PetaLinux Yocto Tips

Xilinx Wiki

Exported on 11/23/2023

Table of Contents

This page is intended to be a collection place for tips and tricks related to Yocto layers and how Yocto works under Petalinux.

1 Table of Contents

2 Documentation

The first place you should start to better understand many details of the Yocto project is the Yocto project website[1].

1 https://www.yoctoproject.org/documentation

3 Override syntax changes from Yocto honister/PetaLinux 2022.1

From 2022 release, the : character replaces the use of _ to refer to an override, most commonly when making a conditional assignment of a variable.

Please refer to https://xilinx-wiki.atlassian.net/wiki/spaces/XWUC/pages/2329444389for more details.

To help with migration of layers, a script has been provided in OE-Core. Once configured with the overrides used by a layer, this can be run as:

With in PetaLinux:

```
$ cd <plnx-proj-root>
$ $PETALINUX/components/yocto/buildtools/sysroots/x86_64-petalinux-linux/usr/bin/
python3 components/yocto/layers/core/scripts/contrib/convert-overrides.py <meta-
layer-path>
```

With in Yocto:

```
$ cd <yocto-proj-dir>
$ sources/core/scripts/contrib/convert-overrides.py <meta-layer-path>
```

To execute above convert-overrides.py you need python3 should be installed on host machine.

4 Working with a User Space Yocto Layer

This tip has been adapted to PetaLinux using the meta-example layer available here meta-example[2]

4.1 Installing the meta-example layer on your host

```
$ git clone https://github.com/DynamicDevices/meta-example
```

4.2 About the meta-example layer

This example layer is providing 3 build recipes, bbexample, bbexample-rt and bbexample-lt respectively fetching the source code from a git tree, a remote tar file and a local tar file

```
$ petalinux-build -c bbexample
$ petalinux-build -c bbexample-rt
$ petalinux-build -c bbexample-lt
```

4.3 Configuring the layer path in the PetaLinux build system

Move to your PetaLinux project directory, add the layer path in below config option.

```
cd <plnx-proj-root>
petalinux-config --> Yocto Settings --> User Layers --> (/
absolute_path_to_meta_layer/meta-example) user layer 0
                                                --> ()    user layer 1
```

Move to your Yocto project build directory, edit the **conf/bblayers.conf** file and add your layer path to **BBLAYERS**

```
BBLAYERS := " \
  ${SDKBASEMETAPATH}/layers/poky/meta \
  ${SDKBASEMETAPATH}/layers/poky/meta-poky \
```

2 https://github.com/DynamicDevices/meta-example

```
  /absolute_path_to_meta_layer/meta-example \
  "
```

4.4 Building the meta-layer example package based on a local source archive

The recipe we are going to build is https://github.com/DynamicDevices/meta-example/blob/master/recipes-example/bbexample/bbexample-lt_1.0.bb

The recipe bbexample-lt is using some variables

1. **SRC_URI** Space-delimited list of URIs to download source code, patches, and other files from git, local absolute path, https, ftp etc.
2. **PN** is the Package Name. The value of this variable is derived by BitBake from the base name of the recipe file.
3. **PV** is the Package Version. which is derived by BitBake from the base name of the recipe file

```
# Use local tarball
SRC_URI = "file://bbexample-${PV}.tar.gz"
```

The recipe **bbexample-lt** can invoked using the following command

```
$ petalinux-build -c bbexample-lt
```

here is the command output

```
$ petalinux-build -c bbexample-lt

[INFO] building bbexample-lt
[INFO] sourcing bitbake
INFO: bitbake bbexample-lt
Loading cache: 100% |
################################################################################
################################################################################
#####| ETA:  00:00:00
Loaded 2941 entries from dependency cache.
Parsing recipes: 100% |
################################################################################
################################################################################
###| Time: 00:00:01
Parsing of 2328 .bb files complete (2294 cached, 34 parsed). 2943 targets, 196
 skipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
```

```
NOTE: Preparing RunQueue
NOTE: Checking sstate mirror object availability (for 38 objects)
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
NOTE: Tasks Summary: Attempted 872 tasks of which 857 didn't need to be rerun and all
succeeded.
INFO: Copying Images from deploy to images
[INFO] successfully built bbexample-lt
```

You have now successfully built the layer but you still need to include the binary produced into the kernel root file system

4.5 Including the meta-layer example build output in the Linux root file system

There are two options in order to do so

1. Edit your project **<BUILD_DIR>/conf/local.conf** file for YOCTO project **<plnx-proj-root>/meta-user/conf/petalinuxbsp.conf** file for PetaLinux project and add

```
# Use Below syntax For 2022.1 Release onwards
IMAGE_INSTALL:append = " bbexample-lt"

# Use Below syntax For 2021.2 and Old Releases
IMAGE_INSTALL_append = " bbexample-lt"
```

2. Adding a package to an image via a **.bbappend**

You may wish to add specific packages to specific images, which is generally viewed as better practice. We are using **core-image-minimal** bitbake recipe for this tip by creating file **core-image-minimal.bbappend.** This .bbappend file is extending the original core-image-minimal recipe in order to include the layer build output into the Linux root file system.

```
recipes-core/images/core-image-minimal.bbappend
```

core-image-minimal.bbappend file content

```
IMAGE_INSTALL += " bbexample-lt"
```

4.6 Running the meta-layer example under QEMU

1. Rebuild your layer

```
$ petalinux-build -c bbexample_lt -x cleansstate
```

2. Rebuild your kernel

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

3. Start qemu

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

4. Log on qemu and start the application

```
root@plnx_arm:/# /usr/bin/bbexample
Hello Yocto World...
Hello World (from a shared library!)
```

5 Patching the Linux Kernel of a PetaLinux Project

1. Copy the patch to project file *<plnx-proj-root>/project-spec/meta-user/recipes-kernel/linux/linux-xlnx* directory.

2. Modify project file *<plnx-proj-root>/project-spec/meta-user/recipes-kernel/linux/linux-xlnx_%.bbappend* to use the patch file by adding the patch file name to the SRC_URI variable. If the variable does not exist in the file then add a new line with
For 2022.1 release onwards :

```
SRC_URI:append = " file://0001-linux-driver-fix.patch"

FILESEXTRAPATHS:prepend := "${THISDIR}/${PN}:"
```

For 2021.2 and old release :

```
SRC_URI_append = " file://0001-linux-driver-fix.patch"

FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"
```

3. Make sure the priority for the meta-user layer is 7 in the project file *<plnx-proj-root>/project-spec/meta-user/conf/layer.conf* .

