hardware configuration. This enables the user to make PetaLinux Tools' software platform ready for building a Linux system, customized to your new hardware platform.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have exported the hardware platform and `.hdf` file is generated. Please refer to section Export Hardware Platform to PetaLinux.
- You have created a new PetaLinux project or have an existing PetaLinux project. Please refer to section Create New PetaLinux Project for more information on creating the PetaLinux project.

Steps to Import Hardware Configuration

Steps to import hardware configuration are:

1. Change into the directory of your PetaLinux project.

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

2. Import the hardware description with `petalinux-config` command, by giving the path to `.hdf` file (e.g.: hwproject/hwproject.sdk/hwproject_design_wrapper_hw_platform_0) as follows:

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

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

```
linux Components Selection  --->
Auto Config Settings  --->
-*- Subsystem AUTO Hardware Settings  --->
Kernel Bootargs  --->
u-boot Configuration  --->
Image Packaging Configuration  --->
Firmware Version Configuration  --->
```

Make sure "`Subsystem AUTO Hardware Settings --->`" is selected, and go into the menu which is similar to the following:

```
--- Subsystem AUTO Hardware Settings
System Processor (ps7_cortexa9_0)  --->
Memory Settings  --->
Serial Settings  --->
Ethernet Settings  --->
Flash Settings  --->
SD/SDIO Settings  --->
[ ]   Advanced bootable images storage Settings  --->
```

"Subsystem AUTO Hardware Settings --->" menu allows customizing system wide hardware settings.

This step may take a few minutes to complete. This is because the tool will parse the hardware description file for hardware information required to update the device tree, PetaLinux u-boot configuration files and the kernel config files based on the "Auto Config Settings --->" and "Subsystem AUTO Hardware Settings --->" settings.

E.g. If ps7_ethernet_0 as the Primary Ethernet is selected and user enables auto update for kernel config and u-boot config, the tool will automatically enable its kernel driver and also updates the u-boot configuration headers for u-boot to use the selected ethernet controller.

NOTE: *For more details on Auto Config Settings menu, please refer to Appendix C Auto Config Settings.*

Build System Image

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have PetaLinux Tools software platform ready for building a Linux system, customized to your hardware platform. Please refer to section Import Hardware Configuration for more details.

Steps to Build PetaLinux System Image

1. Change into the directory of your PetaLinux project.

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

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

```
$ petalinux-build
```

The console shows the compilation progress. e.g.:

```
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
```

The full compilation log `"build.log"` is stored in the `build` subdirectory of your PetaLinux project.

The Linux software images and the device tree are generated in the `images/linux` subdirectory of your PetaLinux project.

IMPORTANT: *By default, besides the kernel, rootfs and u-boot, the PetaLinux project is configured to generate and build the first stage bootloader. Please refer to Appendix B Generate First Stage Bootloader for more details on the auto generated first stage bootloader.*

Generate uImage

When you run `petalinux-build`, it will generate FIT image for Zynq family devices and MicroBlaze platforms and RAM disk image `urootfs.cpio.gz` will also be generated. If you want to use uImage instead, you can use "`petalinux-package --image`" instead. E.g.

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

The uImage will be generated to `images/linux` subdirectory of your PetaLinux project. You will then need to configure your u-boot to boot with uImage. If you have selected "`PetaLinux u-boot config`" as your u-boot config target, you can modify "`subsystems/linux/configs/u-boot/platform-top.h`" of your PetaLinux project to overwirte the `CONFIG_EXTRA_ENV_SETTINGS` macro to define your u-boot boot command to boot with uImage.

Troubleshooting

This section describes some common issues you may experience while building PetaLinux system images.

Problem/Error Message	Description and Solution
[ERROR] <path-to-installed-PetaLinux> /etc/build/common.mk:17: *** "No architecture is defined!". Stop.	***Problem Description:*** This error message indicates that `petalinux-build` process cannot be completed because PetaLinux tools cannot understand hardware architectural definition. ***Solution:*** You have to choose board and device appropriately in Vivado Hardware Project and import hardware description. Please refer to section Import Hardware Configuration.

Generate Boot Image for Zynq Family Devices

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

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have built PetaLinux system image. Please refer to section Build System Image for more information on it.

Generate Boot Image

Before executing this step, ensure you have built the hardware bitstream. The boot image can be put into Flash or SD card. When you power on the board, it can boot from the boot image. A boot image usually contains a first stage bootloader image, FPGA bitstream and u-boot.

Follow the step below to generate the boot image in `".bin"` format.

```
$ petalinux-package --boot --fsbl <FSBL image> --fpga <FPGA bitstream> --u-boot
```

For detailed usage, please refer to the --help option or document `PetaLinux Command Reference Document(ug1157)`.

Generate Boot Image for Zynq UltraScale+ MPSoC

This section is for Zynq UltraScale+ MPSoC only and describes how to generate `BOOT.BIN` for Zynq UltraScale+ MPSoC. Skip this section for MicroBlaze and Zynq targets.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have built PetaLinux system image. Please refer to section Build System Image for more information on it.

Generate Boot Image for Zynq UltraScale+ MPSoC

Before executing this step, ensure you have built the hardware bitstream. The boot image can be put into Flash or SD card. When you power on the board, it can boot from the boot image. A boot image usually contains a first stage bootloader image, FPGA bitstream and u-boot.

Follow the step below to generate the boot image in `".bin"` format.

```
$ petalinux-package --boot --fsbl <FSBL image> --fpga <FPGA bitstream> --u-boot
```

If you want to generate the boot image with PMU firmware, you can use the following command:

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

For detailed usage, please refer to the --help option or document `PetaLinux Command Reference Document(ug1157)`.

Generate Boot Image for MicroBlaze

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

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have built PetaLinux system image. Please refer to section Build System Image for more information on it.

Generate Boot Image for MicroBlaze

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

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

It will generate `boot.mcs` in your working directory and it will be copied to the `<proj>/images/linux/` directory. With the above command, the MCS file contains `fpga bit`, `fs-boot`, `u-boot` and kernel image `image.ub`.

Command to generate `.mcs` file with fs-boot and fpga only:

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

Command to generate `.mcs` file with fpga, fs-boot and u-boot:

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

For detailed usage, please refer to the --help option or document `PetaLinux Command Reference Document(ug1157)`.

Package Prebuilt Image

This section describes how to package newly built images into prebuilt directory.

> **NOTE:** *This step helps in making use of prebuilt capability to boot with JTAG/QEMU. This step is not mandatory to boot with JTAG/QEMU.*

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- For Zynq family devices: You have generated boot image, refer to Generate Boot Image for Zynq Family Devices.
- For MicroBlaze: You have generated system image, refer to Build System Image.

Steps to Package Prebuilt Image

1. Change into the root directory of your project.

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

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

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

   For detailed usage, please refer to the --help option or document `PetaLinux Command Reference Document(ug1157)`.

Using petalinux-boot Command with Prebuilt Images

Booting a PetaLinux Image is achieved with the `petalinux-boot` command, along with `--qemu` option to boot the image under software emulation (QEMU) and `--jtag` on a hardware board. This section describes different boot levels for prebuilt option.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have packaged prebuilt images. Please refer to Package Prebuilt Image.

Boot Levels for Prebuilt Option

`--prebuilt <BOOT_LEVEL>` boots prebuilt images (override all settings). Supported boot level is 1 to 3.

- Level 1: Download the prebuilt FPGA bitstream
  - It will also boot FSBL for Zynq-7000 and Zynq UltraScale+ MPSoC
- Level 2: Download the prebuilt FPGA bitstream and boot the prebuilt u-boot.
  - For Zynq-7000: it will also boot FSBL before booting u-boot.
  - For Zynq UltraScale+ MPSoC: it will also boot FSBL, and then ATF(Arm Trusted Firmware) before booting u-boot.
- Level 3:
  - For MicroBlaze: Download the prebuilt FPGA bitstream and boot the prebuilt kernel image on target.
  - For Zynq-7000: Download the prebuilt FPGA bitstream and FSBL and boot the prebuilt u-boot and boot the prebuilt kernel on target.
  - For Zynq UltraScale+ MPSoC: Download the prebuilt FPGA bitstream, the prebuilt FSBL, the prebuilt ATF and the prebuilt kernel on target.

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

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

Example to show the usage of pmufw option with prebuilt:

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

By default, the `--pmufw` will take `<plnx-proj-root>/pre-built/linux/images/pmufw.elf` as the PMU firmware.

Boot a PetaLinux Image on QEMU

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

> **NOTE:** *For the details on Xilinx IP models supported by QEMU, refer to Appendix E Xilinx IP models supported by QEMU.*

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have a PetaLinux system image by either installing a PetaLinux BSP (refer to section PetaLinux BSP Installation) OR by building your own PetaLinux project (refer to Build System Image).
- If you are going to use `--prebuilt` option for QEMU boot, you will also need to have prebuilt images packaged, refer to Package Prebuilt Image.

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

Steps to Boot a PetaLinux Image on QEMU

PetaLinux provides QEMU support to enable testing of a PetaLinux software image in an emulated 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
```

> **NOTE:** *Please note that if you wish not to use prebuilt capability for QEMU boot, refer to Additional options for booting on QEMU on page 30.*

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

> **TIP:** *To know more about different boot levels for prebuilt option, refer to Using petalinux-boot Command with Prebuilt Images on page 28.*

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

```
Freeing unused kernel memory: 3084K (c067f000 - c0982000)

INIT: version 2.88 booting
...

Creating /dev/flash/* device nodes
random: dd urandom read with 22 bits of entropy available
Starting internet superserver: inetd.
INIT: Entering runlevel: 5
Configuring network interfaces... udhcpc (v1.23.2) started
Sending discover...
macb e000b000.ethernet eth0: link up (1000/Full)
Sending discover...
Sending select for 10.10.70.1...
Lease of 10.10.70.1 obtained, lease time 600
/etc/udhcpc.d/50default: Adding DNS 172.19.128.1
/etc/udhcpc.d/50default: Adding DNS 172.19.129.1
done.

Xilinx-ZC702-2016_2 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*

Additional Options for booting on QEMU

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

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

  - For MicroBlaze and Zynq-7000, it will boot `<plnx-proj-root>/images/linux/u-boot.elf` with QEMU.
  - For Zynq UltraScale+ MPSoC, it will loades the `<plnx-proj-root>/images/linux/u-boot.elf` and boots the ATF image `<plnx-proj-root>/images/linux/bl31.elf` with QEMU, and the ATF will then boot the loaded u-boot image.

- To download newly built kernel with qemu:

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

  - For MicroBlaze, it will boot `<plnx-proj-root>/images/linux/image.elf` with QEMU.
  - For Zynq-7000, it will boot `<plnx-proj-root>/images/linux/zImage` with QEMU.

- For Zynq UltraScale+ MPSoC, it will loads the kernel image `<plnx-proj-root>/images/linux/Image` and boots the ATF image `<plnx-proj-root>/images/linux/bl31.elf` with QEMU, and the ATF will then boot the loaded kernel image.

- Direct Boot a Linux Image with Specific DTB:

  Device Trees (DTB files) are used to describe the hardware architecture and address map to the Linux kernel. The PetaLinux system emulator also uses DTB files to dynamically configure the emulation environment to match your hardware platform.

  If no DTB file option is provided, `petalinux-boot` extracts the DTB file from the given `image.elf` for Microblaze and from `"<plnx-proj-root>/images/linux/system.dtb"` for Zynq-7000 and Zynq UltraScale+ MPSoC. Alternatively, you can use the `--dtb` option as follows:

```
$ petalinux-boot --qemu --image ./images/linux/zImage --dtb ./images/linux/system.dtb
```

Boot a PetaLinux Image on Hardware with SD Card

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

> **NOTE:** *This section is for Zynq-7000 and Zynq UltraScale+ MPSoC only, since they allow you to boot from SD card.*

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have installed PetaLinux Tools on the Linux workstation. If you have not installed, please refer to section PetaLinux Tools Installation Steps.
- You have installed PetaLinux BSP on the Linux workstation. If you have not installed, please refer to section PetaLinux BSP Installation.
- A serial communication program such as minicom/kermit/gtkterm has been installed; the baud rate of the serial communication program has been set to 115200bps.

Steps to Boot a PetaLinux Image on Hardware with SD Card

1. Mount the SD card on your host machine.
2. Copy the following files from <plnx-proj-root>/pre-built/linux/images/ into the root directory of the first partition which is in FAT32 format in the SD card:
   - BOOT.BIN
   - image.ub
3. Connect the serial port on the board to your workstation.
4. Open a console on the workstation and start the preferred serial communication program (e.g. `kermit`, `minicom`, `gtkterm`) with the baud rate set to 115200 on that console.
5. Power off the board.
6. Set the boot mode of the board to SD boot. Refer to the board documentation for details.
7. Plug the SD card into the board.
8. Power on the board.
9. Watch the serial console, you will see the boot messages similar to the following:

```
Freeing unused kernel memory: 3084K (c067f000 - c0982000)

INIT: version 2.88 booting
...

Creating /dev/flash/* device nodes
random: dd urandom read with 22 bits of entropy available
Starting internet superserver: inetd.
INIT: Entering runlevel: 5
Configuring network interfaces... udhcpc (v1.23.2) started
Sending discover...
macb e000b000.ethernet eth0: link up (1000/Full)
Sending discover...
Sending select for 10.10.70.1...
Lease of 10.10.70.1 obtained, lease time 600
/etc/udhcpc.d/50default: Adding DNS 172.19.128.1
/etc/udhcpc.d/50default: Adding DNS 172.19.129.1
done.

Xilinx-ZC702-2016_2 login:
```

Figure 2: Serial console output of `Zynq-7000 SD boot`

TIP: *If the user wishes to stop autoboot, hit any key when you see the messages similar to the following on the console:*

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

10. Type user name root and password root on the serial console to log into the PetaLinux system.

Troubleshooting

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

Problem/Error Message	Description and Solution

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

TIP: *To know more about u-boot options, use the following command.*

```
$ U-Boot-PetaLinux> printenv
```

Boot a PetaLinux Image on Hardware with JTAG

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

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have a PetaLinux system image by either installing a PetaLinux BSP (refer to section PetaLinux BSP Installation) OR by building your own PetaLinux project (refer to section Build System Image).
- This is `OPTIONAL` and only needed if you wish to make use of prebuilt capability for JTAG boot.

