PetaLinux Tools Documentation

Reference Guide

UG1144 (v2017.1) April 5, 2017

XILINX
ALL PROGRAMMABLE™

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/05/2017	2017.1	Updated for PetaLinux Tools 2017.1 release

Table of Contents

Chapter 2: Yocto Features

PetaLinux Tools Documentation

Introduction

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.

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.

The new PetaLinux tool contains the following:

1. Yocto Extensible SDK

   Table 1-1 details the 4 Extensible SDKs installed.

Table 1-1: **Extensible SDKs**

Path	Architecture
`$PETALINUX/components/yocto/source/aarch64`	for Zynq® UltraScale+™ MPSoC
`$PETALINUX/components/yocto/source/arm`	for Zynq
`$PETALINUX/components/yocto/source/microblaze_full`	for MicroBlaze™ full designs
`$PETALINUX/components/yocto/source/microblaze_lite`	for MicroBlaze lite designs

The Yocto extensible SDK (e-SDK) consists of:

a. Layers - This contains all the layers for an architecture. The Yocto e-SDk had core, meta-oe and other popular layers.

Table 1-2: **Layers from Xilinx**

Layer	Recipes
meta-xilinx	Contains recipes of linux kernel, uboot and Arm Trusted Firmware (ATF)
meta-xilinx-tools	Contains recipes of all embeddedsw apps: fsbl, pmufw, fsboot, device-tree
meta-petalinux	Contains distro recipes and package groups petalinux-image-minimal --> minimal feature set petalinux-image-full ---> Full feature set
meta-openamp	Contains openamp recipes and configurations
meta-linaro-toolchain	Contains tool chain recipes for Zynq and ZynqMP

For example, for Zynq UltraScale+ MPSoC:
`$PETALINUX/components/yocto/source/aarch64/layers`

b. sstate-cache - By design, the OpenEmbedded build system builds everything from scratch unless BitBake can determine that parts do not need to be rebuilt. Fundamentally, building from scratch is attractive as it means all parts are built fresh and there is no possibility of stale data causing problems.

The Yocto Project implements shared state code that supports incremental builds. It stores all task intermediate build artifacts and reuses them if there is no change in input tasks, hence reduces the build time

For example: The sstate-cache of Zynq UltraScale+ MPSoC is at:
`$PETALINUX/components/yocto/source/aarch64/sstate-cache`

c. sysroots - This contains sysroot for host and the target

For example: The sysroot of Zynq UltraScale+ MPSoC is at:
`$PETALINUX/components/yocto/source/aarch64/sysroots`

2. Minimal downloads

BitBake checks pre-mirrors before looking upstream for any source files. Pre-mirrors are appropriate when you have shared the directory that is not a directory defined by the DL_DIR variable. A Pre-mirror points to a shared directory that is in tool. All projects of the tool use these pre-mirrors and fetch the source code from them.

The pre-mirror in tool points to: `$PETALINUX/components/yocto/downloads`.

The downloads directory has tar balls of source code for linux kernel, uboot and other minimal utilities.

3. XSCT and tool chains

The PetaLinux tool uses XSCT underneath for all embeddedSW apps. For ZynqMP and Zynq devices, the tool chain from Yocto (meta-linaro-toolchain) is used. For MicroBlaze, SDK tool chain is used.

4. PetaLinux Commands

   This contains all the petalinux commands that you require.

Installation Requirements

The PetaLinux Tools Installation requirements are:

- Minimum workstation requirements:
  - 4 GB RAM (recommended minimum for Xilinx tools)
  - Pentium 4, 2 GHz CPU clock or equivalent (minimum of 4 cores)
  - 100 GB free HDD space
  - Supported OS:
    - RHEL 6.7/6.8 (64-bit)

      Note: Limited/conditional support with workaround: Install devtoolset-2 package on RHEL, Cent OS 6.X. Support is limited to Xilinx PetaLinux Images. Any breakage here (RHEL 6.X + devtools-2,) outside the PetaLinux images is not supported by Xilinx. This will not have any support in 2017.3.

    - RHEL 7.2/7.3 (64-bit)
    - CentOS 7.2/7.3 (64-bit)
    - Ubuntu 16.04.1 (64-bit)
- You need to have root access to perform some operations. The PetaLinux tools needs to be installed as a non-root user.
- PetaLinux requires a number of standard development tools and libraries to be installed on your Linux host workstation. Install the libraries and tools listed in the following table on the host Linux. All of the listed Linux Workstation Environments below have the 32-bit libraries needed by the PetaLinux tool. If any addition tool chains are packages needing 32-bit libs on host are needed, install the same before issuing petalinux-build. Table 1-3 below describes the required packages, and how to install them on different Linux workstation environments.

Table 1-3: **Packages and Linux Workstation Environments**

Tool / Library	CentOS 7.2/7.3	RHEL6.7/6.8[(1)]	RHEL7.2/7.3	Ubuntu 16.04.1
python	python: 3.4.0	python: 3.4.0	python: 3.4.0	python: 3.4.0
dos2unix	dos2unix-6.0.3-4.el7.x86_64.rpm	dos2unix-3.1-37.el6.x86_64.rpm	dos2unix-6.0.3-4.el7.x86_64.rpm	tofrodos_1.7.13+ds-2.debian.tar.xz
ip	iproute-3.10.0-74.el7.x86_64.rpm	iproute-2.6.32-54.el6.x86_64.rpm	iproute-3.10.0-74.el7.x86_64.rpm	iproute2 4.3.0-1ubuntu3
gawk	gawk-4.0.2-4.el7.x86_64.rpm	gawk-3.1.7-6.el6.x86_64.rpm	gawk-4.0.2-4.el7.x86_64.rpm	gawk (1:4.1.3+dfsg-0.1)
xvfb	xorg-x11-server-Xvfb-1.15.0-7.el7.x86_64.rpm	xorg-x11-server-Xvfb-1.17.4-9.5.el6_8.x86_64.rpm	xorg-x11-server-Xvfb-1.15.0-7.el7.x86_64.rpm	xvfb (2:1.18.3-1ubuntu2.3)
gcc (for RHEL, use devtoolset-2)	gcc 4.8.3	gcc 4.8.3	gcc 4.8.5	gcc 4.8
git	git 1.8.3	git 1.7.1	git 1.8.3	git 1.7.1 or above
make	make 3.81	make 3.81	make 3.82	make 3.81
netstat	net-tools 2.0	net-tools 1.60	net-tools 2.0	net-tools
ncurses devel	ncurses -devel 5.9-13	ncurses -devel 5.7-4	ncurses -devel 5.9-13	libncurses5 -dev
tftp server	tftp-server	tftp-server	tftp-server	tftpd
zlib devel (also, install 32-bit of this version)	zlib-devel-1.2.7-17.el7.x86_64.rpm	zlib-devel-1.2.3-29.el6.x86_64.rpm	zlib-devel-1.2.7-17.el7.x86_64.rpm	i386/zlib1g-dev/1:1.2.8.dfsg-2ubuntu4-dev
openssl devel	openssl -devel 1.0	openssl -devel 1.0	openssl -devel 1.0	libssl -dev
flex	flex 2.5.37	flex 2.5.35	flex 2.5.37	flex
bison	bison-2.7	bison-2.4.1	bison-2.7.4	bison
libselinux	libselinux 2.2.2	libselinux 2.0.94	libselinux 2.2.2	libselinux1
gnupg	gnupg	gnupg	gnupg	gnupg
wget	wget	wget	wget	wget
diffstat	diffstat	diffstat	diffstat	diffstat
chrpath	chrpath	chrpath	chrpath	chrpath
socat	socat	socat	socat	socat
xterm	xterm	xterm	xterm	xterm

Table 1-3: **Packages and Linux Workstation Environments**

Tool / Library	CentOS 7.2/7.3	RHEL6.7/6.8[(1)]	RHEL7.2/7.3	Ubuntu 16.04.1
autoconf	autoconf	autoconf	autoconf	autoconf
libtool	libtool	libtool	libtool	libtool
tar	tar:1.24	tar:1.24	tar:1.24	tar:1.24
unzip	unzip	unzip	unzip	unzip
texinfo	texinfo	texinfo	texinfo	texinfo
zlib1g-dev	-	-	-	zlib1g-dev
gcc-multilib	-	-	-	gcc-multilib
build-essential	-	-	-	build-essential
libsdl1.2-dev	-	-	-	libsdl1.2-dev
libglib2.0-dev	-	-	-	libglib2.0-dev
SDL-devel	SDL-devel	SDL-devel	SDL-devel	-
glibc-devel	glibc-devel	glibc-devel	glibc-devel	-
32-bit glibc	glibc.i686	-	glibc.i686	-
glib2-devel	glib2-devel	glib2-devel	glib2-devel	-
automake	automake	automake	automake	-
screen	screen	screen	screen	screen
pax	pax	pax	pax	pax
gzip	gzip	gzip	gzip	gzip
devtoolset-2	-	devtoolset-2	-	-

Notes:

1. Limited/conditional support with workaround: Install devtools-2 package on RHEL, Cent OS 6.X. Support is limited to Xilinx PetaLinux Images. Any breakage here (RHEL 6.X + devtools-2,) outside the PetaLinux images is not supported by Xilinx. This will not have any support in 2017.3.

CAUTION! *Consult your system administrator if you are not sure about the 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 v2017.1 works only with Vivado 2017.1.*

Installation Steps

Prerequisites

The prerequisites to install the PetaLinux tools are:

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

Run PetaLinux Tools Installer

PetaLinux Tools installation is straight-forward. Without any options, the PetaLinux Tools are installed into the current working directory. Alternatively, an installation path may be specified.

For example: To install PetaLinux Tools under "`/opt/pkg/petalinux`":

```
$ mkdir -p /opt/pkg/petalinux
$ ./petalinux-v2017.1-final-installer.run /opt/pkg/petalinux
```

This installs the PetaLinux Tools into "`/opt/pkg/petalinux`" directory.

IMPORTANT: *Once installed you cannot move or copy the installed directory. In the above example you cannot move or copy /opt/pkg/petalinux.*

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

- `$PETALINUX/etc/license/petalinux_EULA.txt`. EULA specifies in detail the rights and restrictions that apply to the PetaLinux.
- `$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 non-distributable 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 need to source PetaLinux settings first. Refer to section PetaLinux Working Environment Setup.*

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

Troubleshooting

This section describes some common issues you may experience while installing the PetaLinux Tools. If the PetaLinux Tools installation fails, the file "`$PETALINUX/post-install.log`" will be generated in your PetaLinux installation directory.

Table 1-4: **PetaLinux Installation Troubleshooting**

Problem / Error Message	Description and Solution
WARNING: You have less than 1 GB 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 the 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 do not have a TFTP service running on the workstation. Without a TFTP service, you cannot download Linux system images to the target system using u-boot's network/TFTP capabilities. This warning can be ignored for other boot modes. **Solution:** Enable the TFTP service on your workstation. If you are unsure how to enable this service, contact your system administrator.
ERROR: GCC is not installed - unable to continue. Please install and retry	**Problem Description:** This error message indicates that you do not have gcc installed on the workstation. **Solution:** Install gcc using your Linux work-stations package management system. If you are unsure how to do this, contact your system administrator.