Any patches in the project will not be applied to an external source tree for the Linux kernel or u-boot. The user should apply patches to the external source tree.

6 How to Modify inittab or getty in a PetaLinux Project

1. Create a sysvinit directory in meta-user layer as

```
$ mkdir -p <plnx-proj-root>/project-spec/meta-user/recipes-core/sysvinit/sysvinit-inittab
```

2. Create a *sysvinit-inittab_%.bbappend* file and your own inittab file or your inittab patch

```
$ vim <plnx-proj-root>/project-spec/meta-user/recipes-core/sysvinit/sysvinit-inittab_%.bbappend
```

For 2022.1 release onwards :

```
# For Files
SRC_URI:append = " file://inittab"
# For patches
SRC_URI:append = " file://inittab.patch"

FILESEXTRAPATHS:prepend := "${THISDIR}/${PN}:"
```

For 2021.2 and old release :

```
# For Files
SRC_URI_append = " file://inittab"
# For patches
SRC_URI_append = " file://inittab.patch"

FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"
```

7 How to Patch the FSBL in a PetaLinux Project

1. Create a fsbl/embeddedsw and files directory in meta-user layer as

```
$ mkdir -p <plnx-proj-root>/project-spec/meta-user/recipes-bsp/fsbl/files
## From 2021.1 onwards
$ mkdir -p <plnx-proj-root>/project-spec/meta-user/recipes-bsp/embeddedsw/files
```

2. Copy patch files to *<plnx-proj-root>/project-spec/meta-user/recipes-bsp/fsbl/files* as

```
$ cp 0001-FSBL.patch <plnx-proj-root>/project-spec/meta-user/recipes-bsp/fsbl/files
## From 2021.1 onwards
$ cp 0001-FSBL.patch <plnx-proj-root>/project-spec/meta-user/recipes-bsp/embeddedsw/
files
```

Note that patches should be generated based on the embeddedsw GIT repo which contains FSBL as the patches require the path into the repo to apply correctly.

3. Create a *fsbl_%.bbappend* file and add below content

```
$ vim <plnx-proj-root>/project-spec/meta-user/recipes-bsp/fsbl/fsbl_%.bbappend
## From 2021.1 onwards
$ vim <plnx-proj-root>/project-spec/meta-user/recipes-bsp/embeddedsw/fsbl-
firmware_%.bbappend
```

For 2022.1 release onwards :

```
SRC_URI:append = " \
        file://0001-FSBL.patch \
        "

FILESEXTRAPATHS:prepend := "${THISDIR}/files:"

#Add debug for FSBL(optional)
XSCTH_BUILD_DEBUG = "1"

#Enable appropriate FSBL debug or compiler flags
YAML_COMPILER_FLAGS:append = " -DXPS_BOARD_ZCU102"
```

For 2021.2 and old release :

```
SRC_URI_append = " \
```

```
        file://0001-FSBL.patch \
        "

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

#Add debug for FSBL(optional)
XSCTH_BUILD_DEBUG = "1"

#Enable appropriate FSBL debug or compiler flags
YAML_COMPILER_FLAGS_append = " -DXPS_BOARD_ZCU102"
```

4. Remove the *<plnx-proj-root>/components/plnx_workspace* and clean your project workspace before rebuilding FSBL components

```
$ petalinux-build -x mrproper

#Note: In v2018.1 PetaLinux release onwards "petalinux-build -x mrproper" command
will remove <plnx-proj-root>/components/plnx_workspace directory
$ rm -rf <plnx-proj-root>/components/plnx_workspace
```

5. Rebuilding FSBL components

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

8 How to Patch the PMU Firmware/PLM/PSM Firmware in a PetaLinux Project

1. Create a pmu/pmu-firmware and files directory in meta-user layer as

```
#For v2018.1 release onwards
$ mkdir -p <plnx-proj-root>/project-spec/meta-user/recipes-bsp/pmu-firmware/files
#For 2021.1 release onwards
$ mkdir -p <plnx-proj-root>/project-spec/meta-user/recipes-bsp/embeddedsw/files
```

2. Copy patch files to *<plnx-proj-root>/project-spec/meta-user/recipes-bsp/pmu/files* as

```
#For v2018.1 release onwards
$ cp 0001-support.patch <plnx-proj-root>/project-spec/meta-user/recipes-bsp/
embeddedsw/files
```

3. Create a corresponding bbappend file to that recipe as below.

Component/recipe name	bbappend file name
PMUFW	pmu-firmware_%.bbappend
PSMFW	psm-firmware_%.bbappend
PLM	plm-firmware_%.bbappend

```
#For v2018.1 or later PetaLinux releases only
$ vim <plnx-proj-root>/project-spec/meta-user/recipes-bsp/pmu-firmware/<bbappend file
name>

#For v2021.1 or later PetaLinux releases
vim <plnx-proj-root>/project-spec/meta-user/recipes-bsp/embeddedsw/<bbappend file
name>
```