  You have packaged prebuilt images, refer to section Package Prebuilt Image.
- A serial communication program such as minicom/kermit/gtkterm has been installed; the baud rate of the serial communication program has been set to 115200bps.
- Appropriate JTAG cable drivers have been installed. You can download drivers from Digilent Adept Downloads.

Steps to Boot a PetaLinux Image on Hardware with JTAG

1. Power off the board.
2. Connect the JTAG port on the board with the JTAG cable to your workstation.
3. Connect the serial port on the board to your workstation.
4. If your system has ethernet, also connect the Ethernet port on the board to your local network.
5. For Zynq family device boards, ensure that the mode switches are set to JTAG mode. Refer to the board documentation for details.
6. Power on the board.
7. Open a console on your workstation and start with preferred serial communication program (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
```

NOTE: *Please note that if you wish not to use prebuilt capability for JTAG boot, refer to Additional options for booting with JTAG.*

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

TIP: *To know more about different boot levels for prebuilt option, refer to Using petalinux-boot Command with Prebuilt Images.*

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

```
$ petalinux-boot --jtag --prebuilt 3
INFO: Launching XSDB for file download and boot.
INFO: This may take a few minutes, depending on the size of your image.
INFO: Configuring the FPGA...
INFO: Downloading bitstream to the target.
INFO: Downloading ELF file to the target.
INFO: Downloading ELF file to the target.
INFO: Downloading ELF file to the target.
INFO: This may take a few minutes, depending on the size of your image.
```

Figure 3: Workstation console output for successful `petalinux-boot`

```
Freeing unused kernel memory: 3084K (c067f000 - c0982000)

INIT: version 2.88 booting
...

Creating /dev/flash/* device nodes
random: dd urandom read with 22 bits of entropy available
Starting internet superserver: inetd.
INIT: Entering runlevel: 5
Configuring network interfaces... udhcpc (v1.23.2) started
Sending discover...
macb e000b000.ethernet eth0: link up (1000/Full)
Sending discover...
Sending select for 10.10.70.1...
Lease of 10.10.70.1 obtained, lease time 600
/etc/udhcpc.d/50default: Adding DNS 172.19.128.1
/etc/udhcpc.d/50default: Adding DNS 172.19.129.1
done.

Xilinx-ZC702-2016_2 login:
```

Figure 4: 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 on the PetaLinux reference design being tested.

9. Type user name `root` and password `root` on the serial console to log into the PetaLinux system.

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

NOTE: *Additional options for booting with JTAG*

- *To download a bitstream to target board:*

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

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

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

- *To download newly built kernel to target board:*

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

For MicroBlaze, this will boot <plnx-proj-root>/images/linux/image.elf on target board.

For Zynq-7000, this will boot <plnx-proj-root>/images/linux/zImage on target board.

For Zynq UltraScale+ MPSoC, this will boot <plnx-proj-root>/images/linux/Image on target board.

- *To Download a image with a bitstream with --fpga --bistream <BITSTREAM> option:*

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

The above command will download the bistream and then download the U-Boot image.

- *To see the verbose output of jtag boot with -v option:*

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

- *To download PMU firmware to target board with U-Boot:*

```
$ petalinux-boot --jtag --pmufw <PATH_TO_PMUFW_ELF> --u-boot
```

Logging Tcl/XMD for JTAG Boot

Use the following command to take log of XMD commands used during JTAG boot. It dumps tcl script (which in turn invokes the xmd commands) data to `test.txt`.

```
$ cd <plnx-proj-root>
$ petalinux-boot --jtag --prebuilt 3 --tcl test.txt
```

Troubleshooting

This section describes some common issues you may experience while booting a PetaLinux image on hardware with JTAG.

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

Problem/Error Message	Description and Solution
Cannot see any console output when trying to boot U-Boot or kernel on hardware but boots correctly on QEMU.	***Problem Description:*** This problem is usually caused by one or more of the following: • The serial communication terminal application is set with the wrong baud rate. • Mismatch between hardware and software platforms. ***Solution:*** • Ensure your terminal application baud rate is correct and matches your hardware configuration. • Ensure the PetaLinux project is built with the right hardware platform. ◦ Import hardware configuration properly, refer to section Import Hardware Configuration. ◦ Check the `"Subsystem AUTO Hardware Settings --->"` submenu to make sure it matches the hardware platform. ◦ Check the `"Serial settings --->"` submenu under `"Subsystem AUTO Hardware Settings --->"` to ensure `stdout`, `stdin` are set to the correct UART IP core. ◦ Rebuild system images, refer to Build System Image.

Boot a PetaLinux Image on Hardware with TFTP

This section describes how to boot a PetaLinux image using TFTP (Trivial File Transfer Protocol).

TIP: *TFTP boot saves a lot of time because it is much faster than booting through JTAG and you don't have to flash the image for every change in kernel source.*

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- Host environment with TFTP server is setup and PetaLinux Image is built for TFTP boot. Please refer to section Configure TFTP Boot.
- You have packaged prebuilt images, refer to section Package Prebuilt Image.
- A serial communication program such as minicom/kermit/gtkterm has been installed; the baud rate of the serial communication program has been set to 115200bps.
- Ethernet connection is setup properly between Host and Linux Target.
- Appropriate JTAG cable drivers have been installed. You can download drivers from Digilent Adept Downloads.

Steps to Boot a PetaLinux Image on Hardware with TFTP

1. Power off the board.
2. Connect the JTAG port on the board to the workstation using a JTAG cable with the JTAG cable.
3. Connect the serial port on the board to your workstation.
4. Connect the Ethernet port on the board to the local network via a network switch.
5. For Zynq family device boards, ensure that the mode switches are set to JTAG mode. Refer to the board documentation for details.
6. Power on the board.
7. Open a console on your workstation and start with preferred serial communication program (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 2
```

The `--jtag` option tells `petalinux-boot` to boot on hardware via JTAG, and the `--prebuilt 2` option will download the prebuilt bitstream (and FSBL for zynq) to target board, and then boot prebuilt u-boot on target board.

9. When autoboot starts, hit any key to stop it.

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

```
U-Boot 2016.02 (Jun 09 2016 - 16:03:40 -0600), Build: jenkins-petalinux_project-common_projects-BSP_zcu102-250

I2C:   ready
DRAM:  2 GiB
Enabling Caches...
EL Level: EL2
MMC:   sdhci@ff170000: 0
SF: Detected N25Q512A with page size 256 Bytes, erase size 64 KiB, total 64 MiB
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Bootmode: JTAG_MODE
SCSI:  SATA link 0 timeout.
AHCI 0001.0000 32 slots 2 ports 1.5 Gbps 0x3 impl SATA mode
flags: ncq only
scanning bus for devices...
Found 0 device(s).
Net:   ZYNQ GEM: ff0e0000, phyaddr 12, interface rgmii-id
eth0: ethernet@ff0e0000
U-BOOT for Xilinx-ZCU102-2016_2

BOOTP broadcast 1
DHCP client bound to address 10.0.2.15 (2 ms)
Hit any key to stop autoboot:  0
U-Boot-PetaLinux>
```

Figure 5: Workstation console output for successful u-boot download

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

```
U-Boot-PetaLinux> print serverip
```

11. Set the server IP address to the host IP address using the following commands.

```
U-Boot-PetaLinux> set serverip <HOST IP ADDRESS>; saveenv
```

12. Boot the kernel using the following command.

```
U-Boot-PetaLinux> run netboot
```

Troubleshooting

Problem/Error Message	Description and Solution
## Error: "serverip" not defined.	***Problem Description:*** This error message indicates that `u-boot` environment variable `serverip` is not set. You have to set it to IP Address of the host where the image resides. ***Solution:*** Use the following command to set the `serverip`: `U-Boot-PetaLinux> set serverip <HOST IP ADDRESS>;saveenv`

Firmware Packaging

This section describes the procedure for Firmware Packaging. The PetaLinux `petalinux-package` tool has a `--firmware` option to package the images and bitstream required for the upgrade.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have built PetaLinux system image. Please refer to section Build System Image for more details.

Steps to Package Firmware with BOOT.BIN and Kernel image for Zynq-7000

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

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

2. The command to package the firmware is:

```
$ petalinux-package --firmware --bootbin=<BOOT_BIN> --linux
```

3. It will create `firmware.tar.gz` archive in your working directory including the specified <BOOT_BIN> and `<plnx-proj-root>/images/linux/image.ub`.

Steps to Package Firmware for MicroBlaze

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

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

2. The command to package the firmware is:

```
$ petalinux-package --firmware --fpga <BITSTREAM> --u-boot --linux
```

3. It will create `firmware.tar.gz` archive in your working directory including

   - Specified <BITSTREAM>
   - <plnx-proj-root>/images/linux/u-boot-s.bin
   - <plnx-proj-root>/images/linux/image.ub

> **NOTE:** *Please note that `petalinux-package --firmware` tool is used to generate the firmware package for the petalinux firmware upgrade demo application only.*

BSP Packaging

BSPs are useful for distribution in general and allude to Xilinx Worldwide Technical Support as a specific use case. Xilinx WTS requires a bare minimum design packaged as a `Petalinux BSP` to get a testcase for further debugging and support. This section explains, with an example, how to package a BSP with PetaLinux project.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. Please refer to section Import Hardware Configuration for more information.

Steps for BSP Packaging

Steps on how to package a project for submission to WTS for debug are as follows:

1. You can go outside the PetaLinux project directory to run `petalinux-package` command.
2. Use the following commands to package the bsp.

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

3. This will generate MY.BSP including the following elements from the specified project:
   - <plnx-proj-root>/hw-description/
   - <plnx-proj-root>/config.project
   - <plnx-proj-root>/.petalinux/
   - <plnx-proj-root>/subsystems/
   - <plnx-proj-root>/pre-built/
   - all selected components

Additional BSP Packaging Options

1. BSP packaging with hardware source.

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

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

2. BSP packaging excluding local components.

```
$ petalinux-package --bsp -p <plnx-proj-root> --output MY.BSP --no-local
```

   It will not include any local component in `<plnx-proj-root>/components` directory. Please note that the configuration file will not be changed. Hence there will be a possibility of failure while rebuilding the project from `MY.BSP`.

3. BSP packaging excluding extern components.

```
$ petalinux-package --bsp -p <plnx-proj-root> --output MY.BSP --no-extern
```

It will not include any extern component in user specified components searchpaths. Please note that the configuration file will not be changed. Hence there will be a possibility of failure while rebuilding the project from `MY.BSP`.

Firmware Version Configuration

This section explains how to do firmware version configuration using `petalinux-config` command.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. Please refer to section Import Hardware Configuration for more information.

Steps for Firmware Version Configuration

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

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

2. Launch the top level system configuration menu.

```
$ petalinux-config
```

3. Select Firmware Version Configuration.
4. Select Host Name, Product Name, Firmware Version as per the requirement to edit them.
5. Exit the menu and select <Yes> when asked `Do you wish to save your new configuration?`:

```
Do you wish to save your new configuration ? <ESC><ESC>
to continue.

< Yes >      <  No  >
```

Root file system Type Configuration

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

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. Please refer to section Import Hardware Configuration for more information.

Steps for Root file system Type Configuration

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

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

2. Launch the top level system configuration menu.

```
$ petalinux-config
```

3. Select Image Packaging Configuration.
4. Select Rootfile System type.
5. Select INITRAMFS/INITRD/JFFS2/NFS/SD card as per the requirement.
6. Save Configuration settings.

Boot Images Storage Configuration

This section provides details about configuration of the Boot Device e.g. Flash, SD/MMC using `petalinux-config` command.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. Please refer to section Import Hardware Configuration for more information.

Steps for Boot Images Storage Configuration

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

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

2. Launch the top level system configuration menu.

```
$ petalinux-config
```

3. Select Subsystem AUTO Hardware Settings.
4. Select Advanced Bootable Images Storage Settings.
5. Select boot image settings.
6. Select Image Storage Media.
7. Select boot device as per the requirement. To set flash as the boot device select `primary flash`. To make SD card as the boot device select `primary sd`.
8. Under the `Advanced Bootable Images Storage Settings` submenu, select kernel image settings.
9. Select Image Storage Media.
10. Select storage device as per the requirement. To set flash as the boot device select `primary flash`. To make SD card as the boot device select `primary sd`.
11. Save Configuration settings.