Table 1-4: **PetaLinux Installation Troubleshooting *(Cont'd)***

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 do not have the required tools or libraries listed in the "missing-tools-list" or "missing-library-list". **Solution:** Install the packages of the missing tools. Refer to section 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 that is 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. 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', eithe 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 remaining 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. Refer to section 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:

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

2. Verify that the working environment has been set:

```
$ echo $PETALINUX
/opt/pkg/petalinux
```

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

Troubleshooting

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

Table 1-5: **PetaLinux Working Environment Troubleshooting**

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:** See Ubuntu Forum and follow the steps.

PetaLinux BSP Installation

PetaLinux reference board support packages (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 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 is downloaded. You can download PetaLinux BSP from PetaLinux Downloads.
- PetaLinux Working Environment Setup is completed. Refer to section PetaLinux Working Environment Setup.

PetaLinux BSP Installation Steps

Follow the below steps to install a BSP:

1. Change to the directory under which you want PetaLinux projects to be created. For example, if you want to create projects under /home/user:

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

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

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

   The board being referenced is based on the BSP installed. You will see the output, similar to the below output:

   ```
   INFO: Create project:
   INFO: Projects:
   INFO:   * xilinx-zcu102-v2017.1-final.bsp
   INFO: has been successfully installed to /home/user/
   INFO: New project successfully created in /home/user/
   ```

If the specified location is on NFS, it changes the TMPDIR to `/tmp/<projname_timestamp>`. If `/tmp/<projname_timestamp>` is also on NFS, it will throw an error.

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.

Table 1-6: **PetaLinux BSP Installation Troubleshooting**

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 cannot proceed with BSP installation. **Solution:** You have to setup your environment for PetaLinux Tools. Refer to section PetaLinux Working Environment Setup to set it up.

BSP Naming

There are multiple revisions of silicon and board that are being shipped. Table 1-7 lists the supported BSPs that can be downloaded, for Zynq UltraScale family.

Table 1-7: **BSP Naming**

Platform	Silicon Version	Board Version	PetaLinux BSP Name
ZCU102	1.0 Silicon (zu9-es1)	Rev-B	xilinx-zcu102-zu9-es1-v2017.1-final.bsp
		Rev-C	
		Rev-D	
	3.0 Silicon (zu9-es2)	Rev-1.0	xilinx-zcu102-zu9-es2-rev1.0-v2017.1-final.bsp
	Production Silicon (zu9)	Rev-1.0	xilinx-zcu102-v2017.1-final.bsp

Click here to download the BSPs.

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 your own 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 hardware requirements for a Zynq® UltraScale+™ MPSoC hardware project to boot Linux:

1. External memory controller with at least 64 MB of memory (Required)
2. UART (Required)

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 hardware 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 32 MB of memory (Required)
3. UART (Required)

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 32 MB of memory (Required)
    - Dual channel timer with interrupt connected (Required)
    - UART with interrupt connected (Required)
    - 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. 4K Byte is required for Parallel flash and 8K Byte 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 a hardware platform is created with the Vivado design suite. 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, you need to boot with the Power Management Unit (PMU) firmware and ATF. Refer to Appendix B, Generating First Stage Bootloader, PMU Firmware and Arm Trusted Firmware Within Project for building PMUFW and ATF.

Create a New PetaLinux Project

This section describes how to create a new PetaLinux project.

Prerequisites

This section assumes that the PetaLinux Working Environment Setup is complete. Refer to section PetaLinux Working Environment Setup for more details.

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 Zynq UltraScale+ MPSoC, Zynq and MicroBlaze.
- `--name <PROJECT_NAME>` - The name of the project you are building.

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

TIP: *If --template option is used instead of a bsp. you can use the petalinux-config command to choose default board configs, that are close to your board design, as shown below:*
1. petalinux-config--get-hw-description=<PATH-TO-HDF-DIRECTORY>
2. Set CONFIG_SUBSYSTEM_MACHINE_NAME as required. The possible values are: kc705-full, kc705-lite, ac701-full, ac701-lite, kcu105, zc702, zc706, zedboard, zcu102, zcu102-revb, zcu106, zc1751-dc1 and zc1751-dc2.

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

CAUTION! *When a PetaLinux project is created on NFS, it changes the TMPDIR to /tmp/<projname_timestamp>. If /tmp/<projname_timestamp> is also on NFS, it will throw an error. If you want to change the TMPDIR to a local storage use petalinux-config --> Yocto-settings --> TMPDIR. Do not configure the same location as TMPDIR for two different PetaLinux projects. This may cause build errors.*

Version Control

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

Prerequisites

This section assumes that the you have created a new PetaLinux project or have an existing PetaLinux project. Refer to section Create a 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/`"
- "`<plnx-proj-root>/pre-built/`"
- "`<plnx-proj-root>/project-spec/meta-plnx-generated/`"

Replace `.gitignore` in project with:

```
*/config.old
build/
images/linux/
.petalinux/*
!.petalinux/metadata
*.o
*.jou
*.log
.metadata/
meta-plnx-generated/
.Xil/
```

Note: A PetaLinux project should be cleaned before submitting to the source control.

IMPORTANT: *The version control is not fully covered for now in a petalinux project. There are few files which get updated with time stamps or paths. You will have to add them to .gitignore explicitly.*

Import Hardware Configuration

This Section explains how to update an existing/newly created PetaLinux project with a new hardware configuration. This enables you 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. Refer to section Exporting Hardware Platform.
- You have created a new PetaLinux project or have an existing PetaLinux project. Refer to section Create a 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 of the directory containing the `.hdf` file as follows:

   ```
   $ petalinux-config
   --get-hw-description=<path-to-directory-which-contains-hardwaredescription-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  --->
ARM Trusted Firmware Compilation Configuration  --->
PMU FIRMWARE Configuration  --->
u-boot Configuration  --->
Image Packaging Configuration  --->
Firmware Version Configuration  --->
(zcu102-revb) MACHINE_NAME
```

```
Yocto Settings  --->
```

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

```
Subsystem AUTO Hardware Settings
System Processor (psu_cortexa53_0)  --->
Memory Settings  --->
Serial Settings  --->
Ethernet Settings  --->
Flash Settings  --->
SD/SDIO Settings  --->
RTC 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.

For example, if `ps7_ethernet_0` as the `Primary Ethernet` is selected and you enable the 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, refer to Appendix C, Auto Config Settings.

Build System Image

Prerequisites

This section assumes that you have PetaLinux Tools software platform ready for building a Linux system, customized to your hardware platform. Refer to section 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. For example:

```
[INFO] building project
[INFO] generating Kconfig for project
[INFO] oldconfig project
[INFO] sourcing bitbake
[INFO] generating plnxtool conf
[INFO] generating meta-plnx-generated layer
/home/user/xilinx-zcu102-zu9-es2-rev1.0-2017.1/build/misc/plnx-generated
[INFO] generating machine configuration
[INFO] generating bbappends for project . This may take time !
/home/user/xilinx-zcu102-zu9-es2-rev1.0-2017.1/build/misc/plnx-generated
[INFO] generating u-boot configuration files
[INFO] generating kernel configuration files
[INFO] generating kconfig for Rootfs
Loading cache: 100%
|#################################################################################
#################################################################| Time:
0:00:00
Loaded 3235 entries from dependency cache.
Parsing recipes: 100%
|#################################################################################
#################################################################| Time: 0:00:02
Parsing of 2446 .bb files complete (2407 cached, 39 parsed). 3236 targets, 224
skipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
NOTE: PN build list saved to 'pn-buildlist'
NOTE: PN dependencies saved to 'pn-depends.dot'
NOTE: Package dependencies saved to 'package-depends.dot'
NOTE: Task dependencies saved to 'task-depends.dot'
Generate rootfs kconfig
[INFO] oldconfig rootfs
[INFO] generating petalinux-user-image.bb
INFO: bitbake petalinux-user-image
Loading cache: 100%
|#################################################################################
#################################################################| Time:
0:00:00
Loaded 3235 entries from dependency cache.
Parsing recipes: 100%
|#################################################################################
#################################################################| Time: 0:00:02
Parsing of 2446 .bb files complete (2414 cached, 32 parsed). 3236 targets, 224
skipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
Initialising tasks: 100%
|#################################################################################
#################################################################| Time: 0:00:09
Checking sstate mirror object availability: 100%
|#################################################################################
########################################| Time: 0:00:35
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
NOTE: Tasks Summary: Attempted 2645 tasks of which 1892 didn't need to be rerun and
all succeeded.
INFO: generating FIT Image
INFO: bitbake petalinux-user-image -R
/home/user/xilinx-zcu102-zu9-es2-rev1.0-2017.1/build/conf/fit-image.conf
```

```
Parsing recipes: 100%
|##################################################################################
##################################################################| Time: 0:00:29
Parsing of 2446 .bb files complete (0 cached, 2446 parsed). 3236 targets, 224
skipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
Initialising tasks: 100%
|##################################################################################
##################################################################| Time: 0:00:09
Checking sstate mirror object availability: 100%
|##################################################################################
########################################| Time: 0:00:04
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
NOTE: Tasks Summary: Attempted 2646 tasks of which 2576 didn't need to be rerun and
all succeeded.
INFO: Copying Images from deploy to images
INFO: Creating images/linux directory
[INFO] successfully built project
```

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. Refer to Appendix B, Generating First Stage Bootloader, PMU Firmware and Arm Trusted Firmware Within Project for more details on the auto generated first stage bootloader.*

Generate uImage

When you run `petalinux-build`, it generates 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. For example:

```
$ 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 "`<plnx-proj-root>/project-spec/meta-user/recipes-bsp/u-boot/files/platform-top.h`" of your PetaLinux project to overwrite 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.

Table 1-8: **Build System Image Troubleshooting**

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. Refer to section Import Hardware Configuration.

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.

Prerequisites

This section assumes that you have built PetaLinux system image. Refer to section Build System Image for more information.

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, PMU firmware and u-boot.

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

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

For detailed usage, refer to the `--help` option or *PetaLinux Tools Documentation: PetaLinux Command Line Reference* (UG1157) [Ref 4].

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 you have built PetaLinux system image. For more information, refer Build System Image.

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, refer to the --help option or *PetaLinux Tools Documentation: PetaLinux Command Line Reference* (UG1157) [Ref 4].

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 you have built the PetaLinux system image. For more information, refer Build System Image.

Generate Boot Image

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

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

It generates `boot.mcs` in your working directory and it will be copied to the `<plnx-proj-root>/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 the `.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, refer to the `--help` option or document *PetaLinux Tools Documentation: PetaLinux Command Line Reference* (UG1157) [Ref 4].

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. For more information, refer to Generate Boot Image for Zynq UltraScale+ MPSoC.
- 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, refer to the `--help` option or document *PetaLinux Tools Documentation: PetaLinux Command Line Reference* (UG1157) [Ref 4].