For 2022.1 release onwards :

```
# Patch for PMUFW

SRC_URI:append = " file://0001-support.patch"
```

```
FILESEXTRAPATHS:prepend := "${THISDIR}/files:"
```

For 2021.2 and old release :

```
# Patch for PMUFW

SRC_URI_append = " file://0001-support.patch"
FILESEXTRAPATHS_prepend := "${THISDIR}/files:"
```

5. Rebuilding PMUFW components

```
$ petalinux-build -c pmufw (or)
$ petalinux-build -c psmfw (or)
$ petalinux-build -c plm
```

9 Configuring the FSBL component in a PetaLinux Project

1. Create a *fsbl bbappend* file and add below content

```
$ vim <plnx-proj-root>/project-spec/meta-user/recipes-bsp/fsbl/fsbl_%.bbappend
# From 2021.1 onwards
$ vim <plnx-proj-root>/project-spec/meta-user/recipes-bsp/embeddedsw/fsbl-
firmware_%.bbappend
```

2. To enable debugs in FSBL:

```
#Add debug for FSBL
XSCTH_BUILD_DEBUG = "1"
```

3. To add compiler flags in FSBL:

```
#Add compiler flags for FSBL
#For v2022.1 or later PetaLinux releases
YAML_COMPILER_FLAGS:append = " -DFSBL_PROT_BYPASS"

#For v2021.2 and old PetaLinux releases
YAML_COMPILER_FLAGS_append = " -DFSBL_PROT_BYPASS"
```

4. To add BSP flags(secure mode) in FSBL:

```
#Add BSP flags for FSBL
YAML_BSP_CONFIG += "secure_mode"
YAML_BSP_CONFIG[secure_mode] = "set,true"
```

5. Build the FSBL

```
$ petalinux-build -c fsbl
```

10 Configuring the PMUFW component in a PetaLinux Project

1. Create a *pmu-firmware_%.bbappend* file and add below content

```
#For v2018.1 or later PetaLinux releases only
$ vim <plnx-proj-root>/project-spec/meta-user/recipes-bsp/pmu-firmware/pmu-
firmware_%.bbappend

# From 2021.1 onwards
$ vim <plnx-proj-root>/project-spec/meta-user/recipes-bsp/embeddedsw/pmu-
firmware_%.bbappend
```

2. To enable debugs in PMUFW refer this https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841724/PMU+Firmware page for DEBUG flags list

```
#Add debug for PMUFW
XSCTH_BUILD_DEBUG = "1"
```

3. To add compiler flags in PMUFW, see PMU FW Build Flags:[3]

> Note that the method of loading the PMUFW can prevent early prints (during code initialization) from being output. See PMU Firmware > Using FSBL to load PMUFW[4] for additional details.

```
#Add compiler flags for PMUFW
# From 2022.1 onwards
YAML_COMPILER_FLAGS:append = " -DDEBUG_MODE -DXPFW_DEBUG_DETAILED"

# For 2021.2 and old PetaLinux releases
YAML_COMPILER_FLAGS_append = " -DDEBUG_MODE -DXPFW_DEBUG_DETAILED"
```

4. To add BSP flags(secure mode) in PMUFW:

```
#Add BSP flags for PMUFW
YAML_BSP_CONFIG += "secure_mode"
YAML_BSP_CONFIG[secure_mode] = "set,true"
```

3 https://xilinx-wiki.atlassian.net/wiki/display/A/PMU+Firmware#PMU+FW+Build+Flags
4 https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841724/PMU+Firmware#PMUFirmware-UsingFSBLtoloadPMUFW

5. To add BSP flags(debug mode) in PMUFW: For example if you want to enable https://github.com/Xilinx/embeddedsw/blob/release-2018.3/lib/sw_apps/zynqmp_pmufw/misc/xfpga_config.h#L34 debug is Xil BSP libraries

```
#Add BSP flags for PMUFW
YAML_BSP_CONFIG += "debug_mode"
YAML_BSP_CONFIG[debug_mode] = "set,true"
```

6. By pass PMUFW debug logs on different UART console:

```
# From 2022.1 PetaLinux Release onwards
YAML_SERIAL_CONSOLE_STDIN:zynqmp = "psu_uart_1"
YAML_SERIAL_CONSOLE_STDOUT:zynqmp = "psu_uart_1"

# For 2021.2 and old releases
YAML_SERIAL_CONSOLE_STDIN_zynqmp = "psu_uart_1"
YAML_SERIAL_CONSOLE_STDOUT_zynqmp = "psu_uart_1"
```

7. Build the PMUFW

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

11 Configuring the PLM component in a PetaLinux Project

1. Create a *plm-firmware_%.bbappend* file and add below content

```
$ vim <plnx-proj-root>/project-spec/meta-user/recipes-bsp/embeddedsw/plm-
firmware_%.bbappend
```

2. To add BSP flags(to enable USB boot mode) in PLM: For more option see https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/2037088327/Versal+Platform+Loader+and+Manager#PLM-Build-Flags

```
#Add BSP flags for PLM. Here USB boot mode is enabled and debug level changed to
level0 (minimal)
#Alternatively, each of these two parameters can be configured separately
YAML_BSP_CONFIG += "plm_usb_en plm_dbg_lvl"
YAML_BSP_CONFIG[plm_usb_en] = "set,true"
YAML_BSP_CONFIG[plm_dbg_lvl] = "set,level0"
```

3. To build PLM

```
$ petalinux-build -c plm
```

12 How to Update psu_init Files in PetaLinux Project

> Note: PSU_INIT files are generated based on HDF or XSA parsing, Hence these files can't be patched on top of embedded_sw repo using Yocto SRC_URI variable. Alternatively we can update these files by editing <plnx-proj-root>/project-spec/hw-description/psu_init.c and copy to FSBL build workspace during do_compile_prepend bitbake task.