TIP: *To select a menu/submenu which was deselected before, press the down arrow key (↓) to scroll down the menu or the up arrow key (↑) to scroll up the menu.Once the cursor is on the menu, then press "y". To deselect a menu/submenu, follow the same process and press "n" at the end.*

Troubleshooting

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

Problem/Error Message	Description and Solution
ERROR: Failed to config linux/kernel!	***Problem Description:*** This error message indicates that it is unable to configure the linux-kernel component with menuconfig. ***Solution:*** Check whether all required libraries/packages are installed properly. Please refer to section PetaLinux Tools Installation Requirements.

Primary Flash Partition Configuration

This sections provides details on how to configure flash partition with PetaLinux menuconfig.

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

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

2. Launch the top level system configuration menu.

```
$ petalinux-config
```

3. Select Subsystem AUTO Hardware Settings.
4. Select Flash Settings.
5. Select a flash device as the `Primary Flash`.
6. Set the name and the size of each partition.

TIP: *Please note that PetaLinux tools uses the following partitions to generate u-boot commands:*

- *boot*
- *bootenv, it is for u-boot environment variables.*
- *kernel*

PetaLinux tools uses the start address for parallel flash or start offset for SPI flash and the size of the above partitions to generate the following u-boot commands:

- *update_boot if the boot image, which is a u-boot image for MicroBlaze, a BOOT.BIN image for Zynq family devices, is selected to be stored in the primary flash.*
- *update_kernel, and load_kernel if the kernel image which is the FIT image image.ub, is selected to be stored in the flash.*

Base Root File System Configuration

This section talks about Base Root File System Configuration.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. Please refer to section Import Hardware Configuration for more information.

Steps for Base Root File System Configuration

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

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

2. Launch the top level system configuration menu.

```
$ petalinux-config
```

3. Select `linux Components Selection`
4. Select `petalinux-rootfs` as rootfs.
5. Set the Yocto RPM repo url to `rootfs package feed url`. By default, it is the one from your installed PetaLinux tools.

TIP:

- *The repo needs to have 3 channels: `all`, <ARCH>, and <FAMILY>_generic. E.g. The RPM repo for Zynq UltraScale+ MPSoC:*
  - *<repo>/rpm/all*
  - *<repo>/rpm/aarch64*
  - *<repo>/rpm/zynqmp_generic*
- *If the repo is located in local file system, please start the repo path with `file:///`*
- *You can input up to 4 repos in the menuconfig. The first repo has the lowest priority and the last repo has the highest priority in the sequence shown in the menu.*

6. Save Configuration settings.
7. Select which prebuilt Yocto package you want to inlucde in your rootfs
   (a) Launch rootfs menuconfig

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

(b) Select `Filesystem Packages`

(c) Select the package you want to include

TIP: *If you select `base-system-default`, it will automatically select the packages which will compose a basic root file system.*

Use your own Yocto RPM repo

You can rebuild the Yocto RPM packages repo, or you can use `meta-petalinux` and `meta-xilinx` to generate a RPM repo to include more prebuilt packages. Here are the links on how to generate the PetaLinux Yocto RPM packages:

- Build PetaLinux Yocto RPM Packages
- Add New Packages in meta-petalinux

For more information on how to use Yocto, please refer to the following manual: Yocto Reference Manual

When you generate the RPM packages, please make sure you use the Linux toolchain from your installed PetaLinux tools or the one from the Xilinx SDK of the same version as the PetaLinux tools.

After you have generated the RPM packages repo, you can follow the steps in above section `Steps for Base Root File System Configuration` to configure your PetaLinux project to use the repo you have generated.

The generated RPM packages are in `<YOCTO_BUILD>/tmp-glibc/deploy/rpm`. You can configure your PetaLinux project to use `file:///<YOCTO_BUILD>/tmp-glibc/deploy` as the repo URL, or you can copy the `<YOCTO_BUILD>/tmp-glibc/deploy/rpm` to somewhere else such as `<MY_LOCAL_REPO>/rpm`, and then configure your PetaLinux project to use `<MY_LOCAL_REPO>` as the repo URL.

Managing Image Size

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

FIT image is the default bootable image format. By default the FIT image is composed of kernel image, DTB and rootfs image.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. Please refer to section Import Hardware Configuration for more information.

Steps for Managing Image Size

FIT Image size can be reduced using the following methods:

1. Launch the RootFS configuration menu using the following command:

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

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

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

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

   Select General Setup. Under this submenu, you can find options to set the `config` items. Any item that is not mandatory to have in the system can be disabled to reduce the kernel image size. For e.g. `CONFIG_SHMEM, CONFIG_AIO, CONFIG_SWAP, CONFIG_SYSVIPC`. For more details, refer Linux kernel documentation.

WARNING: *Note that disabling of some config items may lead to unsuccessful boot. So it is expected that the user has knowledge of `config` items before disabling them.*

TIP:

- *Inclusion of extra config items and Filesystem packages lead to increase in the kernel image size and RootFS size respectively.*

Configuring INITRAMFS Boot

INITRAMFS, abbreviated as initial RAM File System, is the successor of `initrd`. It is a `cpio` archive of the initial file system that gets loaded into memory during the PetaLinux startup process. The Linux kernel mounts it as `RootFS` and starts the initialization process.

This section describes how to configure INITRAMFS boot.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have created a new PetaLinux project (refer to section Create New PetaLinux Project) and imported the hardware platform (refer to section Import Hardware Configuration).

Steps to Configure INITRAMFS Boot

1. Set the RootFS type to `INITRAMFS`. Please refer to Root file system Type Configuration.
2. Build the system image. Please refer to Build System Image.
3. Use one of the following methods to boot the system image.
    - Boot a PetaLinux Image on QEMU, refer to section Boot a PetaLinux Image on QEMU.
    - Boot a PetaLinux Image on Hardware with SD Card, refer to section Boot a PetaLinux Image on Hardware with SD Card.
    - Boot a PetaLinux Image on Hardware with JTAG, refer to section Boot a PetaLinux Image on Hardware with JTAG.

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

Configure TFTP Boot

This section describes how to configure the Host and the PetaLinux image for TFTP (Trivial File Transfer Protocol) boot.

TIP: *TFTP boot saves a lot of time because it is much faster than booting through JTAG and you don't have to flash the image for every change in kernel source.*

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have created a new PetaLinux project (refer to section Create New PetaLinux Project) and imported the hardware platform (refer to section Import Hardware Configuration).
- You have TFTP server running on your host.

PetaLinux Configuration and Build System Image

Steps to configure PetaLinux for TFTP boot and build the system image are:

1. Change to root directory of your PetaLinux project.

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

2. Launch the top level system configuration menu.

```
$ petalinux-config
```

3. Select `"Image Packaging Configuration"`.
4. Select `"Copy final images to tftpboot"` and set `"tftpboot directory"`.
5. Save Configuration settings and build system image as explained in Build System Image.

Configuring NFS Boot

One of the most important components of a Linux system is the root file system. A good development root file system provides the developer with all the useful tools that can help him/her on his/her work. Such a root file system can become very big in size, so it is hard to store it in flash memory.

The most convenient thing is to mount the entire root file system from the network, allowing the host system and the target to share the same files. The root file system can be modified quickly and also on the fly (meaning that the file system can be modified while the system is running). The most common way to setup a system like the one described is through NFS.

TIP: *In case of NFS, no manual refresh is needed for new files.*

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have created a new PetaLinux project (refer to section Create New PetaLinux Project) and imported the hardware platform (refer to section Import Hardware Configuration).
- You have Linux file and directory permissions.
- You have NFS server setup on your host.

PetaLinux Configuration and Build System Image

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

1. Change to root directory of your PetaLinux project.

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

2. Launch the top level system configuration menu.

```
$ petalinux-config
```

3. Select Image Packaging Configuration.
4. Select Root filesystem type.
5. Select NFS as the RootFS type.
6. Select `Location of NFS root directory` and set it to `/home/NFSshare`.
7. Exit menuconfig and save configuration settings. The bootargs in the auto generateid DTSI will be updated with the PetaLinux loading rootfs from SD card default settings. You can check `"subsystems/linux/configs/device-tree/system-conf.dtsi"`.
8. Build the system image. Please refer to section Build System Image.

Booting with NFS

In case of NFS Boot, RootFS is mounted through the NFS. But bootloader (fsbl, bitstream, u-boot) and kernel can be downloaded using various methods as mentioned below.

1. JTAG: In this case, bootloader and kernel will be downloaded on to the target through JTAG. Please refer to Boot a PetaLinux Image on Hardware with JTAG.

TIP: *If you want to make use of prebuilt capability to boot with JTAG, package images into prebuilt directory. Please refer to Package Prebuilt Image.*

2. TFTPBOOT: In this case, bootloader will be downloaded through JTAG and kernel will be downloaded on to the target through TFTPB00T. Please refer to Boot a PetaLinux Image on Hardware with TFTP.
3. SD card: In this case, bootloader (BOOT.BIN) and kernel image (image.ub) will be copied to SD card and will be downloaded from SD card. Please refer to Boot a PetaLinux Image on Hardware with SD Card.

Configuring SD Card ext filesystem Boot

This section describes how to configure SD Card ext filesystem boot.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have created a new PetaLinux project (refer to section Create New PetaLinux Project) and imported the hardware platform (refer to section Import Hardware Configuration).
- An SD memory card with at least 4 GB of storage space. It is recommended to use a card with speed-grade 6 or higher to achieve optimal file transfer performance.

Preparing the SD card

Steps to prepare the SD card for PetaLinux SD card ext filesystem boot:

1. The SD card is formatted with two partitions using a partition editor such as `gparted`.
2. The first partition should be at least 40MB in size and formatted as a `FAT32` filesystem. Ensure that there is 4MB of free space preceding the partition. The first partition will contain the bootloader, devicetree and kernel images. Label this partition as `BOOT`.
3. The second partition should be formatted as an `ext4` filesystem and can take up the remaining space on the SD card. This partition will store the system root filesystem. Label this partition as `rootfs`.

TIP: *For optimal performance make sure that the SD card partitions are 4MB aligned.*

PetaLinux Configuration and Build System Image

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

1. Change to root directory of your PetaLinux project.

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

2. Launch top level system configuration menu.

```
$ petalinux-config
```

3. Select Image Packaging Configuration.
4. Select Root filesystem type.
5. Select `SD card` as the RootFS type.

6. Exit menuconfig and save configuration settings. The bootargs in the auto generateid DTSI will be updated with the PetaLinux loading rootfs from SD card default settings. You can check "`subsystems/linux/configs/device-tree/system-conf.dtsi`".
7. Build PetaLinux images. Please refer to section Build System Image.
8. Generate boot image. Please refer to section Generate Boot Image for Zynq family devices.
9. Generate the `rootfs.cpio` image. If you select `SD card` as the RootFS type, `petalinux-build` will not generate the `rootfs.cpio` image. You will need to run `petalinux-package` as follows to generate it:

```
$ petalinux-package --image -c rootfs --format initramfs
```

   The generated `rootfs.cpio` will be in `images/linux/` directory.

Copying Image Files

This section explains how to copy image files to SD card partitions.

1. Change to root directory of your PetaLinux project.

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

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

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

3. Copy `rootfs.cpio` file to `rootfs` partition of SD card and extract the file system. .

```
$ cp images/linux/rootfs.cpio /media/rootfs/
$ cd /media/rootfs
$ sudo pax -rvf rootfs.cpio
```

In order to boot this SD card ext image, refer to section Boot a PetaLinux Image on Hardware with SD Card.

Troubleshooting

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

Including Prebuilt Applications

If an application is developed outside PetaLinux, e.g. through Xilinx SDK, you may just want to add the application binary in the PetaLinux root file system. In this case, an application template is created to allow copying of the existing content to target filesystem.

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

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. Please refer to section Import Hardware Configuration for more information.

Steps to Include Prebuilt Applications

If your prebuilt application name is myapp, including this into PetaLinux root file system is explained in following steps.

1. Ensure that the pre-compiled code has been compiled for your PetaLinux target architecture (e.g. MicroBlaze, ARM etc.).
2. Create an application with the following command.

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

3. Edit the Makefile and add the following line under the `install` section.

```
$ (TARGETINST) -d data/myapp /bin/myapp
```

4. Change to the newly created application directory.

```
$ cd <plnx-proj-root>/components/apps/myapp/data
```

5. Remove existing `myapp` app and copy the prebuilt `myapp`.

```
$ rm myapp
$ cp <path-to-prebuilt-app> .
```

IMPORTANT: *You need to ensure that the binary data being installed into the target file system by an install template application is compatible with the underlying hardware implementation of your system.*

Including Prebuilt Libraries

If you have pre-compiled binary libraries (e.g. Qt), you may just want to add the library binary into PetaLinux root file system.

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

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. Please refer to section Import Hardware Configuration for more information.

Steps to Include Prebuilt Libraries

If your prebuilt library name is mylib, including this into PetaLinux root file system is explained in following steps.

1. Ensure that the pre-compiled library has been compiled for your PetaLinux target architecture (e.g. MicroBlaze, ARM etc.).
2. To create a library project, use the following command.

```
$ petalinux-create -t libs --template install --name mylib --enable
```

3. Change to the newly created library directory.

```
$ cd <plnx-proj-root>/components/libs/mylib
```

4. Place the pre-built library `mylib`.

```
$ cp <path-to-prebuilt-lib> .
```

5. Edit the Makefile for the new library located inside `<plnx-proj-root>/components/libs/mylib/` and add the following line under the `install` section.

```
$(TARGETINST) -d mylib.so /lib/mylib.so
```

Including Prebuilt Modules

If you have pre-compiled kernel modules, you may just want to add the module into PetaLinux root file system.