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 you have packaged prebuilt images. 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, FSBL and PMUFW for 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, PMU firmware and then ATF 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 PMUFW, 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
```

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 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 PetaLinux software image in a simulated environment without any hardware.

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

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

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

Note: If you do not wish to use prebuilt capability for QEMU boot, refer to Additional Options for Booting on QEMU.

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

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

The example of the kernel boot log messages displayed on the serial console during successful `petalinux-kernel`, is shown below:

```
[   13.399282] Freeing unused kernel memory: 11392K (ffffffc000bd0000 -
ffffffc0016f0000)
[   16.804862] udevd[1668]: starting version 3.2
[   16.819766] random: udevd: uninitialized urandom read (16 bytes read)
[   16.941649] udevd[1669]: starting eudev-3.2
[   22.690630] hrtimer: interrupt took 17368800 ns
[   34.708066] random: dd: uninitialized urandom read (512 bytes read)
[   37.317181] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
[   38.401704] macb ff0e0000.ethernet eth0: link up (100/Full)
[   38.436587] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[   42.300546] random: dropbearkey: uninitialized urandom read (32 bytes read)

PetaLinux 2017.1 plnx_aarch64 /dev/ttyPS0

plnx_aarch64 login: root
Password:
root@plnx_aarch64:~# [  110.059298] random: fast init done
```

```
root@plnx_aarch64:~#
root@plnx_aarch64:~#
```

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 loads 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 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, with PMUFW running in the background.

- 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.

Note: QEMU version has been upgraded to 2.6. The old options are deprecated in new version, they functionally operational. PetaLinux tools still use old options, therefore it gets warning messages, which can be ignored.

Warning message for Zynq UltraScale+ MPSoC:

```
qemu-system-aarch64: -tftp /home/user/xilinx-zcu102-2017.1/images/linux: The -tftp
option is deprecated. Please use '-netdev user,tftp=...' instead.g
```

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, refer to section Installation Steps.
- You have installed PetaLinux BSP on the Linux workstation. If you have not installed, 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 115200 bps.

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 (For example: 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:

```
[    4.233206] Freeing unused kernel memory: 512K (ffffffc000bb0000 - ffffffc000c30000)
INIT: version 2.88 booting
Starting udev
[    4.351288] udevd[1624]: starting version 3.2
[    4.361779] udevd[1625]: starting eudev-3.2
[    4.364957] random: crng init done
Populating dev cache
Tue Mar 14 11:19:35 UTC 2017
Starting internet superserver: inetd.
INIT: Entering runlevel: 5
Configuring network interfaces... ifconfig: SIOCGIFFLAGS: No such device
Starting Dropbear SSH server: Generating key, this may take a while...
Public key portion is:
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDivIP/aVx0x2TkA2hY0+mCmb0M0ySyU3XKKz7/fGqG5egd3JaA8kCwKf
9l74eEfMh/YUnPc1s+ujBq2UWRnBMJbO7jk6KBv0GBVvJtrMxl5cgM7DlWd/jnQESOLjGuXM2OmzYbP4NzVQiX
cjk0q5+cvayzN39c6B3V+a749wxwKzb8UE1DWc4kEswYsOSO8EL8T9y7tJbrD9mOPkoyQRI4lPHveapI3uB+NC
mKabHMqCFsrsL+YYIYlWo7ZXxZdEZ11bncdd9LtzFXYSuTC0IhWnF9DgBfPWxmtu/fyRzmKxMeUY+1NrbBfUwn
4Rf+utxrVWtCPCG3A6+krDYbccVT root@plnx_aarch64
Fingerprint: md5 9e:8e:e0:01:73:1d:48:d6:73:b0:c8:19:7c:15:d1:c6
dropbear.
Starting syslogd/klogd: done
Starting tcf-agent: OK
INIT: Id "hvc0" respawning too fast: disabled for 5 minutes
PetaLinux 2017.1 plnx_aarch64 /dev/ttyPS0
plnx_aarch64 login: root
Password:
root@plnx_aarch64:~#
```

TIP: *If you wish to stop auto-boot, 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.

Table 1-9: **PetaLinux Image on Hardware Troubleshooting**

Problem / Error Message	Description and Solution

Wrong Image Format for boot command.
ERROR: Can't get kernel image!
	Problem Description:
	This error message indicates that the `u-boot` 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. 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. Refer to section Generate Boot Image for Zynq UltraScale+ MPSoC on how to generate `BOOT.BIN`.

TIP: *To know more about u-boot options, use the 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 115200 bps.
- 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 (For example, kermit, minicom) with the baud rate set to 115200 on that console.
8. Run the petalinux-boot command as follows on your workstation:

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

Note: If you wish not to use prebuilt capability for JTAG boot, refer to Additional options for booting with JTAG.

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

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

The example of the message on the workstation command console for successful `petalinux-boot` is:

```
$ 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: Downloading ELF file to the target.
```

The example of the message on the serial console for successful `petalinux-boot` is:

```
[    4.862842] Freeing unused kernel memory: 5376K (ffffffc000bb0000 -
ffffffc0010f0000)
[    5.546506] udevd[1762]: starting version 3.2
[    5.557113] udevd[1763]: starting eudev-3.2
[    6.088398] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
[    7.111774] macb ff0e0000.ethernet eth0: link up (1000/Full)
[    7.119195] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready

PetaLinux 2017.1 plnx_aarch64 /dev/ttyPS0

plnx_aarch64 login: root
Password:
root@plnx_aarch64:~#
```

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.

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 --bitstream <BITSTREAM> option:

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

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

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

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

Logging Tcl/XSDB for JTAG Boot

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

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

Troubleshooting

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

Table 1-10: **PetaLinux Image on Hardware with JTAG Troubleshooting**

Problem / Error Message	Description and Solution
ERROR: This tool requires 'xsdb' and it is missing. Please source Xilinx Tools settings first.	**Problem Description:** This error message indicates that PetaLinux tools can not find the xsdb tool that is a part of the Xilinx Vivado or SDK tools. **Solution:** You have to setup Vivado Tools Working Environment. Refer to PetaLinux Working Environment Setup.
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 ensure 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 Trivial File Transfer Protocol (TFTP).

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

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- Host environment with TFTP server is setup and PetaLinux Image is built for TFTP boot. 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 115200 bps.
- 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.
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 (For example, kermit, minicom) with the baud rate set to 115200 on that console.
8. Run the petalinux-boot command as follows on your workstation.

```
$ petalinux-boot --jtag --prebuilt 2
```

The `--jtag` option tells `petalinux-boot` to boot on hardware via JTAG, and the `--prebuilt 2` option downloads 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 example of a Workstation console output for successful u-boot download is:

```
U-Boot 2017.01 (Mar 28 2017 - 14:14:14 -0600) Xilinx ZynqMP ZCU102

I2C:   ready
DRAM:  2 GiB
EL Level:       EL2
Chip ID:        xczu9eg
MMC:   sdhci@ff170000: 0 (SD)
SF: Detected n25q512a with page size 512 Bytes, erase size 128 KiB, total 128 MiB
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Bootmode: JTAG_MODE
Net:   ZYNQ GEM: ff0e0000, phyaddr c, interface rgmii-id
eth0: ethernet@ff0e0000
U-BOOT for xilinx-zcu102-zu9-es2-rev1_0-2017.1

ethernet@ff0e0000 Waiting for PHY auto negotiation to complete....... done
BOOTP broadcast 1
BOOTP broadcast 2
BOOTP broadcast 3
DHCP client bound to address 10.10.70.2 (1257 ms)
Hit any key to stop autoboot:  0
Device: sdhci@ff170000
Manufacturer ID: 3
OEM: 5054
Name: SL16G
Tran Speed: 50000000
Rd Block Len: 512
SD version 3.0
High Capacity: Yes
Capacity: 14.5 GiB
Bus Width: 4-bit
Erase Group Size: 512 Bytes
** Invalid partition 1 **
ZynqMP>
```

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

```
ZynqMP> print serverip
```

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

```
ZynqMP> set serverip <HOST IP ADDRESS>; saveenv
```

12. Boot the kernel using the following command.

```
ZynqMP> run netboot
```

Troubleshooting

Table 1-11: **PetaLinux Image on Hardware with TFTP**

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`: `ZynqMP> set serverip <HOST IP ADDRESS>;saveenv`

BSP Packaging

BSPs are useful for distribution in general and allude to Xilinx Worldwide Technical Support (WTS) 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 you have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. 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 generates MY.BSP including the following elements from the specified project:
   - `<plnx-proj-root>/project-spec/`
   - `<plnx-proj-root>/config.project`
   - `<plnx-proj-root>/.petalinux/`
   - `<plnx-proj-root>/pre-built/`
   - 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 with external sources.

   The support for search path is obsolete. It is your responsibility to copy the external sources under components/ext_sources. Refer to Using External Kernel and U-boot With PetaLinux for more details. The BSP has to be packaged.

Firmware Version Configuration

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

Prerequisites

This section assumes that you have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. For more information, refer to Import Hardware Configuration.

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 you have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. 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 Root file 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, for example, Flash and SD/MMC using `petalinux-config` command.

Prerequisites

This section assumes that you have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. 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 (ë) 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.

Table 1-12: **Boot Images Storage Troubleshooting**

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. Refer to section 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.

Note: The PetaLinux tools uses the boot, bootenv (it is for u-boot environment variables) and kernel partitions to generate the u-boot commands:

The 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 describes about Base Root File System Configuration.

Prerequisites

This section assumes that you have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. Refer to section 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.

Managing Image Size

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

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 you have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. 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 example, `CONFIG_SHMEM`, `CONFIG_AIO`, `CONFIG_SWAP`, `CONFIG_SYSVIPC`. For more details, refer Linux kernel documentation.

CAUTION! *Note that disabling of some `config` items may lead to unsuccessful boot. So it is expected that you have the 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.*

Note: If kernel or rootfs size increases and is greater than 128 MB, you need to do the following:
1. Mention the Bootm length in `platform-top.h`
#define CONFIG_SYS_BOOTM_LEN <value greater then image size>
2. Undef the CONFIG_SYS_BOOTMAPSZ in `platform-top.h`

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 you have created a new PetaLinux project (refer to section Create a 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`. Refer to Root file system Type Configuration.
2. Build the system image. Refer to Build System Image.
3. Use one of the following methods to boot the system image.
    a. Boot a PetaLinux Image on QEMU, refer to section Boot a PetaLinux Image on QEMU.
    b. Boot a PetaLinux Image on Hardware with SD Card, refer to section Boot a PetaLinux Image on Hardware with SD Card.
    c. 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 the TFTP boot.

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

Prerequisites

This section assumes that the following prerequisites have been satisfied:

- You have created a new PetaLinux project (refer to section Create a 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". By default the TFTP directory ID is /tftpboot.
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 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 a 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 "`<plnx-proj-root>/components/plnx_workspace/device-tree-generation/plnx_aarch64-system.dts`".
8. Build the system image. Refer to section Build System Image.
9. You can see the updated boot arguments only after building.

Booting with NFS

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

1. JTAG: In this case, bootloader and kernel will be downloaded on to the target through JTAG. 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. 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 TFTPBOOT. 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. 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 a 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 60 MB in size and formatted as a `FAT32` filesystem. Ensure that there is 4 MB 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 ensure that the SD card partitions are 4 MB 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 generated DTSI will be updated with the PetaLinux loading rootfs from SD card default settings. You can check "`<plnx-proj-root>/components/plnx_workspace/device-tree-generation/system-conf.dtsi`".

Note: These changes will be reflected only after the build.

7. Build PetaLinux images. For more information, refer to Build System Image.
8. Generate boot image. For more information, refer to section Generate Boot Image for Zynq UltraScale+ MPSoC.
9. The generated `rootfs.cpio.gz` file will be present in `images/linux` directory. To extract cpio, use gzip -d rootfs.cpio.gz.

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

Table 1-13: **Configuring SD Card ext Filesystem Boot**

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

Including Prebuilt Applications

If an application is developed outside PetaLinux (for example, 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 you have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. 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 (For example, MicroBlaze, ARM etc.).
2. Create an application with the following command.

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

3. Change to the newly created application directory.

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

4. Remove existing myapp app and copy the prebuilt myapp.