1. Modify *<plnx-proj-root>/project-spec/hw-description/psu_init.c* file as per your requirement

2. Create a FSBL bbappend file under <plnx-proj-root>/project-spec/meta-user/recipes-bsp/embeddedsw/fsbl-firmware_%.bbappend and add below content to update psu_init.c changes to FSBL

```
# For 2018.2 and below versions
do_compile_prepend(){
   install -m 0644 ${TOPDIR}/../project-spec/hw-description/psu_init.c ${TOPDIR}/../
components/plnx_workspace/fsbl/fsbl_hwproj/psu_init.c
}

# For 2018.3 and later versions we removed FSBL and PMUFW external workspace from
<plnx-proj-root>/components/plnx_workspace directory
# to align Yocto patching mechanism. Hence we are copying the files to build
workspace(${B})
# https://github.com/Xilinx/meta-xilinx-tools/blob/rel-v2018.3/classes/
xsctbase.bbclass#L14
do_compile_prepend(){
   install -m 0644 ${TOPDIR}/../project-spec/hw-description/psu_init.c ${B}/fsbl/
psu_init.c
}

## From 2021.1 onwards

do_compile_prepend(){
   install -m 0644 ${TOPDIR}/../project-spec/hw-description/psu_init.c ${B}/fsbl-
firmware/psu_init.c
}

# For 2022.1 release onwards
do_compile:prepend(){
   install -m 0644 ${TOPDIR}/../project-spec/hw-description/psu_init.c ${B}/fsbl-
firmware/psu_init.c
}
```

13 How to Add Pre-built Libraries in PetaLinux or Yocto Projects

1. Create an application/recipe and copy prebuilt library and header files with the following commands. Assuming prebuilt library name is libcpsample.so.1.0.1.

13.1 In PetaLinux Flow:

```
$ petalinux-create -t apps --template install --name libcpsample --enable
$ cp libcpsample.so.1.0.1 <plnx-proj-root>/project-spec/meta-user/recipes-apps/
libcpsample/files
$ cp cpsample.h <plnx-proj-root>/project-spec/meta-user/recipes-apps/libcpsample/
files
```

(or) You can use the below single command to achieve the above functionality.

```
$ petalinux-create -t apps --template install --name libcpsample --enable --srcuri
"<path-to-dir>/libcpsample.so.1.0.1 <path-to-dir>/cpsample.h>"
```

13.2 In Yocto Flow:

```
$ mkdir -p <layer-path>/recipes-apps/libcpsample/files
$ touch <layer-path>/recipes-apps/libcpsample/libcpsample.bb
$ cp libcpsample.so.1.0.1 <layer-path>/recipes-apps/libcpsample/files
$ cp cpsample.h <layer-path>/recipes-apps/libcpsample/files
```

2. Add the below content to recipe(bb) file
For 2022.1 PetaLinux release onwards:

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

SUMMARY = "Sample pre-built library copy to rootfs"
SECTION = "libs"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/
MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

SRC_URI = " \
```

```
    file://libcpsample.so.1.0.1 \
    file://cpsample.h \
"

S = "${WORKDIR}"

# Add dependency libraries if any
# for example DEPENDS = "libpcap"

# If you need to create a symbolic link using the pre-built libraries you should use
oe_soinstall.
# This copies libraries to "{TARGET_ROOTFS}/usr/lib" directory and create a symlink
as
# lrwxrwxrwx libcpsample.so.1.0 -> libcpsample.so.1.0.1
# -rwxr-xr-x libcpsample.so.1.0.1

do_install() {
    install -d ${D}${libdir}
    oe_soinstall ${S}/libcpsample.so.1.0.1 ${D}${libdir}
    install -d -m 0655 ${D}${includedir}/CPSAMPLE
    install -m 0644 ${S}/*.h ${D}${includedir}/CPSAMPLE/
}

# Inhibit warnings about files being stripped
INSANE_SKIP:${PN} = "ldflags"
INSANE_SKIP:${PN} = "already-stripped"

# If you don't have .h file to copy to /usr/include add something like below
# FILES:${PN} = "${libdir}/*.so.*"

FILES:${PN} = "${libdir}/*.so.* ${includedir}/*"
FILES:${PN}-dev = "${libdir}/*.so"
```

For 2021.2 and Below PetaLinux releases:

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

SUMMARY = "Sample pre-built library copy to rootfs"
SECTION = "libs"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/
MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

SRC_URI = " \
    file://libcpsample.so.1.0.1 \
    file://cpsample.h \
"

S = "${WORKDIR}"
```

```
# Add dependency libraries if any
# for example DEPENDS = "libpcap"

# If you need to create a symbolic link using the pre-built libraries you should use
oe_soinstall.
# This copies libraries to "{TARGET_ROOTFS}/usr/lib" directory and create a symlink
as
# lrwxrwxrwx libcpsample.so.1.0 -> libcpsample.so.1.0.1
# -rwxr-xr-x libcpsample.so.1.0.1

do_install() {
    install -d ${D}${libdir}
    oe_soinstall ${S}/libcpsample.so.1.0.1 ${D}${libdir}
    install -d -m 0655 ${D}${includedir}/CPSAMPLE
    install -m 0644 ${S}/*.h ${D}${includedir}/CPSAMPLE/
}

# Inhibit warnings about files being stripped
INSANE_SKIP_${PN} = "ldflags"
INSANE_SKIP_${PN} = "already-stripped"

# If you don't have .h file to copy to /usr/include add something like below
# FILES_${PN} = "${libdir}/*.so.*"

FILES_${PN} = "${libdir}/*.so.* ${includedir}/*"
FILES_${PN}-dev = "${libdir}/*.so"
```