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

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. Please refer to section Import Hardware Configuration for more information.

Steps to Include Prebuilt Modules

If your prebuilt module name is mymodule, including this into PetaLinux root file system is explained in following steps.

1. Ensure that the pre-compiled kernel module has been compiled for your PetaLinux target architecture (e.g. MicroBlaze, ARM etc.).
2. To create a module project, use the following command.

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

3. Change to the newly created module directory.

```
$ cd <plnx-proj-root>/components/modules/mymodule
```

4. Place the pre-built library `mymodule`.

```
$ cp <path-to-prebuilt-module> .
```

5. Modify the Makefile for the new module located inside `<plnx-proj-root>/components/modules/mymodule/`.
   - Ensure that the `clean:` and `modules:` section of the Makefile are empty.
   - Change the `install:` section of the Makefile to copy the pre-built module as follows.

```
$(TARGETINST) -d mymodule.ko /lib/modules/<KERNELRELEASE>/extra/mymodule.ko
```

Adding Custom Applications

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

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. Please refer to section Import Hardware Configuration for more information.

Steps to Add Custom Applications

The basic steps are as follows:

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

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

   e.g., to create an user application called myapp in C (the default):

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

   or:

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

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

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

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

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

   You can use `-h` or `--help` to see the usage of the `petalinux-create -t apps`. The new application created can be found in the "`<plnx-proj-root>/components/apps/myapp`" directory.

2. Change to the newly created application directory.

   ```
   $ cd <plnx-proj-root>/components/apps/myapp
   ```

You will see the following PetaLinux template-generated files:

Template	Description
Kconfig	Configuration file template - this file controls the integration of your application into the PetaLinux menu configuration system. It also allows you to add configuration options for your application to control how it is built or installed.
Makefile	Compilation file template - this is a basic Makefile containing targets to build and install your application into the root filesystem. This file needs to be modified when you add additional source code files to your project.
README	A file to introduce how to build the user application.
`myapp.c` for C; `myapp.cpp` for C++	Simple application program in either C or C++, depending upon your choice.

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

Adding Custom Libraries

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

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. Please refer to section Import Hardware Configuration for more information.

Steps to Add Custom Libraries

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

   ```
   $ cd <plnx-proj-root>
   $ petalinux-create -t libs [--template TYPE] --name <user-library-name> --enable
   ```

   e.g., to create a user library called mylib in C (the default):

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

   or:

   ```
   $ petalinux-create -t libs --template c --name mylib --enable
   ```

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

   ```
   $ petalinux-create -t libs --template c++ --name mylib --enable
   ```

   To create a autoconf library template, pass the `--template autoconf` option, as follows:

   ```
   $ petalinux-create -t libs --template autoconf --name mylib --enable
   ```

   You can use `-h` or `--help` to see the usage of the `petalinux-create -t libs`. The new library you have created can be found in the `cd <plnx-proj-root>/components/libs/mylib` directory.

2. Change to the newly created library directory.

   ```
   $ cd <plnx-proj-root>/components/libs/mylib
   ```

You will see the following PetaLinux template-generated files:

Template	Description
Kconfig.N	Configuration file template - this file controls the integration of your library into the PetaLinux menu configuration system. It also allows you to add configuration options for your library to control how it is built or installed. `".N"` suggests the priority of the library. `"1"` is the hightest priority and `"11"` is the lowest. The higher priority libraries are built first than the lower priority ones. When you use `petalinux-create` to create the library, you can use option `"--priority N"` to specify the priority of the birary. It is `"7"` by default.
Makefile	Compilation file template - this is a basic Makefile containing targets to build and install your library into the root filesystem. This file needs to be modified when you add additional source code files to your project.
README	A file to introduce how to build the user library.
`libmylib.c` for C; `libmylib.cpp` for C++	Simple library in either C or C++, depending upon your choice.

3. `libmylib.c`/`libmylib.cpp` file can be edited or replaced with the real source code for your library. Later if you want to modify your custom user library, you should edit this file.

Adding Custom Modules

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

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. Please refer to section Import Hardware Configuration for more information.

Steps to Add Custom Modules

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

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

   e.g., to create a user module called mymodule in C (the default):

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

   or:

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

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

2. Change to the newly created module directory.

   ```
   $ cd <plnx-proj-root>/components/modules/mymodule
   ```

   You will see the following PetaLinux template-generated files:

Template	Description
Makefile	Compilation file template - this is a basic Makefile containing targets to build and install your module into the root filesystem. This file needs to be modified when you add additional source code files to your project.
README	A file to introduce how to build the user module.
`mymodule.c`	Simple kernel module in C.

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

Building User Applications

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

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have included pre-compiled applications to PetaLinux root file system (refer to section Including Prebuilt Applications) OR added custom applications to PetaLinux root file system (refer to section Adding Custom Applications).

Steps to Build User Applications

Running `petalinux-build` in the project directory `"<plnx-proj-root>"` will rebuild the system image including the selected user application `myapp`. (The output directory for this build process is `"<plnx-proj-root>/build/linux/rootfs/apps/myapp"`)

```
$ petalinux-build
```

To build myapp into an existing system image:

```
$ cd <plnx-proj-root>
$ petalinux-build -c rootfs -x do_gen_sysroot
$ petalinux-build -c rootfs/myapp
$ petalinux-build -x package
```

NOTE: *`do_gen_sysroot` is to generate the sysroot based on the selected prebuilt packages options from the menuconfig. You don't have to always run `do_gen_sysroot` before building the application, but you need to run it at least once before you build the application.*

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

- *To clean the selected user application:*

```
$ petalinux-build -c rootfs/myapp -x clean
```

- *To rebuild the selected user application:*

```
$ petalinux-build -c rootfs/myapp -x build
```

This will just compile the application, the compiled executable files will be in `<plnx-proj-root>/build/linux/rootfs/apps/myapp` directory.

- *To install the selected user application:*

```
$ petalinux-build -c rootfs/myapp -x install
```

This will install the application into the target rootfs host copy: `<plnx-proj-root>/build/linux/rootfs/targetroot/`.

Testing User Application

This section describes how to test a user application.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have built and installed pre-compiled/custom user applications. Please refer to section Building User Applications.

Steps to Test User Application

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

   ```
   # ls /bin
   ```

   Unless you have changed the location of user application through its Makefile, the user application will be put in to "/bin" directory.
3. Run your user application on the target system console, e.g., to run user application myapp:

   ```
   # myapp
   ```

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

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

User Application Sharing between PetaLinux Projects

This section describes how to share user application between PetaLinux projects.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have included pre-compiled applications to PetaLinux root file system (refer to section Including Prebuilt Applications) OR added custom applications to PetaLinux root file system (refer to section Adding Custom Applications).
- You have created a new PetaLinux project. Please refer to section Create New PetaLinux Project.

Steps to Share Application between PetaLinux Projects

The user can share the application source between the PetaLinux projects. If you have created the application `myapp` in `<plnx-proj-root>` and wish to share the application source with `<plnx-proj-root1>`, the steps are as follows:

- Move the application outside the project to `<extern_search_path>/apps/myapp`.
- Inside your <plnx-proj-root1>, run the following command.

```
$ petalinux-config --searchpath --prepend <extern_search_path>
```

- Run the following command.

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

You should be able to see the application `myapp` under the `Apps --->` submenu as follows.

```
[*] fwupgrade  --->
[ ] gpio-demo  --->
[ ] latencystat  --->
[ ] myapp (NEW)  --->
[*] peekpoke  --->
[*] uWeb  --->
```

Building User Libraries

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

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have included pre-compiled libraries to PetaLinux root file system (refer to section Including Prebuilt Libraries) OR added custom libraries to PetaLinux root file system (refer to section Adding Custom Libraries).

Steps to Build User Libraries

Running `petalinux-build` in the project directory "`<plnx-proj-root>`" will rebuild the system image including the selected user library `mylib`. (The output directory for this build process is `<plnx-proj-root>/build/linux/rootfs/libs/mylib`)

```
$ petalinux-build
```

To build mylib into an existing system image:

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

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

- *To clean the selected user library:*

  ```
  $ petalinux-build -c rootfs/mylib -x clean
  ```

- *To rebuild the selected user library:*

  ```
  $ petalinux-build -c rootfs/mylib -x build
  ```

  This will just compile the library, the compiled executable files will be in `<plnx-proj-root>/build/linux/rootfs/libs/mylib` directory.

- *To install the selected user library:*

  ```
  $ petalinux-build -c rootfs/mylib -x install
  ```

  This will install the library into the target rootfs host copy: `<plnx-proj-root>/build/linux/rootfs/targetroot/lib/`.

Building User Modules

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

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have included pre-compiled applications to PetaLinux root file system (refer to section Including Prebuilt Modules) OR added custom modules to PetaLinux root file system (refer to section Adding Custom Modules).

Steps to Build User Modules

Running `petalinux-build` in the project directory "`<plnx-proj-root>`" will rebuild the system image including the selected user module `mymodule`. (The output directory for this build process is `<plnx-proj-root>/build/linux/rootfs/modules/mymodule`)

```
$ petalinux-build
```

To build mymodule into an existing system image:

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

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

- *To clean the selected user module:*

  ```
  $ petalinux-build -c rootfs/mymodule -x clean
  ```

- *To rebuild the selected user module:*

  ```
  $ petalinux-build -c rootfs/mymodule -x build
  ```

  This will just compile the module, the compiled executable files will be in `<plnx-proj-root>/build/linux/rootfs/modules/mymodule` directory.

- *To install the selected user module:*

  ```
  $ petalinux-build -c rootfs/mymodule -x install
  ```

  This will install the module into the target rootfs host copy: `<plnx-proj-root>/build/linux/rootfs/targetroot/lib/modules/<kernel-version>-xilinx/extra/`.

PetaLinux Auto Login

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

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. Please refer to section Import Hardware Configuration for more information.

Steps for PetaLinux Auto Login

1. Create an application called `autologin` using the following command.

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

2. Change to the newly created `autologin` application directory.

```
$ cd <plnx-proj-root>/components/apps/autologin
```

3. Edit the `autologin.c` file and copy the following code. This code replaces the normal 'login' program and enables auto login at bootup. Note: The security implications of this simple auto login choice is left for the user to consider.

```
#include <unistd.h>
#include <stdio.h>
int main()
{
        execlp( "login", "login", "-f", "root", 0);
}
```

4. Modify the Makefile as follows.

    - Change the `install:` section of the Makefile to copy autologin app to `/etc/init.d/` and create a symbolic link to `/etc/rc5.d/` as follows. These changes will ensure that autologin app will execute at system startup.

```
$(TARGETINST) -d -p 0755 autologin /etc/init.d/autologin
$(TARGETINST) -s /etc/init.d/autologin /etc/rc5.d/S99autologin
```

Application Auto Run at Startup

This section explains how to add applications that run automatically at system startup.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. Please refer to section Import Hardware Configuration for more information.

Steps for Application Auto Run at Startup

If you have prebuilt/newly created custom user application `mystartup` located in your PetaLinux project at `<plnx-proj-root>/components/apps/`, you may want to execute it at system startup. The steps to enable that are:

TIP:

- *If you have prebuilt application and you have not included in PetaLinux Root file system, please refer to Including Prebuilt Applications.*
- *If you want to create custom application and install it in PetaLinux Root file system, please refer to Adding Custom Applications.*
- *If your auto run application is a blocking application which will never exit, launch this application as a deamon.*

1. Change to the application directory.

```
$ cd <plnx-proj-root>/components/apps/mystartup
```

2. Change the `install:` section of the Makefile to copy `mystartup` app to `/etc/init.d/` and create a symbolic link to `/etc/rc5.d/` as follows. These changes will make sure that mystartup app will execute at system startup.

```
$(TARGETINST) -d -p 0755 mystartup /etc/init.d/mystartup
$(TARGETINST) -s /etc/init.d/mystartup /etc/rc5.d/S99mystartup
```

Debugging the Linux Kernel in QEMU

This section describes how to debug the Linux Kernel inside QEMU, using the GDB debugger. Please note that this funciton is only tested with Zynq family devices platform.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have built PetaLinux system image. Please refer to section Build System Image for more information.

Steps to Debug the Linux Kernel in QEMU

1. Launch QEMU with the currently built Linux by running the following command:

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

2. watch the qemu console, you should see the details of the QEMU command, please get the GDB TCP port from `-gdb tcp:<TCP_PORT>`.

3. Open another command console (ensuring the PetaLinux settings script has been sourced), and change to the Linux directory:

```
$ cd "<plnx-proj-root>/images/linux"
```

4. Start GDB on the `vmlinux` kernel image in command mode:

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

   You should see the gdb prompt. e.g.:

```
GNU gdb (Sourcery CodeBench Lite 2013.11-53) 7.6.50.20130726-cvs
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-pc-linux-gnu --target=arm-xilinx-linux-gnueabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://sourcery.mentor.com/GNUToolchain/>.
```

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

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

6. To let QEMU continue execution:

```
(gdb) continue
```

7. You can use `Ctrl+C` to interrupt the kernel and get back the GDB prompt.
8. You can set break points and run other GDB commands to debug the kernel.