```
$ 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 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 you have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. 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 (For example, MicroBlaze, ARM etc.).
2. To create a module project, use the following command.

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

3. Change to the newly created module directory.

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

4. Place the pre-built library mymodule

```
$ cp <path-to-prebuilt-module>/gpiomod.ko./
```

5. Edit `mymodule.bb` file, the file should look like the following:

```
#
# This file is the mymodule recipe.
#

SUMMARY = "Simple shivamod application"
SECTION = "PETALINUX/apps"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

SRC_URI = "file://gpiomod.ko \
        "

inherit module-base

S = "${WORKDIR}"

do_install() {
    install -d ${D}${base_libdir}/modules/${KERNEL_VERSION}/extra
    install -m 0755 ${S}/gpiomod.ko ${D}/${base_libdir}/modules/${KERNEL_VERSION}/extra/
}

FILES_${PN} = "${base_libdir}/modules/"
```

6. Run.

```
petalinux-build
```

Adding Custom Applications

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

Prerequisites

This section assumes that you have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. Refer to section 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
```

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

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

or:

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

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

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

Note: If the application name has '_', refer Appendix H, App/module name having ' _ '.

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

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

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

2. Change to the newly created application directory.

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

You will see the following PetaLinux template-generated files:

Table 1-14: **Adding Custom Applications Template Generated Files**

Template	Description
`<plnx-proj-root>/project-spec/meta-user/recipes-core/images/petalinux-image.bbappend`	Configuration file template - This file controls the integration of your application into the PetaLinux rootfs menu configuration. It also allows you select or de-select the app and its dev, dbg packages into the target root file system
Makefile	Compilation file template - This is a basic Makefile containing targets to build and install your application into the root 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.

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

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

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

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.

CAUTION! *You can delete the app directory if it is no longer required. Apart from deleting the app directory, you have to remove the line: IMAGE_INSTALL_append= " myapp" from <plnx-proj-root>/project-spec/meta-user/recipes-core/images/petalinux-image.bbappend. Deleting the directory by keeping the mentioned line will throw an error.*

Adding Custom Modules

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

Prerequisites

This section assumes that you have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. Refer to section 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
   ```

   For example, 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>/project-spec/meta-user/recipes-modules/mymodule` directory.

Note: If the application name has '_', refer Appendix H, App/module name having ' _ '.

2. Change to the newly created module directory.

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

You will see the following PetaLinux template-generated files:

Table 1-15: **Adding Custom Modules 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. Click here to customize the make file.
README	A file to introduce how to build the user module.
mymodule.c	Simple kernel module in C.
`<plnx-proj-root>/project-spec/meta-user/recipes-core/images/petalinux-image.bbappened`	Configuration file template - This file controls the integration of your application into the petalinux rootfs menu configuration system. It also allows you to select or de-select the app and its dev, dbg packages into the target root file system.

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

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

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

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

CAUTION! *You can delete the module directory if it is no longer required. Apart from deleting the module directory, you have to remove the line: IMAGE_INSTALL_append= "mymodule" from <plnx-proj-root>/project-spec/meta-user/recipes-core/images/petalinux-image.bbappend. Deleting the directory by keeping the mentioned line in petalinux-image.bbappend will throw an error.*

Building User Applications

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

Prerequisites

This section assumes that you have included 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 "`<TMPDIR>/work/aarch64-xilinx-linux/myapp/1.0-r0/`")

```
$ petalinux-build
```

To build myapp into an existing system image:

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

Note: `do_populate_sysroot` is to generate the sysroot based on the selected prebuilt packages options from the menuconfig. You do not have to always run `do_populate_sysroot` before building the application, but you need to run it at least once before you build the application.

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

- To clean the selected user application:

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

- To rebuild the selected user application:

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

  This will just compile the application, the compiled executable files will be in `${TMPDIR}/work/aarch64-xilinx-linux/myapp/1.0-r0/ directory.`

  Note: If you want to use the build artifacts for debugging with the third party utilities, add the line: `RM_WORK_EXCLUDE += "myapp"` in `<plnx-proj-root>/<plnx-proj-root>/project-spec/meta-user/conf/petalinuxbsp.conf`. Without this line the bitbake will remove all the executables after building.

  To see all list of tasks for myapp: `petalinux-build -c myapp -x listtasks`.

- To install the selected user application:

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

  This will install the application into the target rootfs host copy: `<TMPDIR>/work/plnx_aarch64-xilinx-linux/petalinux-user-image/1.0-r0/rootfs/`.

Note: TMPDIR can be found in petalinux-config->Yocto-settings --> TMPDIR. If the project is on local storage, TMPDIR is `<plnx-proj-root>/build/tmp/`. Do not configure the same location as TMPDIR for two different PetaLinux projects. This can cause build errors.

For Zynq UltraScale+ MPSoC:
ARCH: aarch64
MACHINE: plnx_aarch64

Testing User Application

This section describes how to test a user application.

Prerequisites

This section assumes that you have built and installed pre-compiled/custom user applications. 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 /usr/bin
   ```

Unless you have changed the location of user application through its Makefile, the user application will be put in to "`/usr/bin`" directory.

3. Run your user application on the target system console. For example, 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 `petalinuxbuild -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.

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 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 `<TMPDIR>/work/{ARCH}-xilinx-linux-gnueabi/mymodule/1.0-r0/`)

```
$ petalinux-build
```

To build mymodule into an existing system image:

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

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

- To clean the selected user module:

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

- To rebuild the selected user module:

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

  This will just compile the module, the compiled executable files will be in `<TMPDIR>/work/aarch64-xilinx-linux/mymodule/1.0-r0/` directory.

  To see all list of tasks for this module: `petalinux-build -c mymodule -x listtasks`.

- To install the selected user module:

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

  This will install the module into the target rootfs host copy: `<TMPDIR>/work/plnx_aarch64-xilinx-linux/petalinux-user-image/1.0-r0/rootfs/`.

Note: TMPDIR can be found in petalinux-config->Yocto-settings --> TMPDIR. Do not configure the same TMPDIR for two different PetaLinux projects. This can cause build errors. If the project is on local storage, TMPDIR is `<plnx-proj-root>/build/tmp/`.

For Zynq UltraScale+ MPSoC:
ARCH: aarch64
MACHINE: plnx_aarch64

PetaLinux Auto Login

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

Prerequisites

This section assumes that you have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. Refer to section Import Hardware Configuration for more information.

Steps for PetaLinux Auto Login

Follow the below steps for PetaLinux Auto Login:

1. cd <plnx-proj-root>
2. petalinux-config
3. Select Yocto-settings --> Enable debug-tweaks
4. Save the configuration and exit
5. petalinux-build

Application Auto Run at Startup

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

Prerequisites

This section assumes that you have already added and built the PetaLinux application. Refer to sections Adding Custom Applications and Building User Applications for more information.

Steps for Application Auto Run at Startup

If you have a prebuilt or newly created custom user application myapp located in your PetaLinux project at `<plnx-proj-root>/project-spec/meta-user/recipes-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, refer to Including Prebuilt Applications.*
If you want to create custom application and install it in PetaLinux Root file system, refer to Adding Custom Applications.
If your auto run application is a blocking application which will never exit, launch this application as a daemon.

1. Create new install app myapp-init

```
cd <plnx-proj-proot>/
petalinux-create -t apps --template install -n myapp-init  --enable
```

2. Edit the file `project-spec/meta-user/recipes-apps/myapp-init/myapp-init.bb`. The file should look like the following:

```
#
# This file is the myapp-init recipe.
#

SUMMARY = "Simple myapp-init application"
SECTION = "PETALINUX/apps"
LICENSE = "MIT"
LIC_FILES_CHKSUM =
"file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

SRC_URI = "file://myapp-init \
        "

S = "${WORKDIR}"

FILESEXTRAPATHS_prepend := "${THISDIR}/files:"

inherit update-rc.d

INITSCRIPT_NAME = "myapp-init"
INITSCRIPT_PARAMS = "start 99 S ."

do_install() {
        install -d ${D}${sysconfdir}/init.d
        install -m 0755 ${S}/myapp-init ${D}${sysconfdir}/init.d/myapp-init
}

FILES_${PN} += "${sysconfdir}/*"
```

3. To run myapp as daeomon

```
edit the file project-spec/meta-user/recipes-apps/myapp-init/files/myapp-init
The file should look like
#!/bin/sh

DAEMON=/usr/bin/myapp

start ()
{
        echo " Starting myapp"
        start-stop-daemon -S -o --background -x $DAEMON
}

stop ()
{
        echo " Stoping myapp"
        start-stop-daemon -K -x $DAEMON
}

restart()
{
        stop
        start
}

[ -e $DAEMON ] || exit 1

        case "$1" in
                start)
                        start; ;;
                stop)
                        stop; ;;
                restart)
                        restart; ;;
                *)
                        echo "Usage: $0 {start|stop|restart}"
                        exit 1
        esac
exit $?
```

4. Run petalinux-build

Debugging the Linux Kernel in QEMU

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

Prerequisites

This section assumes that you have built PetaLinux system image. 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, get the GDB TCP port from `-gdb tcp:<TCP_PORT>`.

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

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

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

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

You should see the gdb prompt. For example:

```
GNU gdb (Linaro GDB 2017.01) 7.12.1.20170130-git
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=x86_64-unknown-linux-gnu
--target=aarch64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from vmlinux...done.
```

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

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

6. To let QEMU continue execution:

   ```
   (gdb) continue
   ```

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

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

CAUTION! *If another process is using port 9000, petalinux-boot 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.

Table 1-16: **Debugging the Linux Kernel in QEMU Troubleshooting**

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:** • Check your QEMU console to ensure QEMU is running. • 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 describes debugging user applications with the Eclipse Target Communication Framework (TCF) 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 the XSDK tool.
- The Vivado Tools Working Environment is properly set. Refer to section PetaLinux Working Environment Setup.
- You have created a user application and built the system image including the selected user application. 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:

   ```
   admin   --->
   ```

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

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

