Incorporating Out-of-Tree Modules in YOCTO

Incorporating Out-of-Tree Modules in YOCTO

Sometimes an external *Linux Kernel Module* is needed. This document describes the steps to create your own *out-of-tree* kernel module recipe for Yocto.

In order to do this, the document will guide you through the process by adding a Linux kernel module called *pmu_user.ko* to the *avs-image* for the i.MX 8M boards. These steps should apply for similar incorporation of any kernel to Yocto images.

On the Host

Create meta layer using out-of-tree module template

Go to the directory where you have your Yocto build environment ready and create a new meta layer in Yocto:

```
cd ../imx-yocto-bsp/<build dir>
#Create a new meta layer and give the directory a descriptive name
bitbake-layers create-layer ../layers/meta-pmu
#Go to layer directory
cd ../layers/meta-pmu
```

There should be already a layer example created by default. We will remove the *recipes-example* directory and create a new one with the corresponding name of the kernel module we are incorporating:

```
.
├── conf
│   └── layer.conf
├── COPYING.MIT
├── README
└── recipes-example
    └── example
        └── example_0.1.bb
3 directories, 4 files
```

```
rm -rf recipes-example/
mkdir recipes-pmu
```

The poky git repository of Yocto Project provides a template for *out-of-tree* modules. The easiest way to incorporate a kernel module for the first time is by using this resource.

Let's go to the created directory and copy the *hello-mod.bb* recipe available at *poky/meta-skeleton/recipes-kernel/hello-mod/*. This is the template from which you can create your *own out-of-tree* Linux Kernel module recipe.

```
cd recipes-pmu
# Copy the template to the recipes-pmu directory
cp -fvR ../../../sources/poky/meta-skeleton/recipes-kernel/hello-mod .
```

Now you should see the following directories and files inside *recipes-pmu*:

```
└── hello-mod
    ├── files
    │   ├── COPYING
    │   ├── hello.c
    │   └── Makefile
    └── hello-mod_0.1.bb
2 directories, 4 files
```

You can store any source files, patches, or other files as necessary for building the module in the files directory. For this example, no additional files were needed.

Before anything else, we should change the name of *hello-mod* directory to *pmu-mod* and the name of *hello-mod_0.1.bb* to *pmu-mod_0.1.bb*:

```
mv hello-mod/ pmu-mod/
cd pmu-mod
mv hello-mod_0.1.bb pmu-mod_0.1.bb
```

Now, let's see what's inside the *pmu-mod_0.1.bb* file:

```
SUMMARY = "Example of how to build an external Linux kernel module"
DESCRIPTION = "${SUMMARY}"
LICENSE = "GPLv2"
LIC_FILES_CHKSUM = "file://COPYING;md5=12f884d2ae1ff87c09e5b7ccc2c4ca7e"

inherit module

SRC_URI = "file://Makefile \
           file://hello.c \
           file://COPYING \
           "

S = "${WORKDIR}"

# The inherit of module.bbclass will automatically name module packages with
# "kernel-module-" prefix as required by the oe-core build environment.

RPROVIDES_${PN} += "kernel-module-hello"
```

The most important line is the **inherit module**, which tells the *bitbake* command how to properly build the kernel module. Notice that there is no *do_compile* or *do_install* needed.

Create git repository for kernel module and edit files

For best practices, create a new git repository for the kernel module and move the *makefile* and *COPYING* files to it. Also, add the necessary source files for the kernel module. For the *pmu_user.ko* we need to include the *pmu_user.c* file.

Finally, let's edit the *makefile* so it compiles the *pmu_user* kernel module:

```
obj-m := pmu_user.o

SRC := $(shell pwd)

all:
    $(MAKE) -C $(KERNEL_SRC) M=$(SRC)

modules_install:
    $(MAKE) -C $(KERNEL_SRC) M=$(SRC) modules_install

clean:
    rm -f *.o *~ core .depend .*.cmd *.ko *.mod.c
    rm -f Module.markers Module.symvers modules.order
    rm -rf .tmp_versions Modules.symvers
```

After that, the *pmu-mod_0.1.bb* needs to be updated like this:

```
SUMMARY = "pmu-user-module"
DESCRIPTION = "Enables user mode access to PMU registers"
LICENSE = "CLOSED"
LIC_FILES_CHKSUM = ""
SRCBRANCH = "master"

inherit module

SRC_URI = "git://git@bitbucket.sw.nxp.com/mag/pmu-user-module.git;protocol=ssh;
branch=${SRCBRANCH}"
SRCREV = "${AUTOREV}"

S = "${WORKDIR}/git"

# The inherit of module.bbclass will automatically name module packages with
# "kernel-module-" prefix as required by the oe-core build environment.

RPROVIDES_${PN} += "kernel-module-pmu-user"
```

NOTE: *LIC_FILES_CHKSUM* and *LICENSE* must be provided, but let's keep it simple here.

Add kernel module layer

Now it's time to add the created layer to the image. Let's move to the build directory:

```
#Go to your build directory
cd <imx-yocto-bsp>/<build dir>
#Add layer to bblayers.conf
bitbake-layers add-layer ../layers/meta-pmu/
```

One last step: add the recipe at the end of the *local.conf* file:

```
vi conf/local.conf
```

```
#Add this line at the end of local.conf
IMAGE_INSTALL_append = " pmu-mod"
```

Now we can generate the image:

```
bitbake <name of image>
```

Note: This is only a brief document that explains how to incorporate an *out-of-tree* kernel module to the yocto image. However, it would be best to put the *meta-pmu* directory inside the *sources/* directory and follow Yocto's best practices.

On the target (i.MX 8M Plus)

Once the image is ready and flashed to the board, we should be able to find the kernel module inside of */lib/modules/<your-release-version>/extra* :

```
cd /lib/modules/5.10.52-lts-5.10.y+ga11753a89ec6/extra
```

We can load the kernel module using *modprobe*:

```
modprobe pmu_user
```

To check that it is currently loaded, we use *lsmode*:

i.MX 8 Family | i.MX 8QuadMax (8QM) | 8QuadPlus Linux Yocto Project

8m

8mp 8MQ Kernel modules out of tree yocto

Version history

Revision #: 4 of 4
Last update: 11-18-2021 02:55 PM
Updated by: alberto_alvarez

View Article History