WARNING: *If another process is using port 9000, `petalinux-boot` will attempt to use a different port. Look at the output of `petalinux-boot` to determine what port was used. In the following example port 9001 was used.*

```
INFO: qemu-system-arm ... -gdb tcp::9001 ...
```

TIP: *It may be helpful to enable kernel debugging in the kernel configuration menu (`petalinux-config --kernel` > **Kernel hacking** > **Kernel debugging**), so that kernel debug symbols are present in the image.*

Troubleshooting

This section describes some common issues you may experience while debugging the Linux kernel in QEMU.

Problem/Error Message	Description and Solution
(gdb) target remote W.X.Y.Z:9000 :9000: Connection refused.	***Problem Description:*** GDB failed to attach the QEMU target. This is most likely because the port `9000` is not the one QEMU is using. ***Solution:*** 1. Check your QEMU console to make sure QEMU is running. 2. Watch the Linux host command line console. It will show the full QEMU commands, you should be able to see which port is used by QEMU.

Debugging Applications with TCF Agent

This section talks about debugging user applications with the Eclipse TCF (Target Communication Framework) Agent. This section describes the basic debugging procedure for zynq user application `myapp`.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- Working knowledge with Xilinx Software Development Kit (XSDK) tool.
- The Vivado Tools Working Environment is properly set. Please refer to section Vivado Tools Working Environment.
- You have created a user application and built the system image including the selected user application. Please refer to section Building User Applications.

Preparing the build system for debugging

1. Change to the project directory:

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

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

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

3. Scroll down the linux/rootfs Configuration menu to `Filesystem Packages`, followed by the `base` sub-menu:

```
[ ] Advanced Package Selection
[*] base-system-default
    base  --->

    base/shell  --->
    console/network  --->
    console/utils  --->
    devel  --->
    doc  --->
    libs  --->
    libs/network  --->
```

4. Select `base --->` submenu, and then click into `tcf-agent --->` submenu:

```
base-files  --->
base-passwd  --->
busybox  --->
cryptodev-linux  --->
e2fsprogs  --->
external-xilinx-toolchain  --->
i2c-tools  --->
init-ifupdown  --->
initscripts  --->
iproute2  --->
iptables  --->
kmod  --->
libaio  --->
libffi  --->
m4  --->
modutils-initscripts  --->
mtd-utils  --->
net-snmp  --->
net-tools  --->
netbase  --->
opkg-utils  --->
sysfsutils  --->
sysvinit  --->
sysvinit-inittab  --->
tcf-agent  --->

update-rc.d  --->
usbutils  --->
util-linux  --->
```

5. Ensure `tcf-agent` is enabled:

```
[*] tcf-agent
```

6. Select `console/network --->` submenu, and then click into `dropbear --->` submenu. Ensure `"dropbear-openssh-sftp-server"` is enabled.

```
-*- dropbear
[*] dropbear-openssh-sftp-server
```

7. Exit the menu and select <Yes> to save the configuration.
8. Rebuild the target system image including `myapp`. Please refer to section Build System Image.

Performing a Debug Session

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

```
Starting tcf-agent: OK
```

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

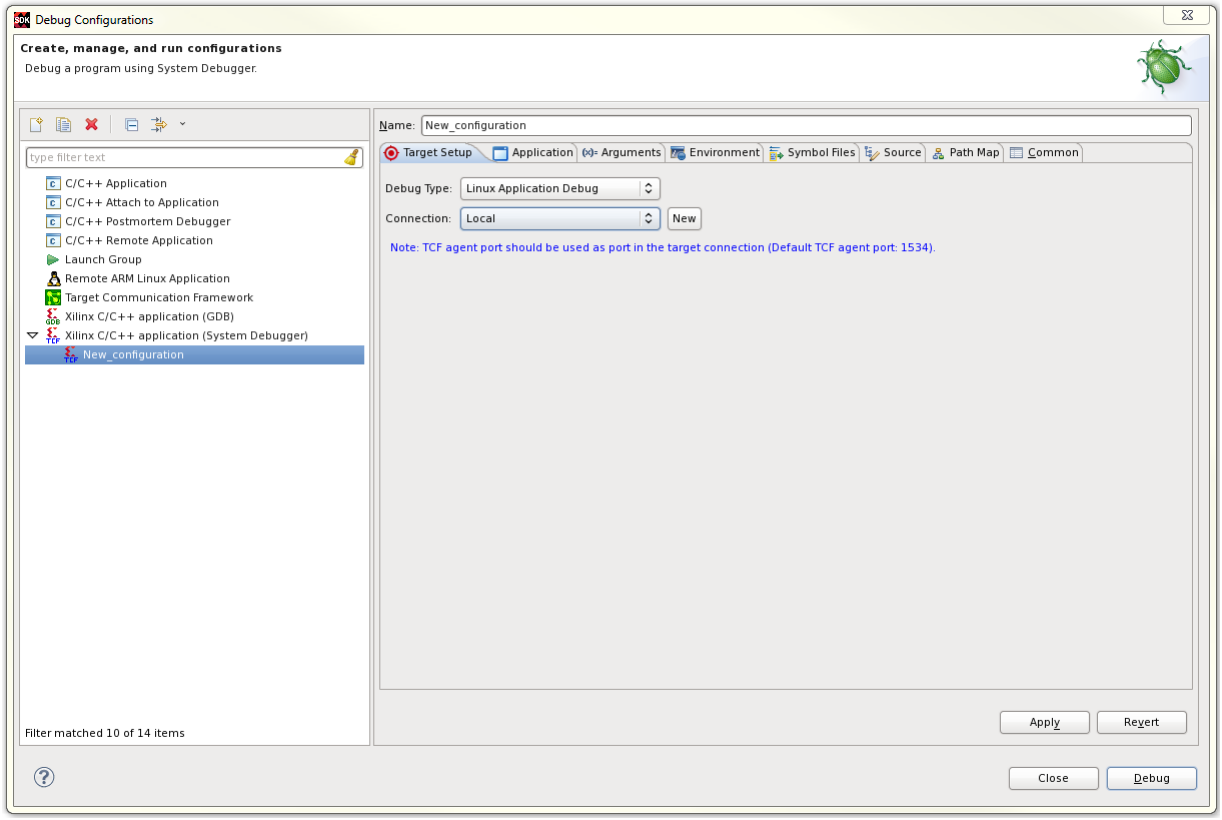

Figure 6: XSDK Debug Configurations

10. The `Debug Type` should be set to `Linux Application Debug`.

11. Select the New option to enter the Connection details.

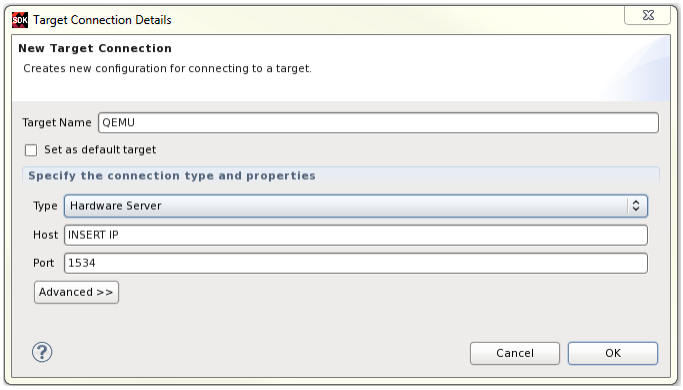

Figure 7: XSDK Debug New Target Configuration

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

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

14. Switch to the `Application Tab`.

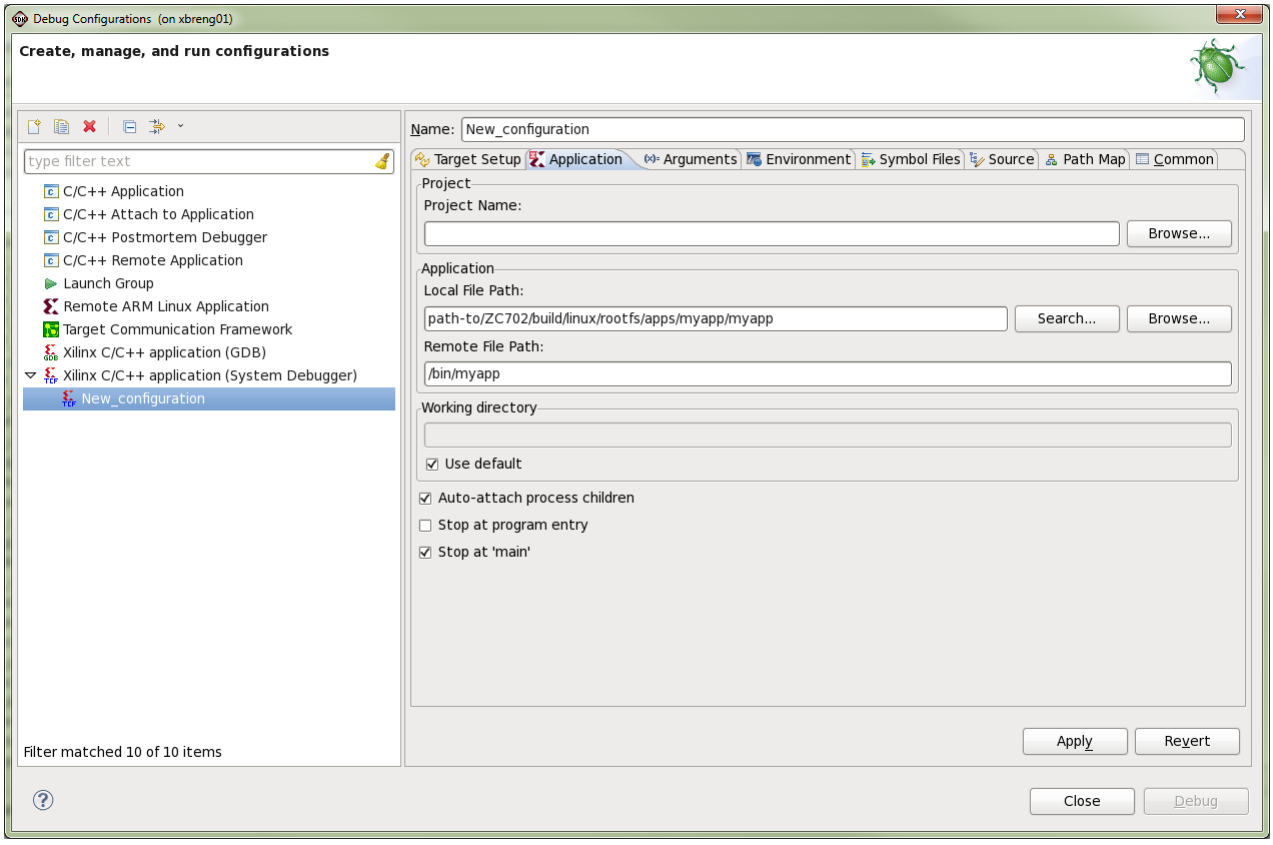

Figure 8: XSDK Debug Configurations

15. Enter the `Local File Path` to your compiled application in the project directory,

    e.g `"<plnx-proj-root>/build/linux/rootfs/apps/myapp/myapp"`

16. The `Remote File Path` on the target file system should be the location where the application can be found, e.g `"/bin/myapp"`

17. Select Debug to Apply the configuration and begin the Debug session. (If asked to switch to Debug Perspective, accept).