```
base --->
baseutils --->
benchmark --->
console --->
devel --->
fonts --->
kernel -->
libs --->
misc --->
multimedia -->
net -->
network -->
optional -->
utils -->
xll -->
```

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

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

6. Ensure that `tcf-agent` is enabled.

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

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

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

8. Exit the menu.
9. Rebuild the target system image including myapp. 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. For example, ZC702
7. Locate the system.hdf for your target hardware. This can be found in `<plnx-proj-root>/project-spec/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)` and launch configuration:

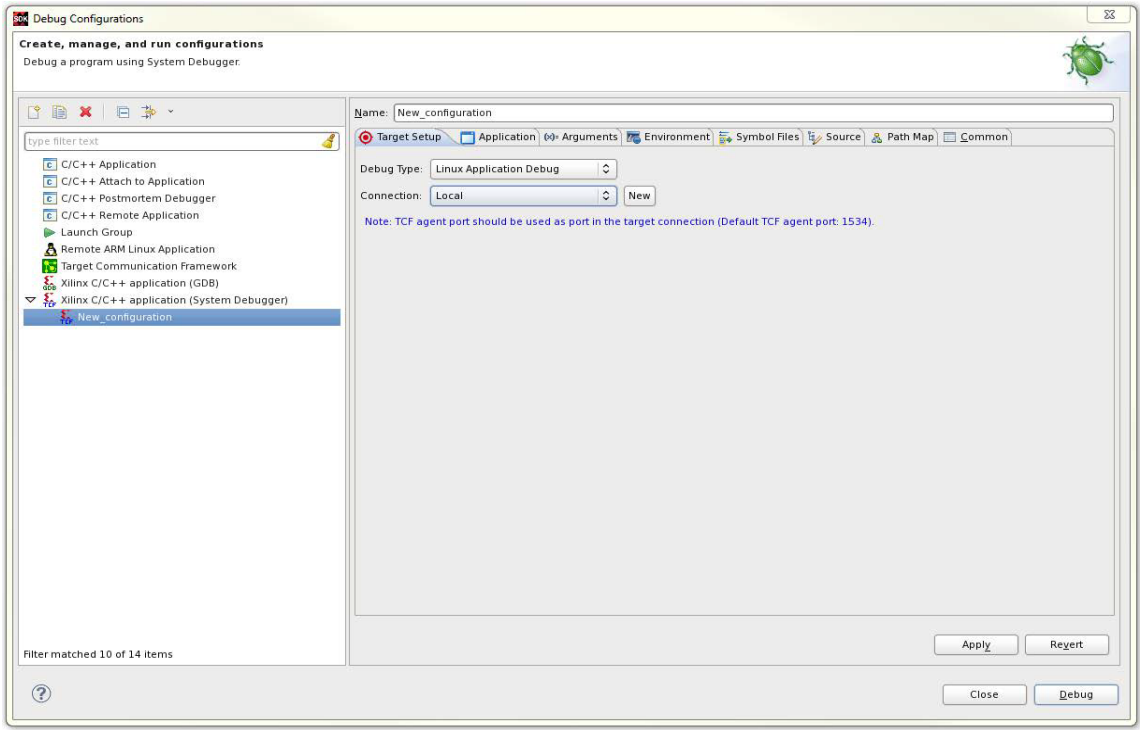

Figure 1-1: **XSDK Debug Configurations**

10. The `Debug Type` should be set to `Linux Application Debug`.
11. Select the New option to enter the Connection details.

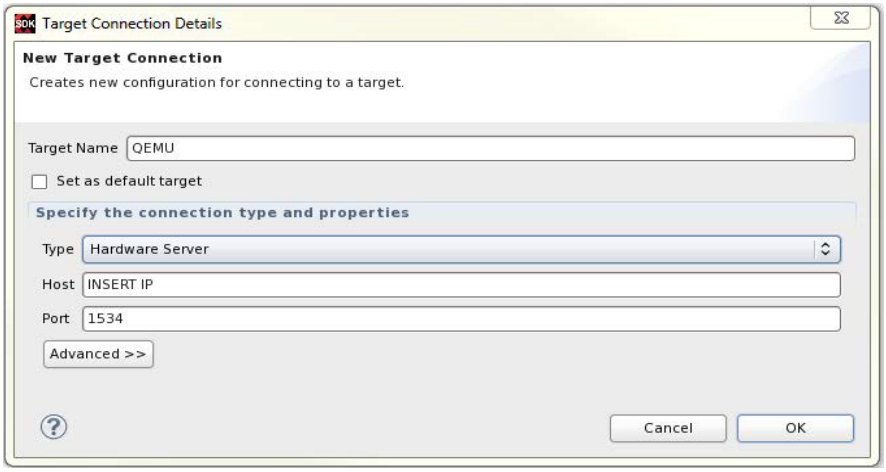

Figure 1-2: **XSDK Debug New Target Configuration**

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. For example, 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.

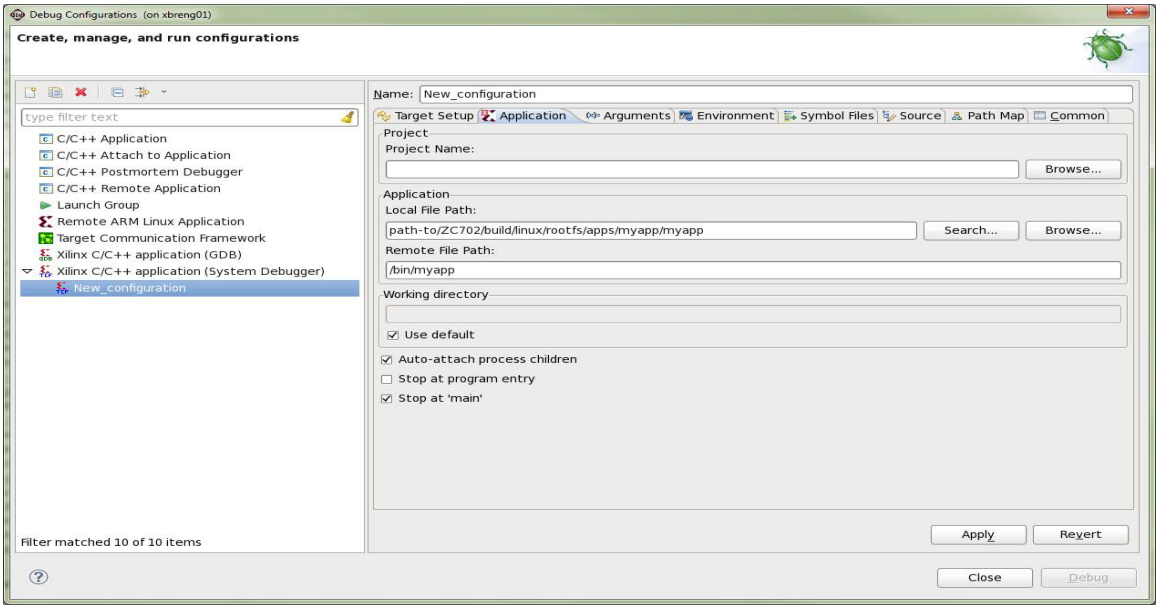

Figure 1-3: **XSDK Debug Configurations**

15. Enter the Local File Path to your compiled application in the project directory. For example, `<TMPDIR>/work/aarch64-xilinx-linux/myapp1/1.0-r0/image/usr/bin/`.

    Note: While creating the app, you need to add RM_WORK_EXCLUDE += "myapp" in `project-spec/meta-user/conf/petalinuxbsp.conf`, otherwise the images will not be available for debugging.

16. The Remote File Path on the target file system should be the location where the application can be found. For example, `/usr/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:

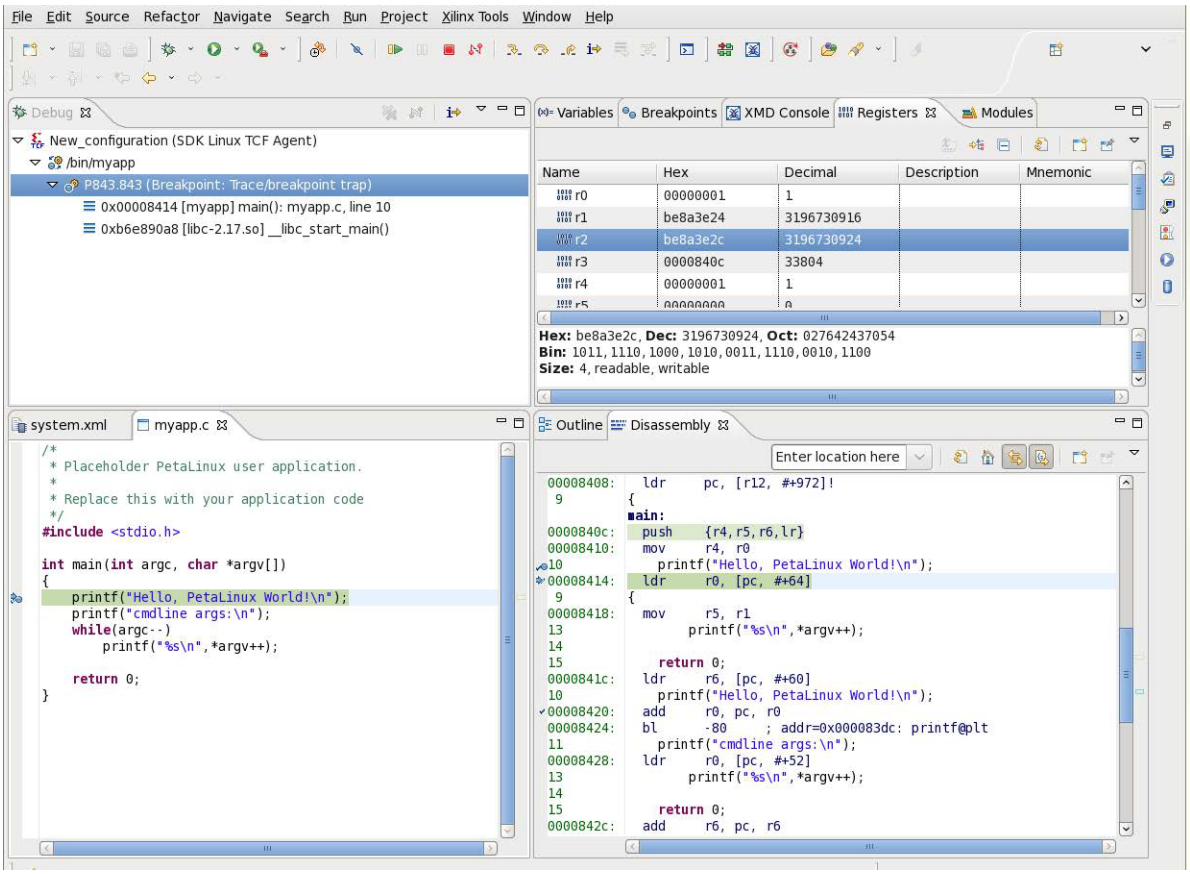

Figure 1-4: **XSDK Debug**

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 GDBs

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. Refer to section PetaLinux Working Environment Setup.
- You have created a user application and built the system image including the selected user application. 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 Debugging:

   ```
   Filesystem Packages --->
   Libs --->
   Apps --->
   Modules --->
   User packages --->
   PetaLinux RootFS Settings --->
   ```

4. Select Apps -->myapp -->

   ```
   [ ] myapp
   [X] myapp-dbg
   [ ] myapp-dev
   ```

   Select `myapp-dbg`. Exit the myapp sub-menu.

5. Exit the App sub-menu, and select the Filesystem Packages, followed by the base sub-menu:

   ```
   misc
   ```

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

   ```
   selecting_gdb server
   ```