4. Build the pre-built library recipe

```
$ petalinux-build -c rootfs
```

14
Creating Libraries in a PetaLinux Project

1. Create a library using PetaLinux tools

```
$ petlainux-create -t apps --template c --name libsample --enable
```

2. Modify the *libsample.bb* recipe file *<plnx-proj-root>/project-spec/meta-user/recipes-apps/libsample/libsample.bb* as
For 2022.1 PetaLinux release onwards:

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

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

FILESEXTRAPATHS:prepend := "${THISDIR}/files:"

SRC_URI = " \
          file://sample.c \
          file://sample.h \
          file://Makefile \
         "

S = "${WORKDIR}"

PACKAGE_ARCH = "${MACHINE_ARCH}"
PROVIDES = "sample"
TARGET_CC_ARCH += "${LDFLAGS}"

do_install() {
    install -d ${D}${libdir}
    install -d ${D}${includedir}
    oe_libinstall -so libsample ${D}${libdir}
    # This is optional and depends if you have any headers to copied along with
libraries
    # This example includes sample.h to to copied to <TARGET_ROOTFS>/usr/lib/SAMPLE/
sample.h
    install -d -m 0655 ${D}${includedir}/SAMPLE
    install -m 0644 ${S}/*.h ${D}${includedir}/SAMPLE/
```

```
}

FILES:${PN} = "${libdir}/*.so.* ${includedir}/*"
FILES:${PN}-dev = "${libdir}/*.so"
```

For 2021.2 and Below PetaLinux releases:

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

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

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

SRC_URI = " \
          file://sample.c \
          file://sample.h \
          file://Makefile \
         "

S = "${WORKDIR}"

PACKAGE_ARCH = "${MACHINE_ARCH}"
PROVIDES = "sample"
TARGET_CC_ARCH += "${LDFLAGS}"

do_install() {
    install -d ${D}${libdir}
    install -d ${D}${includedir}
    oe_libinstall -so libsample ${D}${libdir}
    # This is optional and depends if you have any headers to copied along with
libraries
    # This example includes sample.h to to copied to <TARGET_ROOTFS>/usr/lib/SAMPLE/
sample.h
    install -d -m 0655 ${D}${includedir}/SAMPLE
    install -m 0644 ${S}/*.h ${D}${includedir}/SAMPLE/
}

FILES_${PN} = "${libdir}/*.so.* ${includedir}/*"
FILES_${PN}-dev = "${libdir}/*.so"
```

3. Modify the *<plnx-proj-root>/project-spec/meta-user/recipes-apps/libsample/files/Makefile* generated from PetaLinux tools as

```
APP = sample

LIBSOURCES=*.c
OUTS = *.o
NAME := sample
MAJOR = 1.0
MINOR = 1
VERSION = $(MAJOR).$(MINOR)

all: lib$(NAME).so
lib$(NAME).so.$(VERSION): $(OUTS)
    $(CC) $(LDFLAGS) $(OUTS) -shared -Wl,-soname,lib$(NAME).so.$(MAJOR) -o lib$
(NAME).so.$(VERSION)

lib$(NAME).so: lib$(NAME).so.$(VERSION)
    rm -f lib$(NAME).so.$(MAJOR) lib$(NAME).so
    ln -s lib$(NAME).so.$(VERSION) lib$(NAME).so.$(MAJOR)
    ln -s lib$(NAME).so.$(MAJOR) lib$(NAME).so

%.o: %.c
    $(CC) $(CFLAGS) -c -fPIC $(LIBSOURCES)

clean:
    rm -rf *.o *.so *.so.*
```

4. Build the library recipe

```
$ petalinux-build -c libsample
```

14.1 Creating Apps(which uses libraries) in PetaLinux Project

1. Create a library using PetaLinux tools

```
$ petlainux-create -t apps --template c --name sampleapp --enable
```

2. Modify the *sampleapp.bb* recipe file *<plnx-proj-root>/project-spec/meta-user/recipes-apps/sampleapp/sampleapp.bb* as
For 2022.1 PetaLinux release onwards:

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

SUMMARY = "Simple sampleapp application"
SECTION = "PETALINUX/apps"
LICENSE = "MIT"
```

```
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/
MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

FILESEXTRAPATHS:prepend := "${THISDIR}/files:"

SRC_URI = " \
          file://sampleapp.c \
         "

S = "${WORKDIR}"

DEPENDS += " libsample"

#This uses libsample.so.1.0.1 (-lsample)
do_compile() {
    ${CC} ${CFLAGS} ${LDFLAGS} -o sampleapp sampleapp.c -lsample
}

do_install() {
    install -d ${D}${bindir}
    install -d -m 0755 sampleapp ${D}${bindir}
}

FILES:${PN} += "sampleapp"
```

For 2021.2 and Below PetaLinux releases:

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

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

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

SRC_URI = " \
          file://sampleapp.c \
         "

S = "${WORKDIR}"

DEPENDS += " libsample"

#This uses libsample.so.1.0.1 (-lsample)
do_compile() {
    ${CC} ${CFLAGS} ${LDFLAGS} -o sampleapp sampleapp.c -lsample
}
```

```
do_install() {
    install -d ${D}${bindir}
    install -d -m 0755 sampleapp ${D}${bindir}
}

FILES_${PN} += "sampleapp"
```

3. Modify the *<plnx-proj-root>/project-spec/meta-user/recipes-apps/libsample/files/sampleapp.c* generated from PetaLinux tools as

```
#include <stdio.h>
#include <SAMPLE/sample.h>  // From above libs <TARGET_ROOTFS>/usr/lib/SAMPLE/
sample.h

int main(int argc, char **argv)
{
    printf("Hello World!\n");
    sample_hello();
    return 0;
}
```