18. Standard XSDK debug flow is ready to start:

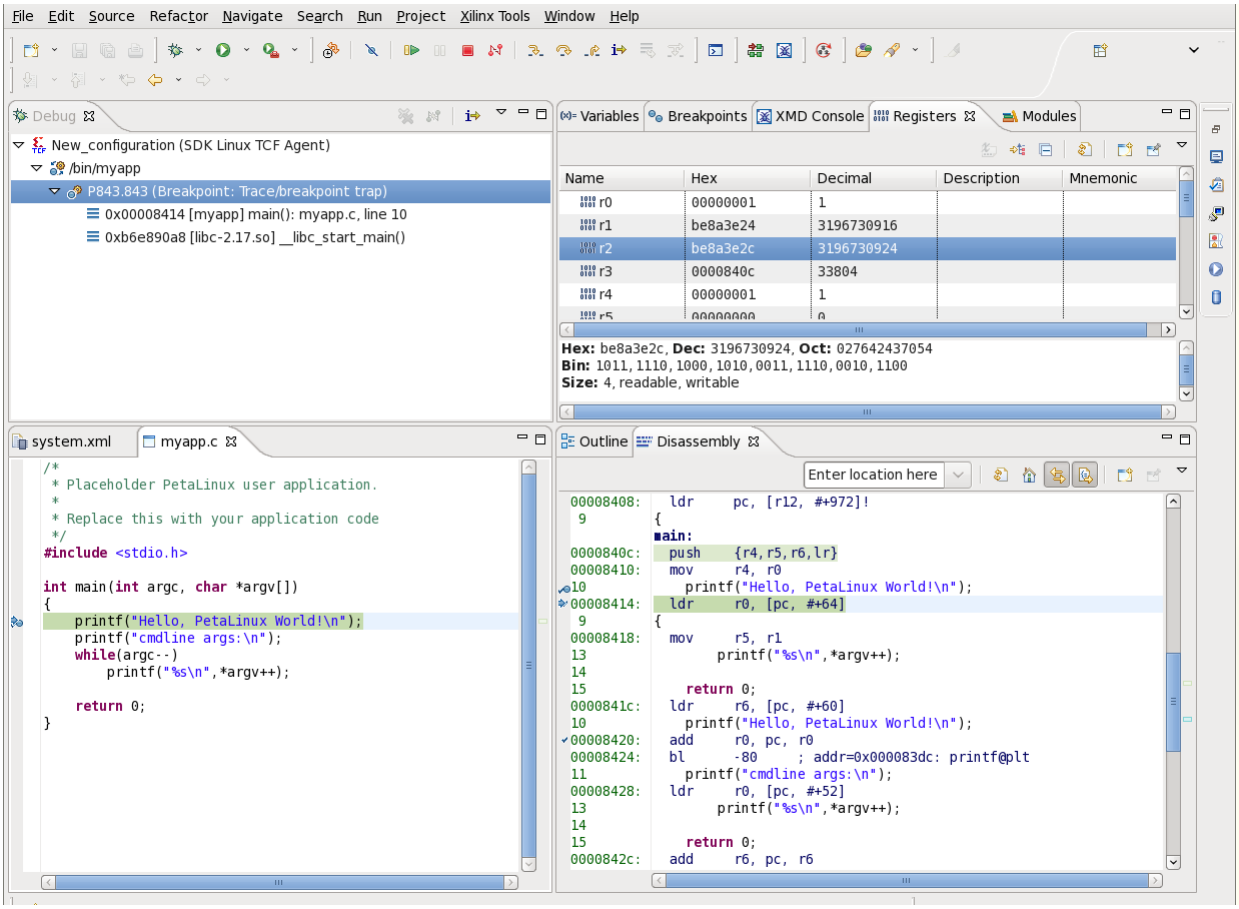

Figure 9: XSDK Debug

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

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

Debugging Zynq UltraScale+ MPSoC Applications with GDB

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

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- The Vivado Tools Working Environment is properly set. Please refer to section Vivado Tools Working Environment.
- You have created a user application and built the system image including the selected user application. Please refer to section Building User Applications.

Preparing the build system for debugging

1. Change the user application Makefile so that compiler optimizations are disabled. Compiler optimizations make debugging difficult since the compiler can re-order or remove instructions that do not impact the program result.

   Here is an example taken from a changed Makefile:

   ```
   ...
   include apps.common.mk

   CFLAGS += -O0

   APP = myapp
   ...
   ```

2. Change to the project directory:

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

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

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

4. Scroll down the linux/rootfs Configuration menu to **Debugging**:

   ```
   Filesystem Packages  --->
   Libs  --->
   Apps  --->
   Modules  --->
   PetaLinux RootFS Settings  --->
   Debugging  --->
   ```

5. Select the **Debugging** sub-menu and ensure that `build debuggable applications` is selected:

```
[*] build debuggable applications
```

6. Exit the **Debugging** sub-menu, and select the `Filesystem Packages`, followed by the `base` sub-menu:

```
[ ] Advanced Package Selection
[*] base-system-default
    base  --->

    base/shell  --->
    console/network  --->
    console/utils  --->
    devel  --->
    doc  --->
    libs  --->
    libs/network  --->
```

7. Select the `external-xilinx-toolchain` sub-menu option, and ensure `gdbserver` is enabled:

```
[ ] catchsegv
[ ] eglibc-extra-nss
[ ] eglibc-pcprofile
[ ] eglibc-utils
[*] gdbserver

[ ] ldd
-*- libc6
[ ] libcidn1
-*- libgcc-s1
[ ] libmemusage
[ ] libsegfault
[ ] libsotruss
[ ] libstdc++6
[ ] libthread-db1
[ ] linux-libc-headers
[ ] nscd
[ ] sln
```

8. Exit the menu and select <Yes> to save the configuration.
9. Rebuild the target system image. Please refer to section Build System Image.

Performing a Debug Session

1. Boot your board (or QEMU) with the new image created above.
2. Run `gdbserver` with the user application on the target system console (set to listening on port 1534):

```
root@Xilinx-KC705-AXI-full-2016.2:~# gdbserver host:1534 /bin/myapp
Process /bin/myapp created; pid = 73
Listening on port 1534
```

   1534 is the `gdbserver` port - it can be any unused port number

3. On the workstation, navigate to the compiled user application's directory:

```
$ cd <plnx-proj-root>/build/linux/rootfs/apps/myapp
```

4. Run GDB client.

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

5. The GDB console will start:

```
...
GNU gdb (crosstool-NG 1.18.0) 7.6.0.20130721-cvs
...
(gdb)
```

6. In the GDB console, connect to the target machine using the command:

   - Use the IP address of the target system, e.g.: `192.168.0.10`. If you are not sure about the IP address, run `ifconfig` on the target console to check.
   - Use the port `1534`. If you chose a different gdbserver port number in the earlier step, use that value instead.

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

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

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

```
Remote Debugging from host 192.168.0.9
```

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

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

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

```
(gdb) continue
Continuing.

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

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

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

10. Try the `step`, `next` and `continue` commands. Breakpoints can be set and removed using the `break` command. More information on the commands can be obtained using the GDB console `help` command.
11. When the program finishes, the GDB server application on the target system will exit. Here is an example of messages shown on the console:

```
Hello, PetaLinux World!
cmdline args:
/bin/myapp

Child exited with status 0
GDBserver exiting
root@Xilinx-KC705-AXI-full-2016.2:~#
```

TIP: *A `.gdbinit` file will be automatically created, to setup paths to libraries. You may add your own GDB initialization commands at the end of this file.*

Going Further With GDB

For more information on general usage of GDB, please refer to the GDB project documentation:

- http://www.gnu.org/software/gdb/documentation

Troubleshooting

This section describes some common issues you may experience while debugging applications with GDB.

Problem/Error Message	Description and Solution
GDB error message: `<IP Address>:<port>: Connection refused. GDB cannot connect to the target board using <IP>:<port>`	***Problem Description:*** This error message indicates that the GDB client failed to connect to the GDB server. ***Solution:*** 1. Check whether the `gdbserver` is running on the target system. 2. Check whether there is another GDB client already connected to the GDB server. This can be done by looking at the target console. If you can see: `Remote Debugging from host <IP>` it means there is another GDB client connecting to the server. 3. Check whether the IP address and the port are correctly set.

Debugging MicroBlaze Applications with GDB

PetaLinux supports debugging MicroBlaze user applications with GDB. This section describes the basic debugging procedure.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- The Vivado Tools Working Environment is properly set. Please refer to section Vivado Tools Working Environment.
- You have created a user application and built the system image including the selected user application. Please refer to section Building User Applications.

Preparing the build system for debugging

1. Change the user application Makefile so that compiler optimizations are disabled. Compiler optimizations make debugging difficult since the compiler can re-order or remove instructions that do not impact the program result.

   Here is an example taken from a changed Makefile:

   ```
   ...
   include apps.common.mk

   CFLAGS += -O0

   APP = myapp
   ...
   ```

2. Change to the project directory:

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

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

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

4. Scroll down the linux/rootfs Configuration menu to **Debugging**:

   ```
   Filesystem Packages  --->
   Libs  --->
   Apps  --->
   Modules  --->
   PetaLinux RootFS Settings  --->
   Debugging  --->
   ```

5. Select the **Debugging** sub-menu and ensure that `build debuggable applications` is selected:

```
[*] build debuggable applications
```

6. Exit the **Debugging** sub-menu, and select the `Filesystem Packages`, followed by the `devel` sub-menu:

```
    console/network  --->
    console/utils  --->
    devel  --->

    devel/python  --->
    doc --->
    kernel  --->
    libs  --->
```

7. Select the `gdb` sub-menu option, and ensure `gdbserver` is enabled:

```
 [ ] gdb
 [ ] gdb-dbg
 [ ] gdb-dev
 [ ] gdb-lic
 [*] gdbserver
```

8. Exit the menu and select <Yes> to save the configuration.
9. Rebuild the target system image. Please refer to section Build System Image.

Performing a Debug Session

1. Boot your board (or QEMU) with the new image created above.
2. Run `gdbserver` with the user application on the target system console (set to listening on port 1534):

```
root@Xilinx-KC705-AXI-full-2016.2:~# gdbserver host:1534 /bin/myapp
Process /bin/myapp created; pid = 73
Listening on port 1534
```

   1534 is the `gdbserver` port - it can be any unused port number

3. On the workstation, navigate to the compiled user application's directory:

```
$ cd <plnx-proj-root>/build/linux/rootfs/apps/myapp
```

4. Run GDB client.

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

5. The GDB console will start:

```
...
GNU gdb (crosstool-NG 1.18.0) 7.6.0.20130721-cvs
...
(gdb)
```

6. In the GDB console, connect to the target machine using the command:
   - Use the IP address of the target system, e.g.: `192.168.0.10`. If you are not sure about the IP address, run `ifconfig` on the target console to check.
   - Use the port `1534`. If you chose a different gdbserver port number in the earlier step, use that value instead.

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

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

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

```
Remote Debugging from host 192.168.0.9
```

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

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

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

```
(gdb) continue
Continuing.

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

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

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

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

11. When the program finishes, the GDB server application on the target system will exit. Here is an example of messages shown on the console:

```
Hello, PetaLinux World!
cmdline args:
/bin/myapp

Child exited with status 0
GDBserver exiting
root@Xilinx-KC705-AXI-full-2016.2:~#
```

TIP: *A `.gdbinit` file will be automatically created, to setup paths to libraries. You may add your own GDB initialization commands at the end of this file.*

Going Further With GDB

For more information on general usage of GDB, please refer to the GDB project documentation:

- http://www.gnu.org/software/gdb/documentation

Troubleshooting

This section describes some common issues you may experience while debugging MicroBlaze applications with GDB.

Problem/Error Message	Description and Solution
GDB error message: `<IP Address>:<port>: Connection refused. GDB cannot connect to the target board using <IP>:<port>`	***Problem Description:*** This error message indicates that the GDB client failed to connect to the GDB server. ***Solution:*** 1. Check whether the `gdbserver` is running on the target system. 2. Check whether there is another GDB client already connected to the GDB server. This can be done by looking at the target console. If you can see: `Remote Debugging from host <IP>` it means there is another GDB client connecting to the server. 3. Check whether the IP address and the port are correctly set.

Configuring Out-of-tree Build

PetaLinux has the ability to automatically download up-to-date kernel/u-boot source code from a `Git` repository. This section describes how this features works and how it can be used in system-level menu config.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. Please refer to section Import Hardware Configuration for more information.
- Internet connection with `git` access is available.

Steps to Configure out-of-tree Build

Steps to configure `UBOOT/Kernel` out-of-tree build:

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

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

2. Launch the top level system configuration menu.

```
$ petalinux-config
```

3. Select `"linux Components Selection --->"` submenu.
4. - For kernel, select `"kernel () --->"` and select `remote`.

```
( ) xlnx-3.18
(X) remote
```

   - For u-boot, select `"u-boot () --->"` and select `remote`.

```
( ) u-boot-plnx
(X) remote
( ) none
```

5. - For kernel, select `"Remote linux-kernel settings --->"`, select `Remote linux-kernel git URL` and enter git URL for linux-kernel. E.g.:

```
https://github.com/Xilinx/linux-xlnx.git
```

   - For u-boot, select `"Remote u-boot settings --->"`, select `Remote u-boot git URL` and enter git URL for u-boot. E.g.:

```
https://github.com/Xilinx/u-boot-xlnx.git
```

   - Set a git tag as "Remote git TAG/commit ID". E.g. `petalinux-v2016.2-final`
6. Exit the menu, and save your settings.

Using External Kernel and U-boot With PetaLinux

PetaLinux includes kernel source and u-boot source. However, you can build your own kernel and u-boot with PetaLinux.
PetaLinux searches for kernel and u-boot candidates from your PetaLinux project components search path. You can get the search path with "`petalinux-config --searchpath --print`" command:

```
$ petalinux-config --searchpath --print
<plnx-proj-root>/components/:/opt/petalinux-v2016.2-final/components/
```

To make PetaLinux tools detect your kernel and u-boot, you can do the following steps:

- For kernel, create a <plnx-proj-root>/components/linux-kernel/ directory, add your kernel source to the created directory <plnx-proj-root>/components/linux-kernel/<MY-KERNEL>.
- For u-boot, create a <plnx-proj-root>/components/u-boot/ directory, add your kernel source to the created directory: <plnx-proj-root>/components/u-boot/<MY-U-BOOT>.
- run `petalinux-config`, and go into "`linux Components Selection --->`" submenu,
- For kernel, select "`kernel () --->`", it will list your kernel there. E.g.:

```
(X) <MY-KERNEL>
( ) xlnx-3.18
( ) remote
```

- For u-boot, select "`u-boot () --->`", it will list your u-boot there. E.g.:

```
(X) <MY-U-BOOT>
( ) u-boot-plnx
( ) remote
( ) none
```

WARNING: *PetaLinux u-boot auto config is only tested with the u-boot shipped with PetaLinux, it is not guaranteed to work with all versions of u-boot code.*

- Exit the menu, and save your settings.
- Sometimes, the default kernel config may not work with your kernel, in this case, you will need to:
  - cleanup the kernel build with the following command:

    ```
    $ petalinux-build -c kernel -x mrproper
    ```

  - you can run "`petalinux-config -c kernel`" to configure your kernel, or you can use `defconfig` of your kernel with the following command:

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

You can also put your kernel and u-boot outside your project. E.g. you put your kernel and u-boot to "`<EXTERN_SEARCHPATH>/linux-kernel/<MY_KERNEL>`" and "`<EXTERN_SEARCHPATH>/linux-kernel/<MY_U_BOOT>`". You will need to add "`<EXTERN_SEARCHPATH>`" to the project components searchpath:

```
$ petalinux-config --searchpath --prepend <EXTERN_SEARCHPATH>
```

And then when you run `petalinux-config`, you can see your kernel and u-boot from the kernel/u-boot candidate list.