```
[ ] gdb
[ ] gdb-dev
[X] gdbserver
[ ] gdb-dbg
```

7. Exit the menu and select `<Yes>` to save the configuration.
8. Rebuild the target system image. 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@plnx_aarch64:~# gdbserver host:1534 /usr/bin/myapp
   Process /bin/myapp created; pid = 73
   Listening on port 1534
   ```

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

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:
   a. Use the IP address of the target system, for example: 192.168.0.10. If you are not sure about the IP address, run `ifconfig` on the target console to check.
   b. Use the port 1534. If you select 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. For example, 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:
/usr/bin/myapp
Child exited with status 0
GDBserver exiting
root@plnx_aarch64:~#
```

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

Visit www.gnu.org, for more information on general usage of GDB, refer to the GDB project documentation:

Troubleshooting

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

Table 1-17: **Debugging Zynq UltraScale+ MPSoC Applications with GDB Troubleshooting**

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

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. It describes two ways of doing the out-of-tree builds.

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. 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 --->" sub-menu.
   - For kernel, select "linux-kernel () --->" and then select remote

     ( ) linux-xlnx
     ( ) ext-local-src
     (X) remote

   - For U-boot, select " u-boot () --->" and then select remote

     ( ) u-boot-plnx
     (X) remote
     ( )ext-local-src

4. For kernel, select "Remote linux-kernel settings --->", select Remote linux-kernel git URL and enter git URL for linux-kernel.

   For example: `git://github.com/Xilinx/linux-xlnx.git;protocol=https`

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

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

   Note: Only git:// has to be entered.

   Set a git tag as "Remote git TAG/commit ID".

   You have to set any of the following values to this setting, otherwise an error message appears.

   1 - To point to HEAD of repo
   `${AUTOREV}`

   2 - To point to any tag
   `petalinux-v2017.1-final`

   3 - To point to any commit id:
   `commit id sha key`

   Note: Tag or commit id cannot be on a git branch. They should be in master. If you need to use a branch/tag/commit from a branch, git checkout locally and choose ext-local-src in step 3.

5. Exit the menu, and save your settings.

Using External Kernel and U-boot With PetaLinux

PetaLinux includes kernel source and u-boot source. However, you can build your own kernel and u-boot with PetaLinux.

PetaLinux supports local sources for kernel, Uboot and ATF

For external sources create a directory <plnx-proj-root>/components/ext_sources/

1. Copy the kernel source directory to

   ```
   <plnx-proj-root>/components/ext_sources/<MY-KERNEL>
   ```

2. Copy the u-boot source directory to

   ```
   <plnx-proj-root>/components/ext_sources/<MY-U-BOOT>
   ```

3. Run petalinux-config, and go into "linux Components Selection --->" sub-menu,
   - For kernel, select "linux-kernel () --->" and then select ext-local-src

     ( ) linux-xlnx
     (X) ext-local-src
     ( ) remote

   - For U-boot, select " u-boot () --->" and then select ext-local-src

     ( ) u-boot-plnx
     ( ) remote
     (X) ext-local-src

4. For kernel, select "External linux-kernel local source settings ---> "
5. Enter the path:

   ```
   ${TOPDIR}/../components/ext_sources/<MY-KERNEL>
   ```

6. For uboot, select "External u-boot local source settings --->". Enter the path:

   ```
   ${TOPDIR}/../components/ext_sources/<MY-U-BOOT>
   ${TOPDIR}     -->   is the yocto variable pointing to <plnx-proj-root>/build directory
   ```

You can give an absolute path for the source, but you must ensure you change the path while using the BSP with the source. It is your responsibility to change the source path. The sources can be placed outside the project as well.

Note: When creating a BSP with external sources in project, it is your responsibility to copy the sources into the project and do the packing. Please refer to BSP Packaging section for more details.

Troubleshooting

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

Table 1-18: **Configuring Out-of-Tree Build Troubleshooting**

Problem / Error Message	
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. 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 you have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. 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-projroot>/project-spec/meta-user/recipes-bsp/device-tree/`:

- `multi-arch/`
- `system-user.dtsi`
- `xen-overlay.dtsi`
- `zynqmp-qemu-arm.dts`
- `openamp-overlay.dtsi`

- `xen-qemu-overlay.dtsi`

The generated files will be in the `<plnx-projroot>/components/plnx_workspace/device-tree-generation/` directory.

CAUTION! *All the above mentioned dtsi files are generated by the tool. Editing any of these files is not recommended.*

The `<plnx-projroot>/project-spec/meta-user/recipes-bsp/device-tree` holds the device-tree bbappend and files directory. The files directory holds the `system-user.dtsi` which can be modified.

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

CAUTION! *DTSI files listed above `*.dtsi` are automatically generated; you are not supposed to edit these files.*

If you wish to add information, like the Ethernet PHY information, this should be included in the `system-user.dtsi` file. In this case, device tree should include the information relevant for your specific platform as information (here, Ethernet PHY information) is board level and board specific.

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

An example of a well-formed Device-tree node for the `system-user.dtsi` is shown 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.*

The following example demonstrates adding the `sample-user-1.dtsi` file:

1. Edit the file `project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi` and add this line: `/include/ "system-user-1.dtsi`. It should look like the following:

```
/include/ "system-conf.dtsi"
/include/ "system-user-1.dtsi"
/ {
};
```

2. Edit the file `project-spec/meta-user/recipes-bsp/device-tree/device-tree-generation_%.bbappend` and add this line to it: `file://system-user-1.dtsi`. The file should look like this:

```
SRC_URI_append ="\
    file://system-user.dtsi \
    file://system-user-1.dtsi \
"
FILESEXTRAPATHS_prepend := "${THISDIR}/files:"
```

It is not recommended to change anything in `<plnx-proj-root>/components/plnx_workspace/device-tree-generation/`

All the changes have to be made in meta-user only. For example, if you want to change anything, you can do in meta-user by rewriting the entire node with your changes in meta-user layer. You can delete a particular node from any of the generated dtsi, using delete-node in meta-user.

U-Boot Configuration

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

Prerequisites

This section assumes that you have PetaLinux Tools software platform ready for building a Linux system customized to your hardware platform. Refer to section Import Hardware Configuration for more information.

Configuring U-Boot

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

- Configuration `_OPTIONS_`:

You will be able to select the configuration options. They have names beginning with "CONFIG_".

- Configuration _SETTINGS_:

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

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

PetaLinux u-boot configuration is associated with config.mk and platform-auto.h configuration files which are located at <plnxproj_root>/project-spec/meta-plnx-generated/recipes-bsp/u-boot/configs and platform-top.h located at <plnxproj_root>/project-spec/meta-user/recipes-bsp/u-boot/files/.

For setting u-boot environment variables, edit CONFIG_EXTRA_ENV_SETTINGS variable in platform-auto.h. Note that platform-auto.h is regenerated each time "petalinux-config" is run.

CAUTION! *config.mk and platform-auto.h files are automatically generated; do not edit these files.*

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

Steps to add CONFIG_ option (For example, CONFIG_CMD_MEMTEST) to platform-top.h:

- Change into the root directory of your PetaLinux project.

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

- Open the file platform-top.h

  ```
  $ vi project-spec/meta-user/recipes-bsp/u-boot/files/platform-top.h
  ```

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

  ```
  #define CONFIG_CMD_MEMTEST
  ```

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

- Build the U-Boot image.

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

- Generate BOOT.BIN using the following command.

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

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

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

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

```
Testing 00000000 ... 00001000:
Pattern 00000000 Writing... Reading...Iteration: 20369
```

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

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

Yocto Features

This chapter gives all the information regarding the various features provided by Yocto.

Accessing BitBake in a Project

BitBake is available only in the bash shell.

Steps to get the BitBake utility for Zynq UltraScale+ MPSoC:

1. You should run `petalinux-config` or `petalinux-config --oldconfig` at least once after creating the project, so that the required environment is setup.
2. Source the PetaLinux tools script:

   ```
   source /opt/pkg/petalinux/settings.sh
   ```

3. Source the Yocto e-SDK:

   ```
   source
   /opt/pkg/petalinux/components/yocto/source/aarch64/environment-setup-aarch64-xilinx
   -linux
   ```

4. Source the environment setup script:

   ```
   source
   /opt/pkg/petalinux/components/yocto/source/aarch64/layers/core/oe-init-build-env
   ```

5. After the above step, you will be redirected to the build directory. Stay in the build directory to run bitbake.
6. Export XSCT:

   ```
   export PATH=/opt/pkg/petalinux/tools/hsm/bin:$PATH
   ```

7. Parse PETALINUX variable to recipes:

   ```
   export BB_ENV_EXTRAWHITE="$BB_ENV_EXTRAWHITE PETALINUX"
   ```

8. To test if the bitbake is available, run:

   ```
   bitbake fsbl -c cleansstate
   bitbake fsbl
   ```

The generated images will be placed in the deploy directory. You have to copy the generated images into `<plnx-proj-root>/images/linux` directory to work with the other commands.

Adding a Recipe from Yocto e-SDK Layers to petalinux-image-full.bb

The rootfs menuconfig is populated based on petalinux-image. The petalinux-image target is supported by Xilinx.

An example to add alsa-utils is shown below:

1. The recipes can be found in yocto e-SDK: `/opt/pkg/petalinux/components/yocto/source/aarch64/layers/core/meta/recipes-multimedia/alsa/`
2. Open `<plnx-proj-root>/project-spec/meta-user/recipes-core/images/petalinux-image.bbappend` and add the following line:

   ```
   IMAGE_INSTALL_append = " alsa-utils"
   ```

3. Run `petalinux-config -c rootfs`

   Select "user packages -->"
   Select "alsa-utils-->"
   Enable it, save and exit

4. Run `petalinux-build`

   Note: It is your responsibility to add the recipes in the layers available in petalinux tools, apart from petalinux-image.

IMPORTANT: *All recipes which are in petalinux-image had sstate locked. To unlock you have to add SIGGEN_UNLOCKED_RECIPES += "my-recipe" in project-spec/meta-user/conf/petalinuxbsp.conf.*

Note: Whenever "_append" is used, there should exist an initial space after = ".

Adding Package Group

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

For example:

```
$PETALINUX/components/yocto/source/aarch64/layers/meta-petalinux/recipes-core/packagegroups/packagegroup-petalinux-x11.bb
```

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

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

