Moxa UPort Driver Arm-based Platform Porting Guide

Moxa Technical Support Team

support@moxa.com

Contents

 Released on Oct 21, 2021

About Moxa

Moxa is a leading provider of edge connectivity, industrial computing, and network infrastructure solutions for enabling connectivity for the Industrial Internet of Things (IIoT). With over 30 years of industry experience, Moxa has connected more than 71 million devices worldwide and has a distribution and service network that reaches customers in more than 80 countries. Moxa delivers lasting business value by empowering industries with reliable networks and sincere service. Information about Moxa's solutions is available at www.moxa.com.

How to Contact Moxa

Tel: 1-714-528-6777
Fax: 1-714-528-6778

1 For Linux Kernel 4.x

1.1 Introduction

This document details porting the Moxa UPort driver to a specified Arm-based platform. The following knowledge is recommended before reading the instructions in this guide.

- Linux Kernel Programming
- ARM Platform Compiler
- Yocto Project Document
- MOXA UC-Series Manual
- Raspberry Pi Manual

Instructions in this guide use examples of porting on the Moxa UC-Series Arm platform and Raspberry Pi. You can apply the experience of porting UPort driver to other platforms.

The UPort driver fully supports all modern-day Linux distributions running on x86 environments, and the driver core is also compatible with the Arm platform. This document will guide you on how to port the UPort driver core.

However, some platform-dependent services, such as installer, are not available. You may refer to the platform's documentation to fulfill the requirements.

1.2 Porting to Moxa UC-Series - Arm-based Computer

1.2.1 Build binaries on a general ARM platform

If your platform is powerful and consists of the necessary development tools, the driver can be built on the platform directly. You can refer to readme.txt of UPort driver to understand the requirement.

The step of building this driver in an ARM environment is the same as in x86 and x64 environments.

```
# tar zxvf [DRIVER NAME].tgz
# cd mxu11x0
# ./mxinstall
```

1.2.2 Cross-compiler and the UPort driver

Note To cross-compiler on a x86 or x64 Linux host, the target ARM environment's kernel source package and cross compiler toolchain must be installed first.

After installing and configuring the kernel source package and toolchain, you need to compile the source code with the kernel source package and toolchain.

In this example, we install the cross-compiler for the Moxa UC-Series ARM-based computer. You can refer to the product's manual for further detail.

1. Download the kernel source package webpage under the product page.

   ```
   $ git clone https://github.com/Moxa-Linux/am335x-linux-4.4
   ```

   You can use the following commands to show the git tag list and check out the tag of the specific UC device and firmware version.

   Show the tag list:

   ```
   $ git tag
   ```

   Check out the specific tag:

   ```
   $ git checkout UC-2100_V1.7 ← Replace the tag name, UC-2100_V1.7,
                                 with the UC Series that is being used.
   ```

2. Download the toolchain from the product's webpage. The toolchain, which is used by the UC Series, is arm-linux-gnueabihf. It is a script that will install the related packages. Execute the script and follow the steps to install the Linux cross-compiler tools. You will need the root privilege to install the toolchain and the kernel source.

   ```
   # sh arm-linux-gnueabihf_6.3_Build_amd64_<build_date>.sh
   ```

   If the script shows the notification message: "Please export these environment variables before using the toolchain", enter the following script command:

   ```
   # export PATH=$PATH:/usr/local/arm-linux-gnueabihf-6.3/usr/bin
   ```

3. The kernel source, which is used by the UC Series, is am335x-linux-4.4. You need to configure these files before starting to cross-compile.

   Move the kernel source to /moxa/kernel and configure the kernel source.

   ```
   # mv am335x-linux-4.4 /moxa/kernel
   # cd /moxa/kernel
   # make uc2100_defconfig ← Replace the uc2100 with the UC Series
                             that is being used.
   # make modules_prepare
   ```

   After the abovementioned steps, please follow the processes as set out in Section 2.3, “Moxa cross-compiling interactive script,” and Section 2.4, “Manually build the UPort driver with a cross-compiler,” to cross-compile Moxa's driver for the UC-Series platforms.

   The UPort driver, which includes the driver modules, needs to be compiled. The file of each UPort Series is listed as follow:

   UPort 1100 Series driver

   - mxu11x0.ko

   UPort 1200/1400/1600 Series driver

   - mxu11x0.ko
   - mxusbserial.ko

   UPort 2210/2410 Series driver

   - mxuport2000.ko

If it is preferred to build these binaries with automatic script, please refer to Section 2.3, "Moxa cross-compiling interactive script." If you find the build script troublesome, or you prefer to build these binaries manually, please refer to section 2.4 "Manually build the UPort driver with a cross-compiler."

If you have generated the necessary binaries, please refer to section 2.5 to deploy to the target platform.

1.2.3 Moxa cross-compiling interactive script

To simplify the processes above, Moxa has provided an interactive script, "mxcc", to cross-compile these drivers. You may execute ./mxcc in the UPort driver source directory to cross-compile the MOXA driver.