Troubleshooting

This section describes some common issues you may experience while configuring out-of-tree build.

Problem/Error Message	Description and Solution
fatal: The remote end hung up unexpectedly ERROR: Failed to get linux-kernel	***Problem Description:*** This error message indicates that system is unable to download the source code (`Kernel/UBOOT`) using remote git URL and hence can not proceed with `petalinux-build`. ***Solution:*** • Check whether entered remote `git` URL is proper or not. • If above solution does not solve the problem, Cleanup the build with the following command: `$ petalinux-build -x mrproper` Above command will remove following directories. ◦ `<plnx-proj-root>/images/` ◦ `<plnx-proj-root>/build/` Re build the system image. Please refer to section Build System Image.

Devicetree Configuration

This section describes which files are safe to modify for the device tree configuration and how to add new information into the device tree.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. Please refer to section Import Hardware Configuration for more information.

Configuring Devicetree

PetaLinux device tree configuration is associated with following config files that are located at `<plnx-proj-root>/subsystems/linux/configs/device-tree/`.

- pcw.dtsi
- pl.dtsi
- skeleton.dtsi
- system-conf.dtsi
- system-top.dts
- zynq-7000.dtsi, for Zynq-7000
- zynqmp-clk.dtsi, for Zynq UltraScale+ MPSoC
- zynqmp.dtsi, for Zynq UltraScale+ MPSoC

For more details on device-tree files, please refer to Appendix A PetaLinux Project Structure.

WARNING: *DTSI files listed above `*.dtsi` are automatically generated; user is not supposed to edit these files.*

If the user wishes to add information e.g.Ethernet PHY information, this should be included in the `system-top.dts` file. In this case, device tree should include the information relevant for your specific platform as information (e.g.Ethernet PHY information) is board level and board specific.

NOTE: *The need for this manual interaction is because some information is "`board level`" and the tools do not have a way of predicting what should be here. Please refer to the Linux kernel Device Tree bindings documents (Documentation/devicetree/bindings from the root of the kernel source) for the details of bindings of each device.*

An example of a well-formed Device-tree node for the `system-top.dts` is below.

```
/dts-v1/;
/include/ "system-conf.dtsi"
/ {
};
&gem0 {
        phy-handle = <&phy0>;
        ps7_ethernet_0_mdio: mdio {
                phy0: phy@7 {
                        compatible = "marvell,88e1116r";
                        device_type = "ethernet-phy";
                        reg = <7>;
                };
        };
};
```

IMPORTANT: *Ensure that the device tree node name, MDIO address, and compatible strings correspond to the naming conventions used in your specific system.*

U-Boot Configuration

This section describes which files are safe to modify for the U-Boot configuration and discusses about the U-Boot CONFIG_ options/settings.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. Please refer to section Import Hardware Configuration for more information.

Configuring U-Boot

U-Boot (Universal Bootloader) Configuration is usually done using C pre-processor defines:

- Configuration _OPTIONS_:

  These are selectable by the user and have names beginning with "CONFIG_".

- Configuration _SETTINGS_:

  These depend on the hardware etc. They have names beginning with "CONFIG_SYS_".

TIP: *Detailed documentation on U-Boot can be found at Denx U-Boot Guide. README can be found at Denx U-Boot README, for detailed explanation on CONFIG_ options/settings.*

PetaLinux u-boot configuration is associated with following configuration files which are located at <plnx-proj-root>/subsystems/linux/configs/u-boot/.

- config.mk
- platform-auto.h
- platform-top.h

WARNING: *config.mk and platform-auto.h files are automatically generated; user is not supposed to edit these files.*

PetaLinux does not currently automate U-Boot configuration with respect to CONFIG_ options/settings. The user can add these CONFIG_ options/settings into platform-top.h file.

Steps to add CONFIG_ option (e.g.CONFIG_CMD_MEMTEST) to platform-top.h:

- Change into the root directory of your PetaLinux project.

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

- Open the file `platform-top.h`

```
$ vi subsystems/linux/configs/u-boot/platform-top.h
```

- If you want to add `CONFIG_CMD_MEMTEST` option, add the following line to the file. Save the changes.

```
#define CONFIG_CMD_MEMTEST
```

TIP: *Defining `CONFIG_CMD_MEMTEST` enables the Monitor Command "`mtest`", which is used for simple RAM test.*

- Build the U-Boot image.

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

- Generate `BOOT.BIN` using the following command.

```
$ petalinux-package --boot --fsbl <FSBL image> --fpga <FPGA bitstream> --u-boot
```

- Boot the image either on hardware or QEMU and stop at `U-Boot` stage.
- Enter the "`mtest`" command in the U-Boot console as follows:

```
U-Boot-PetaLinux> mtest
```

- Output on the U-Boot console should be similar to the following:

```
Testing 00000000 ... 00001000:
Pattern 00000000  Writing...  Reading...Iteration:   2069
```

IMPORTANT: *If `CONFIG_CMD_MEMTEST` is not defined, output on U-Boot console will be as follows:*

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

Appendix A: PetaLinux Project Structure

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

- device tree
- first stage bootloader (optional)
- u-boot (optional)
- Linux kernel
- rootfs. And the rootfs is composed of the following components:
    - prebuilt packages
    - Linux user applications (optional)
    - Linux user libraries (optional)
    - user modules (optional)

A PetaLinux project directory contains configuration files of the project, the Linux subsystem, and the components of the subsystem. The `petalinux-build` command builds the project with those configuration files. Users can run `petalinux-config` to modify them.
Here is an example of a PetaLinux project:

```
<plnx-proj-root>
    |-.petalinux/
    |-hw-description/
    |-config.project
    |-subsystems/
    |    |-linux/
    |    |    |-config
    |    |    |-hw-description/
    |    |    |-configs/
    |    |    |    |-device-tree/
    |    |    |    |    |-pcw.dtsi
    |    |    |    |    |-pl.dtsi
    |    |    |    |    |-skeleton.dtsi
    |    |    |    |    |-system-conf.dtsi
    |    |    |    |    |-system-top.dts
    |    |    |    |    |-zynq-7000.dtsi # for Zynq-7000
    |    |    |    |    |-zynqmp.dtsi # for Zynq UltraScale+ MPSoC
    |    |    |    |    |-zynqmp-clk.dtsi # for Zynq UltraScale+ MPSoC
    |    |    |    |-generic/
    |    |    |    |    |-config
    |    |    |    |-kernel/
    |    |    |    |    |-config
    |    |    |    |-rootfs/
    |    |    |    |    |-config
    |    |    |    |-u-boot/
    |    |    |    |    |-config.mk # For MicroBlaze
    |    |    |    |    |-config
    |    |    |    |    |-platform-auto.h
    |    |    |    |    |-platform-top.h
    |-components/
    |    |-bootloader/
    |    |    |-fs-boot/ | zynq_fsbl/
    |    |-apps/
    |    |    |-myapp/
```

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

"<plnx-proj-root>/subsystems/linux/hw-description/"	Subsystem hardware description exported by Vivado

File/Directory in a PetaLinux Project	Description
"<plnx-proj-root>/subsystems/linux/configs/"	Configuration files for the components of the subsystem
"<plnx-proj-root>/subsystems/linux/configs/kernel/config"	Configuration file used to build the Linux kernel
"<plnx-proj-root>/subsystems/linux/configs/rootfs/config"	Configuration file used to build the rootfs
"<plnx-proj-root>/subsystems/linux/configs/device-tree"	Device tree files used to build device tree. The following files are auto generated by `petalinux-config`: • `pcw.dtsi`: (Zynq family device) PS settings DTS generated based on Vivado PCW configurations. • `pl.dtsi`: PL DTSI. • `skeleton.dtsi`: (Zynq-7000 only) minimum Linux kernel required DTS. • `system-conf.dtsi`: PetaLinux auto generated system DTSI. • `system-top.dts`: PetaLinux tools will not touch this file, user can put their DTS stuff here. • `zynq-7000.dtsi`: (Zynq-7000 only) static DTSI for Zynq-7000. • `zynqmp.dtsi`: (Zynq UltraScale+ MPSoC only) static DTSI for Zynq UltraScale+ MPSoC. • `zynqmp-clk.dtsi`: (Zynq UltraScale+ MPSoC only) static DTSI for Zynq UltraScale+ MPSoC.

File/Directory in a PetaLinux Project	Description
"<plnx-proj-root>/subsystems/linux/configs/u-boot"	u-boot PetaLinux auto config files used to build u-boot. The following files are auto generated by petalinux-config: • config • platform-auto.h platform-top.h will not be modified by any PetaLinux tools and is under full control by user. When u-boot builds, these files will be copied into u-boot build source directory build/linux/u-boot/src/<U_BOOT_SRC>/ as follows: • platform-auto.h and platform-top.h will be copied to include/configs/ directory. • config is the u-boot kconfig file.
"<plnx-proj-root>/components/"	Directory for local components. If you don't have local components, this directory is not required. Components created by petalinux-create will be placed into this directory. You can also manually copy components into this directory. Here is the rule to place a local component: "<plnx-proj-root>/components/<COMPONENT_TYPE>/<COMPONENT>"

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

- "<plnx-proj-root>/build" for the files generated for build.
- "<plnx-proj-root>/images" for the bootable images.

Here is an example:

```
<plnx-proj-root>
    |-.petalinux/
    |-hw-description/
    |-config.project
    |-subsystems/
    |    |-linux/
    |    |    |-config
    |    |    |-hw-description/
    |    |    |-configs/
    |    |    |    |-device-tree/
    |    |    |    |-kernel/
    |    |    |    |-u-boot/
    |    |    |    |-rootfs/
    |-components/
    |    |-apps/
    |    |    |-myapp/
    |    |-bootloader/
    |    |    |-fs-boot/ | zynq_fsbl/
    |-build/
    |    |-build.log
    |    |-config.log
    |    |-linux/
    |    |    |-rootfs/
    |    |    |    |-targetroot/
    |    |    |    |-sys_init/
    |    |    |    |-packages-repo/
    |    |    |    |-stage/
    |    |    |    |-apps/
    |    |    |    |    |-myapp/
    |    |    |-kernel/
    |    |    |-u-boot/
    |    |    |-device-tree/
    |    |    |-bootloader/
    |    |    |-data/
    |    |    |-hw-description/
    |-images/
    |    |-linux/
```

WARNING: *"<plnx-proj-root>/build/" are automatically generated. Do not manually edit files in this directory. Contents in this directory will get updated when you run petalinux-config or petalinux-build.*

"<plnx-proj-root>/images/" are also automatically generated. Files in this directory will get updated when you run petalinux-build.

Build Directory in a PetaLinux Project	Description
"<plnx-proj-root>/build/build.log"	Logfile of the build
"<plnx-proj-root>/build/linux/"	Directory to hold files related to the linux subsystem build

Build Directory in a PetaLinux Project	Description
"<plnx-proj-root>/build/linux/rootfs/"	Directory to hold files related to the rootfs build
"<plnx-proj-root>/build/linux/rootfs/targetroot/"	Target rootfs host copy
"<plnx-proj-root>/build/linux/rootfs/stage/"	Stage directory to hold the libs and header files required to build user apps/libs
"<plnx-proj-root>/build/linux/kernel/"	Directory to hold files related to the kernel build
"<plnx-proj-root>/build/linux/u-boot/"	Directory to hold files related to the u-boot build
"<plnx-proj-root>/build/linux/device-tree/"	Directory to hold files related to the device-tree build
"<plnx-proj-root>/build/linux/bootloader/"	Directory to hold files related to the bootloader build

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

Appendix B: Generating First Stage Bootloader Within Project

This is OPTIONAL. By default, the top level system settings are set to generate the first stage bootloader.

WARNING: *If the user wishes not to have PetaLinux build the FSBL/FS-BOOT, then you will need to manually build it on your own. Else, your system will not boot properly.*

If you had disabled first stage bootloader from menuconfig previously, You can configure the project to build first stage bootloader as follows:

1. Launch top level system settings configuration menu and configure:

```
$ petalinux-config
```

   (a) Click into "linux Components Selection --->" submenu.

   (b) Select "First Stage Bootloader" option.

```
[*] First Stage Bootloader
```

   (c) Exit the menu and save the change.

   This operation will generate the FSBL (First Stage Bootloader) source into components/bootloader/ inside your PetaLinux project root directory if it doesn't already exist. For Zynq UltraScale+ MPSoC, it will be:

```
components/bootloader/zynqmp_fsbl
```

   For Zynq-7000, it will be:

```
components/bootloader/zynq_fsbl
```

   For MicroBlaze, it will be:

```
components/bootloader/fs-boot
```

   FSBL should be in the local project directory.