4. Build the apps recipe

```
$ petalinux-build -c sampleapp
```

15 How to Auto Run Application at Startup

1. Create a library using PetaLinux tools

```
$ petalinux-create -t apps --template install -n myapp-init --enable
```

2. Create the file **<plnx-proj-root>/project-spec/meta-user/recipes-apps/myapp-init/files/myapp-init.service.**[5]

```
[Unit]
Description=myapp-init

[Service]
ExecStart=/usr/bin/myapp-init
StandardOutput=journal+console

[Install]
WantedBy=multi-user.target
```

3. Edit the file **<plnx-proj-root>/project-spec/meta-user/recipes-apps/myapp-init/myappinit.bb**. The file should look like the following:

For 2022.1 PetaLinux release onwards:

```
#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 \
        file://myapp-init.service \
"

S = "${WORKDIR}"

FILESEXTRAPATHS:prepend := "${THISDIR}/files:"

inherit update-rc.d systemd

INITSCRIPT_NAME = "myapp-init"
```

5 http://myappinit.bb/

```
INITSCRIPT_PARAMS = "start 99 S ."

SYSTEMD_PACKAGES = "${PN}"
SYSTEMD_SERVICE:${PN} = "myapp-init.service"
SYSTEMD_AUTO_ENABLE:${PN} = "enable"

do_install() {
        if ${@bb.utils.contains('DISTRO_FEATURES', 'sysvinit', 'true', 'false', d)};
then
                install -d ${D}${sysconfdir}/init.d/
                install -m 0755 ${WORKDIR}/myapp-init ${D}${sysconfdir}/init.d/
        fi

        install -d ${D}${bindir}
        install -m 0755 ${WORKDIR}/myapp-init ${D}${bindir}/
        install -d ${D}${systemd_system_unitdir}
        install -m 0644 ${WORKDIR}/myapp-init.service ${D}${systemd_system_unitdir}
}

FILES:${PN} += "${@bb.utils.contains('DISTRO_FEATURES','sysvinit','${sysconfdir}/*',
'', d)}"
```

For 2021.2 and Below PetaLinux releases:

```
#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}/*"
```

4. To run myapp as daemon, edit the file **<plnx-proj-root>/project-spec/meta-user/recipes-apps/myapp-init/files/myapp-init**.

> ⚠ Below step required only if you are using SYSTEMV as INITMANAGER.

```
#!/bin/sh
DAEMON=/usr/bin/myapp-init
start ()
{
        echo " Starting myapp-init"
        start-stop-daemon -S -o --background -x $DAEMON
}
stop ()
{
        echo " Stoping myapp-init"
        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. Build the project

```
petalinux-build
```

16 How to Auto Mount SD card in Yocto Recipes

1. Create a base-files directory in meta-user layer as

```
$ mkdir -p <plnx-proj-root>/project-spec/meta-user/recipes-core/base-files
```

2. Create a base-files_%.bbappend file and add below content

```
$ vim <plnx-proj-root>/project-spec/meta-user/recipes-core/base-files/base-
files_%.bbappend

#base-files_%.bbappend content

dirs755 += "/media/card"

do_install_append() {
    sed -i '/mmcblk0p1/s/^#//g' ${D}${sysconfdir}/fstab
}
```

3. Rebuild rootfs

```
$ petalinux-build
```

17 How to Configure a Second Ethernet Interface(eth1) to Get the IP Address from DHCP in Yocto Recipes/ PetaLinux using SysVinit

17.1 Method 1: Append auto eth1 to existing interfaces file

1. Create a init-ifupdown directory in meta-user layer as

```
$ mkdir -p <plnx-proj-root>/project-spec/meta-user/recipes-core/init-ifupdown/
```

2. Create a *init_ifupdown_%.bbappend* file and add below content

```
$ vim <plnx-proj-root>/project-spec/meta-user/recipes-core/init-ifupdown/init-
ifupdown_%.bbappend

# init-ifupdown_%.bbappend content
#For 2022.1 release onwards
do_install:append() {
    sed -i '/iface eth0 inet dhcp/ a auto eth1' ${D}${sysconfdir}/network/interfaces
}
#For 2021.2 and old releases
do_install_append() {
    sed -i '/iface eth0 inet dhcp/ a auto eth1' ${D}${sysconfdir}/network/interfaces
}
```

3. Rebuild rootfs

```
$ petalinux-build
```

17.2 Method 2: Use your own interfaces file

1. Create a init-ifupdown directory in meta-user layer as

```
$ mkdir -p <plnx-proj-root>/project-spec/meta-user/recipes-core/init-ifupdown/files
```

2. Create your own network interface file as show below

```
$ vim <plnx-proj-root>/project-spec/meta-user/recipes-core/init-ifupdown/files/
myinterfaces

# myinterfaces content

# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)

# The loopback interface
auto lo
iface lo inet loopback

# Wireless interfaces
iface wlan0 inet dhcp
    wireless_mode managed
    wireless_essid any
    wpa-driver wext
    wpa-conf /etc/wpa_supplicant.conf

iface atml0 inet dhcp

# Wired or wireless interfaces
auto eth0
iface eth0 inet dhcp

# Add auto config for eth1
auto eth1
iface eth1 inet dhcp

# Ethernet/RNDIS gadget (g_ether)
# ... or on host side, usbnet and random hwaddr
iface usb0 inet static
    address 192.168.7.2
    netmask 255.255.255.0
    network 192.168.7.0
    gateway 192.168.7.1

# Bluetooth networking
iface bnep0 inet dhcp
```

3. Create a init_ifupdown_%.bbappend file and add below content

```
$ vim <plnx-proj-root>/project-spec/meta-user/recipes-core/init-ifupdown/init-
ifupdown_%.bbappend