```
DESCRIPTION = "PetaLinux ASLA supported Packages"

  inherit packagegroup

ALSA_PACKAGES = " \
        alsa-lib \
        alsa-plugins \
        alsa-tools \
        alsa-utils \
        alsa-utils-scripts \
        pulseaudio \
        "
RDEPENDS_${PN}_append_zynqmp += " \
        ${ALSA_PACKAGES} \
        "
```

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

To add this package group in rootfs menuconfig, add `IMAGE_INSTALL_append = " packagegrouppetalinux-alsa"` in `<plnx-proj-root>/project-spec/meta-plnx-generated/recipes-core/petalinux-image.bbappend` to reflect in menuconfig.

Then Launch petalinux-config -c rootfs, select user packages ---> packagegroup ---> petalinux-alsa and save and exit petalinux-build.

Shared sstate-cache

Yocto e-SDK contains minimal shared sstate-cache. Xilinx hosts the full petalinux-image shared sstate-cache at http://petalinux.xilinx.com/sswreleases/rel-v2017.1/.

During petalinux-build, bitbake will search for sstate cache in the petalinux tool, that is the minimal set. If the sstate cache is not found in this location, bitbake then searches for the same in www.xilinx.com. Yet, if the sstate cache is not found, bitbake will build from scratch. sstate is signature locked. To add any `.bbappend` files for any rootfs components, which already exists, you need to add `SIGGEN_UNLOCKED_RECIPES += "<component>"` in `<plnx-proj-root>/project-spec/meta-user/conf/petalinuxbsp.conf`.

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
- Linux kernel
- rootfs. rootfs is composed of the following components:
  - Prebuilt packages
  - Linux user applications (optional)
  - User modules (optional)

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

```
<plnx-proj-root>
    -components
        - plnx_workspace
             - device-tree-generation
             - fsbl
             - fsbl_bsp
             - fsbl_hwproj
             - .metadata
             - pmu-firmware
             - pmu-firmware_bsp
             - pmu-firmware_hwproj
             - SDK.log
             - .Xil
    -config.project
    -.gitignore
    -hardware
    -.petalinux
     pre-built
        -linux
             - etc
             - images
```

```
            - implementation
        - project-spec
            -attributes
            -configs
                - config
                - rootfs_config
            - hw-description
            - meta-plnx-generated
                - conf
                - COPYING.MIT
                - README
                - recipes-bsp
                - recipes-core
                - recipes-kernel
            - meta-user
                - conf
                - COPYING.MIT
                - README
                - recipes-apps
                - recipes-bsp
                - recipes-core
                - recipes-kernel
                - yocto-layer.log
          - README
```

Table A-1: **PetaLinux Project Description**

File / Directory in a PetaLinux Project	Description
"<plnx-proj-root>/.petalinux/"	Directory to hold tools usage and WebTalk data.
"<plnx-proj-root>/config.project/"	Project configuration file.
"<plnx-proj-root>/project-spec"	Project specification of the project.
"<plnx-proj-root>/project-spec/hw-description"	Hardware description imported from Vivado.
"<plnx-projroot>/project-spec/configs"	Configuration files of top level config and rootfs config
"<plnx-proj-root>/project-spec/configs/config"	Configuration file used to store user setiings
"<plnx-proj-root>/project-spec/configs/rootfs_config"	Configuration file used for root filesystem.

Table A-1: **PetaLinux Project Description *(Cont'd)***

File / Directory in a PetaLinux Project	Description
"<plnx-proj-root>/components/plnx_workspace/device-tree-generation/"	Device tree files used to build device tree. The following files are auto generated by petalinux-config: • skeleton.dtsi (Zynq-7000 only) • zynq-7000.dtsi (Zynq-7000 only) • zynqmp.dtsi (Zynq UltraScale+ MPSoC only) • pcw.dtsi (Zynq-7000 and Zynq UltraScale+MPSoC only) • pl.dtsi • system-conf.dtsi • system-top.dts • <bsp name>.dtsi It is not recommended to edit these files, as these files are regenerated by the tools.
"<plnx-proj-root>/project-spec/meta-user/recipes-bsp/device-tree/files/"	system-user.dtsi is not modified by any PetaLinux tools. This file is safe to use with revision control systems. In addition, you can add your own DTSI files to this directory. You have to edit the <plnx-proj-root>/project-spec/meta-user/recipes-bsp/device-tree/device-tree-generation_%.bbappend by adding your dtsi file.
"<plnx-projroot>/project-spec/meta-plnx-generated/recipes-bsp/u-boot/configs"	U-Boot PetaLinux configuration files. The following files are auto generated by petalinux-config: • config.mk for MicroBlaze only • platform-auto.h • config.cfg platform-top.h will not be modified by any PetaLinux tools. When U-Boot builds, these files are copied into U-Boot build directory build/linux/u-boot/src/<U_BOOT_SRC>/ as follows: • config is the u-boot kconfig file. • config.mk is copied to board/xilinx/microblaze-generic/ for MicroBlaze.
<plnx-projroot>/project-spec/meta-user/recipes-bsp/u-boot/files/platform-top.h	• platform-auto.h and platform-top.h is copied to include/configs/ directory.
"<plnx-proj-root>/components/"	Directory for embeddedSW workspace and place to hold external sources while packing BSP. You can also manually copy components into this directory. Here is the rule to place a external component: "<plnx-proj-root>/components/ext_source/<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.
- "<plnx-proj-root>/build/tmp" for the files generated by Yocto. This directory is configurable through petalinux-config.

Here is an example:

```
<plnx-proj-root>
    -build
        -bitbake.lock
        -build.log
        -config.log
        -cache/
        -conf/
        -downloads/
        -misc/
            -config/
            -plnx-generated/
            -rootfs_config/
      -sstate-cache/
      -tmp/
    -components
        -plnx_workspace/
    -config.project
    -hardware
    -images
        -linux/
    -pre-built
        -linux/
    -project-spec
        -attributes
        -configs/
            -config
            -rootfs_config
        -hw-description/
        -meta-plnx-generated/
        -meta-user/
```

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

The table below is an example for Zynq UltraScale+ MPSoC.

Note: By default the build artifacts are removed to preserve space after petalinux-build. To preserve the build artifacts, you have to remove INHERIT += "rm_work" from build/conf/local.conf, but it increases the project-space.

Table A-2: **Build Directory in a PetaLinux Project**

Build Directory in a aPetaLinux Project	Description
"<plnx-proj-root>/build/build.log"	Logfile of the build
"<plnx-proj-root>/build/misc/config/"	Directory to hold files related to the linux subsystem build
"<plnx-proj-root>/build/misc/rootfs_config/"	Directory to hold files related to the rootfs build
"<plnx-projroot>/build/tmp/work/plnx_aarch64-xilinx-linux/petalinux-ser-image/1.0-r0/rootfs	Target rootfs host copy. This is the staging directory.
"<plnx-projroot>/tmp/plnx_aarch64"	Stage directory to hold the libs and header files required to build user apps/libs
"<plnx-proj-root>/build/tmp/work/plnx_aarch64-xilinx-linux/linux-xlnx"	Directory to hold files related to the kernel build
"<plnx-proj-root>/build/tmp/work/plnx_aarch64-xilinx-linux/u-boot-xlnx"	Directory to hold files related to the u-boot build
"<plnx-proj-root>/components/plnx_workspace/device-tree-generation"	Directory to hold files related to the device-tree build
"<plnx-projroot>/components/plnx_workspace/fsbl"	Directory to hold files related to the bootloader build

Table A-3: **Image Directory in a PetaLinux Project**

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

Project Layers

The PetaLinux project has two following layers under `<proj-plnx-root>/project-spec`

1. meta-plnx-generated:

This layer holds all bbappends and configuration fragment (cfg) for all components. All files in this layer are generated by the tool based on HDF and user configuration. The files in this layer should not be updated manually, as it is regenerated for `petalinux-config` and `petalinux-build` commands.

2. meta-user:

This layer is a place holder for all user-specific changes. You can add your own bbappend and configuration files in this layer.

Priority of the layers:

By default, meta-plnx-generated layer has the highest priority as the tool does not allow tainting of the system based on user-configuration. If you want to taint the system, change the priority of the meta-user to higher than meta-plnx-generated.

By default, meta-plnx-generated is at priority 6 and meta-user is at priority 6.

If you want higher priority for your changes, set meta-user priority to 7.

Note: Changing the meta-user priority is tainting of system. It is user responsibility for stability of system.

Steps to change priority of meta-user layer to 7

1. Open `<plnx-proj-plnx-root>/project-spec/meta-user/conf/layer.conf`.
2. Change `BBFILE_PRIORITY_meta-user` value from 6 to 7.
3. The meta-user priority is now changed to 7.

```
BBFILE_PRIORITY_meta-user = "7"
```

Generating First Stage Bootloader, PMU Firmware and Arm Trusted Firmware Within Project

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

CAUTION! *If you do not want the PetaLinux build 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 "linux Components Selection --->" sub-menu.

   b. Select “First Stage Bootloader” option.

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

   c. Exit the menu and save the change.

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

   ```
   components/plnx_workspace/fsbl/zynqmp_fsbl
   ```

   For Zynq-7000, it will be:

   ```
   components/plnx_workspace/fsbl/zynq_fsbl
   ```

   For MicroBlaze, it will be:

   ```
   components/plnx_workspace/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: *`fsbl_bsp`, `fsbl_bsp` will be auto updated when you run `petalinux-build`.*

Arm Trusted Firmware (ATF)

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

You can set the ATF configurable options as follows:

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

   ```
   $ petalinux-config
   ```

   a. Click into "ARM Trusted Firmware Compilation Configuration" ---> submenu.

   b. Enter your settings.

   c. Exit the menu and save the change.

2. 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.

PMU Firmware

This is for Zynq UltraScale+ MPSoC only. This is Optional. By default, the top level system settings are set to generate the PMU Firmware (PMUFW).

CAUTION! *If the user wishes not to have PetaLinux build the PMUFW, 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 PMUFW as follows:

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

   ```
   $ petalinux-config
   ```

   a. Click "linux Components Selection --->" sub-menu.

   b. Select "PMU Firmware" option.

   ```
   [*] PMU Firmware
   ```

   c. Exit the menu and save the change.

2. Build the PMUFW when building the project:

   ```
   $ petalinux-build
   ```

   Build the PMUFW only:

   ```
   $ petalinux-build -c pmufw
   ```

   The PMUFW ELF file will be installed as `pmufw.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 8 KB 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 header followed by a BIN file. FS-Boot gets the target memory location from wrapper. The wrapper needs to contain the following information:

Table B-1: **Wrapper Information**

Offset	Description	Value
0×0	FS-Boot bootable image magic code	0×b8b40008
0×4	BIN image size	User defined
0×100	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.
0×10c	Where the BIN file start	None

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

Auto Config Settings

When you run `petalinux-config`, you will see the "`Auto Config Settings`" sub-menu. If you click in the sub-menu, 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.