2. Launch petalinux-build to build the FSBL:

   Build the FSBL when building the project:

```
$ petalinux-build
```

   Build the FSBL only:

```
$ petalinux-build -c bootloader
```

The bootloader ELF file will be installed as `zynqmp_fsbl.elf` for Zynq UltraScale+ MPSoC, `zynq_fsbl.elf` for Zynq-7000 and `fs-boot.elf` for MicroBlaze in `images/linux` inside the project root directory.

TIP: *`zynq_fsbl_bsp`, `zynqmp_fsbl_bsp` will be auto updated when you run `petalinux-config`.*

ATF (Arm Trusted Firmware)

This is for Zynq UltraScale+ MPSoC only. This is `OPTIONAL`. By default, the top level system settings are set to generate the ATF.

WARNING: *If the user wishes not to have PetaLinux build the ATF, then you will need to manually build it on your own. Else, your system will not boot properly.*

You can configure the project to build ATF as follows:

1. Launch top level system settings configuration menu and configure:

```
$ petalinux-config
```

   (a) Click into `"linux Components Selection --->"` submenu.

   (b) Select `"arm-trusted-firmware"` option.

```
arm-trusted-firmware(arm-trusted-firmware)
```

   (c) Exit the menu and save the change.

2. Launch `petalinux-build` to build the FSBL:

   Build the ATF when building the project:

```
$ petalinux-build
```

   Build the ATF only:

```
$ petalinux-build -c arm-trusted-firmware
```

   The ATF ELF file will be installed as `bl31.elf` for Zynq UltraScale+ MPSoC in `images/linux` inside the project root directory.

FS-Boot For MicroBlaze Platform Only

FS-Boot in PetaLinux is a first stage bootloader demo for MicroBlaze platform only. It is to demonstrate how to load images from flash to the memory and jump to it. If you want to try FS-Boot, you will need 8KByes BRAM at least.

FS-Boot supports Parallel flash and SPI flash in standard SPI mode only. If you are using `axi_quad_spi`, it only works with X1 mode.

In order for FS-Boot to know where in the flash should get the image, macro `CONFIG_FS_BOOT_START` needs to be defined. This is done by the PetaLinux tools. PetaLinux tools set this macro automatically from the `boot` partition settings in the menuconfig primary flash partition table settings. For parallel flash, it is the start address of `boot` partition. For SPI flash, it is the start offset of `boot` partition.

The image in the flash requires a wrapper headder followed by a BIN file. FS-Boot gets the target memory location from wrapper. The wrapper needs to contain the following information:

Offset	Description	Value
`0x0`	FS-Boot bootable image magic code	0xb8b40008
`0x4`	BIN image size	User defined
`0x100`	FS-Boot bootable image target memory address	User defined. PetaLinux tools automatically calculate it from the `u-boot text base address offset` from the `Memory Settings` from the menuconfig.
`0x10c`	Where the BIN file start	None

FS-Boot ingores other fields in the wrapper header. PetaLinux tools generate the wrapper header to wrap around the u-boot BIN file.

Appendix C: Auto Config Settings

When you run `petalinux-config`, you will see the `"Auto Config Settings"` submenu. If you click in the submenu, you will see the list of components which PetaLinux can do auto config based on the top level system settings. If a component is selected to enable autoconfig, when `petalinux-config` is run, its config files will be auto updated.

component in the menu	Files impacted when autoconfig is enabled
Device tree	• <plnx-proj-root>/subsystems/linux/configs/device-tree/skeleton.dtsi (Zynq-7000 only) • <plnx-proj-root>/subsystems/linux/configs/device-tree/zynq-7000.dtsi (Zynq-7000 only) • <plnx-proj-root>/subsystems/linux/configs/device-tree/zynqmp.dtsi (Zynq UltraScale+ MPSoC only) • <plnx-proj-root>/subsystems/linux/configs/device-tree/zynqmp-clk.dtsi (Zynq UltraScale+ MPSoC only) • <plnx-proj-root>/subsystems/linux/configs/device-tree/pcw.dtsi (Zynq-7000 and Zynq UltraScale+ MPSoC) • <plnx-proj-root>/subsystems/linux/configs/device-tree/pl.dtsi • <plnx-proj-root>/subsystems/linux/configs/device-tree/system-conf.dtsi
kernel	<plnx-proj-root>/subsystems/linux/configs/kernel/config
rootfs	<plnx-proj-root>/subsystems/linux/configs/rootfs/config
u-boot	• <plnx-proj-root>/subsystems/linux/configs/u-boot/config • <plnx-proj-root>/subsystems/linux/configs/u-boot/platform-auto.h

Appendix D: QEMU Virtual Networking Modes

There are two execution modes in QEMU: non-root (the default) and root (requires sudo or root permission). The difference in the modes relates to virtual network configuration.

In non-root mode QEMU sets up an internal virtual network which restricts network traffic passing from the host and the guest. This works similar to a NAT router. You can not access this network unless you redirect tcp ports.

In root mode QEMU creates a subnet on a virtual ethernet adapter, and relies on a DHCP server on the host system.

The following sections detail how to use the modes, including redirecting the non-root mode so it is accessible from your local host.

Redirecting ports in non-root mode

If running QEMU in the default non-root mode, and you wish to access the internal (virtual) network from your host machine (e.g.to debug with either GDB or TCF Agent), you will need to forward the emulated system ports from inside the QEMU virtual machine to the local machine.
The `petalinux-boot --qemu` command utilises the `--qemu-args` option to perform this redirection.
The following table outlines some example redirection arguments. This is standard QEMU functionality, please refer to the QEMU documentation for more details.

QEMU options switch	Purpose	Accessing guest from host
`-tftp <path-to-directory>`	Sets up a TFTP server at the specified directory, the server is available on the QEMU internal IP address of 10.0.2.2.	
`-redir tcp:10021:10.0.2.15:21`	Redirects port 10021 on the host to port 21 (ftp) in the guest	`host> ftp localhost 10021`
`-redir tcp:10023:10.0.2.15:23`	Redirects port 10023 on the host to port 23 (telnet) in the guest	`host> telnet localhost 10023`
`-redir tcp:10080:10.0.2.15:80`	Redirects port 10080 on the host to port 80 (http) in the guest	Type http://localhost:10080 in the web browser
`-redir tcp:10022:10.0.2.15:22`	Redirects port 10022 on the host to port 22 (ssh) in the guest	Run `ssh -P 10022 localhost` on the host to open a SSH session to the target

The following example shows the command line used to redirect ports:

```
$ petalinux-boot --qemu --kernel --qemu-args "-redir tcp:1534::1534"
```

This document assumes the use of port 1534 for gdbserver and tcf-agent, but it is possible to redirect to any free port. The internal emulated port can also be different from the port on the local machine:

```
$ petalinux-boot --qemu --kernel --qemu-args "-redir tcp:1444::1534"
```

Specifying the QEMU Virtual Subnet

By default, PetaLinux uses `192.168.10.*` as the QEMU virtual subnet in `--root` mode. If it has been used by your local network or other virtual subnet, you may wish to use another subnet. You can configure PetaLinux to use other subnet settings for QEMU by running `petalinux-boot` as follows on the command console:

WARNING: *This feature requires sudo access on the local machine, and must be used with the `--root` option.*

```
$ petalinux-boot --qemu --root --u-boot --subnet <subnet gateway IP>/
    <number of the bits of the subnet mask>
```

e.g., to use subnet `192.168.20.*`:

```
$ petalinux-boot --qemu --root --u-boot --subnet 192.168.20.0/24
```

Appendix E: Xilinx IP models supported by QEMU

The QEMU emulator shipped in PetaLinux tools supports a limited number of Xilinx IP models listed as follows:

- Zynq-7000 ARM Cortex-A9 CPU
- Zynq UltraScale+ MPSoC ARM Cortex-A53 MPCore
- Zynq UltraScale+ MPSoC Cortex-R5
- MicroBlaze CPU (little-endian AXI)
- Xilinx Zynq-7000/Zynq UltraScale+ MPSoC DDR Memory Controller
- Xilinx Zynq UltraScale+ MPSoC DMA Controller
- Xilinx Zynq UltraScale+ MPSoC SD/SDIO Peripheral Controller
- Xilinx Zynq UltraScale+ MPSoC Gigabit Ethernet Controller
- Xilinx Zynq UltraScale+ MPSoC NAND Controller
- Xilinx Zynq UltraScale+ MPSoC UART Controller
- Xilinx Zynq UltraScale+ MPSoC QSPI Controller
- Xilinx Zynq UltraScale+ MPSoC I2C Controller
- Xilinx Zynq UltraScale+ MPSoC USB Controller (Host support only)
- Xilinx Zynq-7000 Triple Timer Counter
- Xilinx Zynq-7000 DMA Controller
- Xilinx Zynq-7000 SD/SDIO Peripheral Controller
- Xilinx Zynq-7000 Gigabit Ethernet Controller
- Xilinx Zynq-7000 USB Controller (Host support only)
- Xilinx Zynq-7000 UART Controller
- Xilinx Zynq-7000 SPI Controller
- Xilinx Zynq-7000 QSPI Controller
- Xilinx Zynq-7000 I2C Controller
- Xilinx AXI Timer and Interrupt controller peripherals
- Xilinx AXI External Memory Controller connected to parallel flash
- Xilinx AXI DMA Controller
- Xilinx AXI Ethernet

- Xilinx AXI Ethernet Lite
- Xilinx AXI UART 16650 and Lite

IMPORTANT: *By default, QEMU will disable any devices for which there is no model available. For this reason it is not possible to use QEMU to test your own customized IP Cores (unless you develop C/C++ models for them according to QEMU standard).*

Appendix F: XEN Zynq Ultrascale+ MPSoC Example

This section details on the XEN Zynq Ultrascale+ MPSoC example. It descripbes how to get Linux to boot as `dom0` on top of `XEN` on Zynq Ultrascale+ MPSoc.

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. Please refer to section Import Hardware Configuration for more information.
- You have created a PetaLinux project from the ZCU102 reference BSP.
  - There are XEN related prebuilts in the `pre-built/linux/images` directory, which are `xen.dtb`, `xen.ub`.
  - There are XEN source code in the `components/apps/xen` directory.
  - There are prebuilt XEN tools and its dependencies tar balls in `components/apps/xen-tools` directory.

Boot prebuilt Linux as `dom0`

1. Copy prebuilt XEN images and Linux Kernel image to your tftp directory so that you can load them from u-boot with tftp.

```
$ cd <plnx-proj-root>
$ cp pre-built/linux/images/xen.dtb <tftpboot>/
$ cp pre-built/linux/images/xen.ub <tftpboot>/
$ cp pre-built/linux/images/Image <tftpboot>/
```

2. Boot prebuilt u-boot image on the board with either jtag boot or boot from SD card.
3. Setup tftp server IP from u-boot

```
U-Boot-PetaLinux> setenv serverip <TFTP SERVERIP>
```

4. Load XEN images and kernel images from u-boot

```
U-Boot-PetaLinux> tftpboot 4000000 xen.dtb
U-Boot-PetaLinux> tftpboot 80000 Image
U-Boot-PetaLinux> tftpboot 6000000 xen.ub
U-Boot-PetaLinux> bootm 6000000 - 4000000
```

Kernel Configuration Requirement

In order to run Linux kernel as dom0, the following options are required to be on:

- CONFIG_XEN
- CONFIG_HVC_DRIVER
- CONFIG_HVC_XEN

The reference PetaLinux project has already enabled them by default.

XEN Device Tree Requirements

You can see the subsystems/linux/configs/device-tree/xen-overlay.dtsi for how the XEN configuration should be in a DTS. Note that DTS for QEMU platform is different from hardware. Use xen-qemu.dts/xen-qemu-overlay.dts when running on QEMU.

The DTS file for XEN, should be the same as for plain Linux, with a few entries added to the chosen node.

The xen,dom0-bootargs corresponds to the Linux Kernel command line. Here is an example:

```
chosen {
        #address-cells = <0x2>;
        #size-cells = <0x1>;
        xen,xen-bootargs = "console=dtuart dtuart=serial0 dom0_mem=512M bootscrub=0 maxcpus=1 timer_slop=0";
        xen,dom0-bootargs = "console=hvc0 earlycon=xen earlyprintk=xen maxcpus=1";
        dom0 {
                compatible = "xen,linux-zimage", "xen,multibootmodule";
                reg = <0x0 0x00080000 0x3100000>;
        };
};
```

If you want to try XEN on QEMU, you will need to disable the other CPUs except CPU0 in the DTS, otherwise, QEMU will run very slow since it is a single threaded application:

```
cpus {
        cpu@1 {
                device_type = "none";
        };
        cpu@2 {
                device_type = "none";
        };
        cpu@3 {
                device_type = "none";
        };
};
```

Rebuild XEN

Assuming your PetaLinux project is created from the Zynq Ultrascale+ MPSoC PetaLinux reference BSP. XEN source code is included in `components/apps/xen` directory. You can rebuild the xen.ub as follows:

- Configure rootfs to build XEN when buildling rootfs

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

- Select xen from the Apps submenu
- Edit DTS to include XEN. Add `/include/ "xen-overlay.dtsi"` to `subsystems/linux/configs/device-tree/system-top.dts`.
- Rebuild PetaLinux:

```
$ petalinux-build
```

- `xen.ub` and the `system.dtb` with the XEN configuration are generated in the `images/linux` directory.

Additional Resources

References

PetaLinux Tools Documentation is available at http://www.xilinx.com/petalinux.