# init-ifupdown_%.bbappend content

SRC_URI += " \
        file://myinterfaces \
        "
# For 2022.1 release onwards
FILESEXTRAPATHS:prepend := "${THISDIR}/files:"
```

```
# For 2021.2 and old releases
FILESEXTRAPATHS_prepend := "${THISDIR}/files:"

# Overwrite interface file with myinterface file in rootfs
# For 2022.1 release onwards
do_install:append() {
     install -m 0644 ${WORKDIR}/myinterfaces ${D}${sysconfdir}/network/interfaces
}

#For 2021.2 and old releases
do_install_append() {
     install -m 0644 ${WORKDIR}/myinterfaces ${D}${sysconfdir}/network/interfaces
}
```

4. Rebuild rootfs

```
$ petalinux-build
```

18 How to Run RootFS Post Process Command in PetaLinux

Below is one of the example and you can add rootfs post process command in *<plnx-proj-root>/project-spec/meta-user/recipes-core/images/petalinuxbsp-user-image.bbappend* file. You add a function which can perform modifying file, deleting and adding file and directories.

1. Create a **petalinux-user-image.bbappend** file in meta-user layer as show below if doesn't exists(Note: Don't use project-spec/meta-user/recipes-core/images/petalinux-image-full.bbappend(2018.3 or later) or petalinux-image.bbappend(2018.2 or below))

```
$ vim <plnx-proj-root>/project-spec/meta-user/recipes-core/images/petalinux-user-image.bbappend
```

2. Create a post process function as shown below in **petalinux-user-image.bbappend**

petalinux-user-image.bbappend

```
# petalinux-user-image.bbappend content

# Auto DHCP for second network interface
rootfs_postprocess_function() {
    if [ -e ${IMAGE_ROOTFS}/etc/network/interfaces ]; then
        sed -i '/iface eth0 inet dhcp/ a auto eth1' ${IMAGE_ROOTFS}/etc/network/interfaces
    fi
}

# For 2022.1 PetaLinux release onwards:
ROOTFS_POSTPROCESS_COMMAND:append = " \
  rootfs_postprocess_function; \
"
For 2021.2 and Below PetaLinux releases:
ROOTFS_POSTPROCESS_COMMAND_append = " \
  rootfs_postprocess_function; \
"
```

3. Clean petalinux-user-image sstate and rebuild petalinux-user-image recipe.

```
$ petalinux-build -x cleansstate
$ petalinux-build
```

19 How to Add Users and Set Passwords for Users in PetaLinux

1. Create a **petalinux-user-image.bbappend** file in meta-user layer as show below (Note: Don't use project-spec/meta-user/recipes-core/images/petalinux-image-full.bbappend(2018.3 or later) or petalinux-image.bbappend(2018.2 or below))

```
$ vim <plnx-proj-root>/project-spec/meta-user/recipes-core/images/petalinux-user-image.bbappend
```

2. Inherit extrausers.bbclass from openembedded-core and add users in **petalinux-user-image.bbappend**

petalinux-user-image.bbappend

```
# petalinux-user-image.bbappend content
inherit extrausers

EXTRA_USERS_PARAMS = "\
    usermod -P root root; \
    useradd -P test@123 test1; \
    useradd -P test@987 test2; \
    "
```

3. Rebuild rootfs

```
$ petalinux-build -x cleanall
$ petalinux-build
```

20 How to Override a Recipe Variable in PetaLinux

1. First step is to find the recipe directory hierarchy, search the recipe in https://layers.openembedded.org/layerindex/branch/rocko/recipes/
2. Here we are taking Protobuf package as example as SRC_URI like is broken in Yocto 2.4 release. Same concept applies to all the recipes. Protobuf is categorized in meta-openembedded/meta-oe/recipes-devtools/protobuf
3. Create a directory recipes-devtools/protobuf directory in <plnx-porj-root>/project-spec/meta-user

```
$ mkdir -p <plnx-proj-root>/project-spec/meta-user/recipes-devtools/protobuf
```

4. Create a protobuf_%.bbappend file. Note: The % wildcard is actually allow matching of the name and version up to the point of encountering the %. This approach will allow for matching of the major or major.minor. version of recipes.

```
$ vim <plnx-proj-root>/project-spec/meta-user/recipes-devtools/protobuf/
protobuf_%.bbappend
```

5. Now override a recipe variable for protobuf recipe.

protobuf_%.bbappend

```
# protobuf_%.bbappend content
SRC_URI = "git://github.com/google/protobuf.git"
```

6. Build protobuf

```
$ petalinux-build -c protobuf -x cleanall
$ petalinux-build -c protobuf
```

21 How to Enable Verbose in sysvinit

1. First step is to find the recipe directory hierarchy, search the recipe in http://layers.openembedded.org/layerindex/branch/thud/recipes/
2. sysvinit is categorized in meta/recipes-core/sysvinit
3. Create a directory recipes-core/sysvinit directory in <plnx-porj-root>/project-spec/meta-user

```
$ mkdir -p <plnx-proj-root>/project-spec/meta-user/recipes-core/sysvinit
```

4. Create a sysvinit_%.bbappend file. Note: The % wildcard is actually allow matching of the name and version up to the point of encountering the %. This approach will allow for matching of the major or major.minor. version of recipes.