Table C-1: **Auto Config Settings**

Component in the Menu	Files impacted when autoconfig is enabled
Device tree	• `<plnx-proj-root>/components/plnx_workspace/device-tree-generation/skeleton.dtsi` (Zynq-7000 only) • `<plnx-proj-root>/components/plnx_workspace/device-tree-generation/zynq-7000.dtsi` (Zynq-7000 only) • `<plnx-proj-root>/components/plnx_workspace/device-tree-generation/zynqmp.dtsi` (Zynq UltraScale+ MPSoC only) • `<plnx-proj-root>/components/plnx_workspace/device-tree-generation/zynqmp-clk.dtsi` (Zynq UltraScale+ MPSoC only) • `<plnx-proj-root>/components/plnx_workspace/device-tree-generation/pcw.dtsi` (Zynq-7000 and Zynq UltraScale+ MPSoC) • `<plnx-proj-root>/components/plnx_workspace/device-tree-generation/pl.dtsi` (Microblaze only) • `<plnx-proj-root>/components/plnx_workspace/device-tree-generation/system-conf.dtsi` • `<plnx-proj-root>/components/plnx_workspace/device-tree-generation/system-top.dts`
kernel	`<plnx-proj-root>/project-spec/meta-plnx-generated/recipes-kernel/linux/configs/plnx_kernel.cfg`
u-boot	• `<plnx-proj-root>/project-spec/meta-plnx-generated/recipes-bsp/u-boot/configs/config.cfg` • `<plnx-proj-root>/project-spec/meta-plnx-generated/recipes-bsp/u-boot/configs/platform-auto.h`
fsbl	`<plnx-proj-root>/components/plnx_workspace/fsbl`
pmufw	`<plnx-proj-root>/components/plnx_workspace/pmu-firmware`
fsboot	`<plnx-proj-root>/components/plnx_workspace/fs-boot`

QEMU Virtual Networking Modes

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

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 (For example, 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 utilizes the `--qemu-args` option to perform this redirection. The following table outlines some example redirection arguments. This is standard QEMU functionality, refer to the QEMU documentation for more details.

Table D-2: **Redirection Arguments**

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

Table D-2: **Redirection Arguments *(Cont'd)***

QEMU Options Switch	Purpose	Accessing guest from host
-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:

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

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

For example, to use subnet `192.168.20.*`:

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

Xilinx IP Models Supported by QEMU

The QEMU emulator shipped in PetaLinux tools supports the following Xilinx IP models:

- 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).*

For more information refer to *Zynq UltraScale+ MPSoC Quick Emulator User Guide* (UG1169) [Ref 5].

Xen Zynq Ultrascale+ MPSoC Example

This section details on the Xen Zynq® Ultrascale+™ MPSoC example. It describes 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. 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`, `xen-image` and `xen-rootfs.cpio.gz.u-boot`.

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/xen-Image <tftpboot>/
$ cp pre-built/linux/images/xen-rootfs.cpio.gz.u-boot <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-boots

```
U-Boot-PetaLinux> tftpboot D80000 xen.dtb
U-Boot-PetaLinux> tftpboot 80000 xen-Image
U-Boot-PetaLinux> tftpboot 1000000 xen.ub
U-Boot-PetaLinux> tftpboot 2000000 xen-rootfs.cpio.gz.u-boot
U-Boot-PetaLinux> bootm 1000000 2000000 D80000
```

Rebuild Xen

After creating a PetaLinux project for Zynq Ultrascale + MPSoC, follow the below steps to build xen images:

1. Go to `cd <proj root directory>`
2. In the `Petalinux-config` command, select 'Image Packaging Configuration and then 'Root filesystem type (INITRD)'
3. In `Petalinux-config -c rootfs`, select Filesystem Packages ---> misc ---> packagegroup-petalinux-xen ---> [*] packagegroup-petalinux-xen
4. Edit the edit the device tree to build in the extra Xen related configs. Edit this file: `project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi` and add this line:

   ```
   /include/ "xen-overlay.dtsi".
   ```

   It should look like the following:

   ```
   /include/ "system-conf.dtsi"
   /include/ "xen-overlay.dtsi"
   / {
   };
   ```

5. Edit the file: `project-spec/meta-user/recipes-bsp/device-tree/device-tree-generation_%.bbappend` and add this line to it: `file://xen-overlay.dtsi`.

   The file should look like this:

   ```
   SRC_URI_append ="\
       file://system-user.dtsi \
       file://xen-overlay.dtsi \
   "
   FILESEXTRAPATHS_prepend := "${THISDIR}/files:"
   ```

6. Run petalinux-build: `$ petalinux-build`
7. The build artifacts will be in `images/linux` in the project directory.

Note: By default, the petalinux-build command does not build Xen. The default root file system does not contain the Xen tools. You have to use Xen rootfs.

TIP: *For more information on XEN, refer to* Building the Xen Hypervisor with PetaLinux 2016.4 and newer.

Execute OpenAMP

Use the following steps to execute OpenAMP:

1. Go to `$ cd <plnx-proj-root>`

   ```
   $ cp pre-built/linux/images/openamp.dtb pre-built/linux/images/system.dtb
   $petalinux-boot --jtag --prebuilt 3 --hw_server-url <hostname:3121>
   ```

2. Login to linux
   a. echo <echo_test_firmware> > /sys/class/remoteproc/remoteproc0/firmware. For example, # echo image_echo_test > /sys/class/remoteproc/remoteproc0/firmware
   b. echo start > /sys/class/remoteproc/remoteproc0/state
   c. modprobe rpmsg_user_dev_driver
   d. echo_test

For more information on OpenAMP, refer to *OpenAMP Framework for Zynq Devices (UG1186)* [Ref 7]

Obsolete Features

This section details the obsolete features/options in the PetaLinux Commands.

1. petalinux-create
    a. -t libs: You have to use the apps methodology to develop or install libraries.
    b. -t libs --priority: This option is no longer supported.
    c. -t generic: Generic components are no longer supported
    d. --out: This option can be used to specify if the project location is not supported. '-n' can also be used.
2. petalinux-config
    a. --search: The entire search path implementation and all its sub options are no longer supported.
3. petalinux-build
    a. --makeenv - You have to provide your flags in the corresponding bbappends or in Makefile
4. petalinux-package
    a. --bsp --no-extern: You have to explicitly copy the local external components.
    b. --bsp --no-local: This option is no longer supported.
    c. --firmware: This option is no longer supported.
    d. --image -c rootfs: You have to choose the type of File system from petalinux-config
5. petalinux-boot
    a. --qemu --qemu-gdb: You can directly use --qemu-args option to ask qemu to launch gdb.
6. petalinux-util
    a. --update-sdcard: You have to update the sdcard manually. For more details, refer to Configuring SD Card ext filesystem Boot.

Common Errors and Recovery

This section details the common errors that appear, while working with the PetaLinux commands, and also lists their recovery steps in detail.

The common errors and their recovery methods are:

1. TMPDIR on NFS

   "ERROR: OE-core's config sanity checker detected a potential misconfiguration"

   Either fix the cause of this error or disable the checker at your own risk (see sanity.conf). Following is the list of potential problems / advisories:

   ```
   The TMPDIR: /home/user/xilinx-kc705-axi-full-2017.1/build/tmp cannot be located on NFS.
   ```

   The TMPDIR cannot be on NFS, therefore, it will throw an error while parsing. You have to change it from petalinux-config and then provide any local storage.

   ```
   Yocto-settings -->  TMPDIR
   ```

   Do not configure the same TMPDIR for two different PetaLinux projects. This can cause build errors.

2. Bitbake cannot run always.

   Only one instance of bitbake runs for one project. If the previous build is exited abruptly, there are chances for bitbake exiting non-gracefully for the current build.

   ```
   to reciver: rm -rf <plnx-proj-root>/build/bitbake.lock
   ```

3. App/module name having ' _ '

   If the app name is "`plnx_myapp`", bitbake throws an error. A version number has to be entered after ' _ '.

   For example, myapp_1 is an accurate app/module name.

   To recover, you have delete the app created and also delete the line in `<plnx-proj-root>/project-spec/meta-user/recipes-core/images/petalinux-image.bbappend`.

   ```
   IMAGE_INSTALL_append = " plnx_myapp"
   ```

4. When PetaLinux is exited forcefully by entering Ctrl+c twice, the following error appears:

```
NOTE: Sending SIGTERM to remaining 1 tasks
Error in atexit._run_exitfuncs:
Traceback (most recent call last):
  File
"/opt/pkg/petalinux/components/yocto/source/aarch64/layers/core/bitbake/lib/bb/ui/k
notty.py", line 313, in finish
    self.termios.tcsetattr(fd, self.termios.TCSADRAIN, self.stdinbackup)
termios.error: (5, 'Input/output error')
```

After this error, the console is broken, you cannot see the text that you typed. To restore the console, enter stty sane and press enter twice.

5. Language settings missing. If you see the error "Could not find the /log/cooker/plnx_microblaze in the /tmp directory" during petalinux-config.

   To resolve, set the following:

   ```
   export LC_ALL=en_US.UTF-8
   export LANG=en_US.UTF-8
   export LANGUAGE=en_US.UTF-8
   ```

6. For petalinux-config -c kernel and u-boot bitbake tries to open a new terminal inside, sometimes it fails. Error:

   ```
   ERROR: linux-xlnx-4.9-xilinx+gitAUTOINC+a96b8d2e1c-r0 do_menuconfig: Unable to spawn
   terminal auto:
   Execution of
   '/home/user/xilinx-zcu102-2017.1/build/tmp/work/plnx_aarch64-xilinx-linux/linux-xln
   x/4.9-xilinx+gitAUTOINC+a96b8d2e1c-r0/temp/run.do_terminal.8249' failed with exit
   code -6:
   Fatal Python error: Py_Initialize: Unable to get the locale encoding
   ImportError: No module named 'encodings'
   Current thread 0x00007fd12b4f8700 (most recent call first):
   ERROR: linux-xlnx-4.9-xilinx+gitAUTOINC+a96b8d2e1c-r0 do_menuconfig: Function
   failed: do_menuconfig
   ERROR: Logfile of failure stored in:
   /home/user/xilinx-zcu102-2017.1/build/tmp/work/plnx_aarch64-xilinx-linux/linux-xlnx
   /4.9-xilinx+gitAUTOINC+a96b8d2e1c-r0/temp/log.do_menuconfig.8249
   ```

   You have to change the OE_TERMINAL as it is not able to get through default. Uncomment OE_TERMINAL in <plnx-proj-root>/project-spec/meta-user/conf/petalinuxbsp.conf and set it to xterm or screen. For this, you are required to have the corresponding utility installed in your PC.

Appendix I

Additional Resources and Legal Notices

Xilinx Resources

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

Solution Centers

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

References

1. *PetaLinux Documentation* (www.xilinx.com/petalinux)
2. Xilinx Answer Record (55776)
3. *Ultrascale+ MPSoC Software Developer Guide* (UG1137)
4. *PetaLinux Tools Documentation: Command Line Reference* (UG1157)
5. *Zynq Ultrascale+ MPSoC Quick Emulator User Guide* (UG1169)
6. *PetaLinux Tools Documentation: Workflow Tutotial* (UG1156)
7. *OpenAMP Framework for Zynq Devices* (UG1186)

Please Read: Important Legal Notices

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

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.