```
$ vim <plnx-proj-root>/project-spec/meta-user/recipes-core/sysvinit/sysvinit_%.
bbappend
```

5. Now enable verbose for sysvinit recipe (refer http://cgit.openembedded.org/openembedded-core/tree/meta/recipes-core/sysvinit/sysvinit/rcS-default?h=thud#n15)

sysvinit_%.bbappend

```
# sysvinit_%.bbappend content
# For 2022.1 PetaLinux release onwards:
do_install:prepend () {
        sed -i 's/VERBOSE=no/VERBOSE=yes/' ${WORKDIR}/rcS-default
}
# For 2021.2 and Below PetaLinux releases:
do_install_prepend () {
        sed -i 's/VERBOSE=no/VERBOSE=yes/' ${WORKDIR}/rcS-default
}
```

6. Clean and rebuild sysvinit

```
$ petalinux-build -c sysvinit -x cleanall
$ petalinux-build -c sysvinit
```

22 How to reduce build time using SSTATE CACHE

The OpenEmbedded(**OE**) build system produces a lot of intermediate output when processing the many tasks entailed in building the packages comprising the Linux OS stack. **SSTATE**(Shared State) **CACHE** provides a cache mechanism which drastically reduces build time, especially when you create a new PetaLinux project and run *"petalinux-build --sdk"*. The OE build system detects changes in the "inputs" to a given task by creating a checksum (or signature) of the task's inputs. If the checksum changes, the system assumes the inputs have changed and the task needs to be rerun. The intermediate output can be reused for future builds and shared between multiple build environments to speed up the build process.

Two key variables are used with the shared state cache feature. The **SSTATE_DIR** variable contains the path to the shared state cache. The variable **DL_DIR** tells BitBake where to place the source downloads. The default setting places the files in the directory *downloads* beneath the top directory(**TOPDIR**) of your build environment. The variable **TOPDIR** contains the full (absolute) path to the build environment. Source downloads can be shared among multiple build environments.

For best practice you need to set DL_DIR and SSTATE_DIR outside the petalinux project so that when you run *"petalinux-build -s mrproper"* which removes build(TOPDIR)directory forcefully, but retaining DL_DIR and SSTATE_DIR directories. These contents can be reused for incremental build or building another project. Note: You can use DL_DIR content for all the PetaLinux project irrespective of the architecture(**AARCH64**, **ARM, MicroBlaze**) but SSTATE_DIR can be used for only one architecture. For example, if you build a AARCH64 PetaLinux project the content of SSTATE_DIR can be reused for another AARCH64 project but not for ARM or MicroBlaze projects.

In order to utilize the sstate cache feature in PetaLinux projects follow these steps.

1. Download and untar the PetaLinux release sstate-cache files from https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools.html. Here we are taking 2019.2 as an example.
2. Go to your 2019.2 PetaLinux project and set the following options using petalinux-config

SSTATE Template

```
$ petalinux-config ---> Yocto Settings ---> Add pre-mirror url ---> file://
<path_to_downloads>
$ petalinux-config ---> Yocto Settings ---> Local sstate feeds settings --->
local sstate feeds url ---> <path_to_sstate_aarch64_2019.2/aarch64>
```

SSTATE Example Usage

```
$ petalinux-config ---> Yocto Settings ---> Add pre-mirror url ---> file:///
opt/xilinx/petalinux/2019.2/downloads
```

```
$ petalinux-config ---> Yocto Settings ---> Local sstate feeds settings --->
local sstate feeds url ---> /opt/xilinx/petalinux/2019.2/sstate_aarch64_2019.2/
aarch64
```

Note: For PREMIRRORS usage you need to add the *file, git, http or https* protocol followed by the path, but this is not true for SSTATE feeds.

3. For setting up the download and shared state directories add the following variables in the *<plnx-proj-root>/project-spec/meta-user/conf/petalinuxbsp.conf* file.

DL_DIR and SSTATE_DIR Template

```
DL_DIR = "<PATH-TO-DOWNLOADS-DIRECTORY>/<PetaLinux-Version>/downloads"
SSTATE_DIR = "<PATH-TO-SSTATE-DIRECTORY>/<PetaLinux-Version>/sstate-cache/
<ARCH>"
```

DL_DIR and SSTATE_DIR example usage

```
DL_DIR = "/home/$USER/plnx-workspace/2019.2/downloads"
SSTATE_DIR = "/home/$USER/plnx-workspace/2019.2/sstate-cache/aarch64"
```

Note: DL_DIR and SSTATE_DIR directories should have read and write permission.

4. Now you need to start the build and if you are building the SDK (*petalinux --sdk*) then enabling the network is mandatory for a fresh build.

Build Commands

```
$ petalinux-build
$ petalinux-build --sdk
```

5. If you are building without network access then set the BB_NO_NETWORK variable.

Set BB_NO_NETWORK

```
$ petalinux-config ---> Yocto Settings ---> [*] Enable BB NO NETWORK
```

Note: In case the project was built previously, run *petalinux-build -x mrproper* with above variables to reduce build time especially when using *petalinux-build --sdk*.

23 How to disable hwcodecs(Xilinx vcu firmware) in PetaLinux for ZynqMP EV devices

1. By default hwcodecs are enabled in petalinux project when zynqmp ev device xsa is parsed while creating the project. To disable hwcodecs add below variable in *<plnx-proj-root>/project-spec/meta-user/conf/petaliuxbsp.conf*

```
# For 2022.1 PetaLinux release onwards:
IMAGE_FEATURES:remove = "hwcodecs"
# For 2021.2 and Below PetaLinux releases:
IMAGE_FEATURES_remove = "hwcodecs"
```

2. Clean and rebuild complete project.

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

24 Yocto References

https://wiki.yoctoproject.org/wiki/Technical_FAQ
https://wiki.yoctoproject.org/wiki/Building_your_own_recipes_from_first_principles
https://www.yoctoproject.org/docs/current/dev-manual/dev-manual.html#new-recipe-writing-a-new-recipe
https://www.openembedded.org/Layers_FAQ
https://www.yoctoproject.org/docs/current/bitbake-user-manual/bitbake-user-manual.html

Related Links

Xilinx Yocto[6]

6 https://xilinx-wiki.atlassian.net/wiki/display/A/Yocto