For the UPort 1100 Series

The steps are as follows:

```
#./mxcc
Enter target device architecture (ARCH) [arm]:
Enter cross-compile (CROSS_COMPILE) [arm-linux-gnueabihf-]:
Enter target device kernel source directory [/moxa/kernel]:
****************************************
mxu11x0 cross-compile is success.
****************************************

************************************************************************
MOXA UPort 1100 Series driver cross-compile finished.
If cross compile is success, driver is outputted to the output folder.
************************************************************************
```

The binaries will now be generated and placed in the output directory under the /moxa/mxu11x0 folder.

For the UPort 1200/1400/1600 Series

The steps are as follows:

```
# ./mxcc
Enter target device architecture (ARCH) [arm]:
Enter cross-compile (CROSS_COMPILE) [arm-linux-gnueabihf-]:
Enter target device kernel source directory [/moxa/kernel/]:
****************************************
mxuport cross-compile is success.
****************************************
****************************************
mxusbserial cross-compile is success.
****************************************
************************************************************************
MOXA UPort 1200/1400/1600 Series driver cross-compile finished.
If cross compile is success, driver is outputted to the output folder.
************************************************************************
```

The binaries will now be generated and placed in the output directory under the /moxa/mxuport folder.

For the UPort 2210/2410 Series

The steps are as follows:

```
Enter target device architecture (ARCH) [arm]:
Enter cross-compile (CROSS_COMPILE) [arm-linux-gnueabihf-]:
Enter target device kernel source directory [/moxa/kernel/]:
****************************************
mxuport2000 cross-compile is success.
****************************************
**********************************************************************
MOXA UPort 2210/2410 Series driver cross-compile finished.
If cross compile is success, driver is outputted to the output folder.
**********************************************************************
```

The binaries will now be generated and placed in the output directory under the /moxa/mxuport2000 folder.

1.2.4 Manually build the UPort driver with a cross-compiler

For the UPort 1100 Series

To cross-compile the UPort 1100 Series driver, you can find "Makefile" in the driver folder, and then run it.

```
# make ARCH=<ARCH> CROSS_COMPILE=<CROSS_COMPILE> KDIR=<KERNEL_SOURCE>
KVER_MAJOR=<KERNEL_MAJOR> KVER_MINOR=<KERNEL_MINOR>
```

<ARCH>: The target ARM environment device's CPU architecture. For example, arm, arm64.

<CROSS_COMPILE>: The cross-compile toolchain path. If the toolchain is arm-linux-gnueabihf, and you have already added the path of the toolchain to the environment variable, you should enter "arm-linux-gnueabihf-" here.

<KERNEL_SOURCE>: The directory of the target kernel source.

<KERNEL_MAJOR>: The major version of the target ARM system's kernel source. You can use the command "make kernelversion" to get the kernel source's major version.

e.g.,

```
# make kernelversion
4.4.0
|
+--- kernel major version
```

<KERNEL_MINOR>: The minor version of the target ARM system's kernel source. You can use the command "make kernelversion" to get the kernel source's minor version.

e.g.,

```
# make kernelversion
4.4.0
  |
  +--- kernel minor version
```

The "make" command would be similar to the following example:

```
#make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- KDIR=/moxa/kernel
KVER_MAJOR=4 KVER_MINOR=4
```

After using the "make" command to cross-compile the drivers, the driver file "mxu11x0.ko" can be found in the /moxa/ mxu11x0/driver directory.

For the UPort 1200/1400/1600 Series

To cross-compile UPort 1200/1400/1600 Series driver, there are two drivers, mxuport.ko and mxusbserial.ko. You can find "Makefile" in the /moxa/mxuport/driver/mxuport and /moxa/mxuport/driver/mxusbserial folder.

Build mxuport.ko.

Run the Makefile in the /moxa/mxuport/driver/mxuport folder.

```
# make ARCH=<ARCH> CROSS_COMPILE=<CROSS_COMPILE> KDIR=<KERNEL_SOURCE>
KVER_MAJOR=<KERNEL_MAJOR> KVER_MINOR=<KERNEL_MINOR>
```

<ARCH>: The target ARM environment device's CPU architecture. For example, arm, arm64.

<CROSS_COMPILE>: The cross-compile toolchain path. If the toolchain is arm-linux-gnueabihf, and you have already added the path of the toolchain to the environment variable, you should enter "arm-linux-gnueabihf-" here.

<KERNEL_SOURCE>: The directory of the target kernel source.

<KERNEL_MAJOR>: The major version of the target ARM system's kernel source. You can use the command "make kernelversion" to get the kernel source's major version.

e.g.,

```
# make kernelversion
4.4.0
|
+--- kernel major version
```

<KERNEL_MINOR>: The minor version of the target ARM system's kernel source. You can use the command "make kernelversion" to get the kernel source's minor version.

e.g.,

```
# make kernelversion
4.4.0
  |
  +--- kernel minor version
```

The "make" command would be similar to the following example:

```
# make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- KDIR=/moxa/kernel
KVER_MAJOR=4 KVER_MINOR=4
```

After using the "make" command to cross-compile the drivers, the driver file "mxuport.ko" can be found in the /moxa/mxuport/driver/mxuport directory.

Build mxusbserial.ko in the /moxa/mxuport/driver/mxusbserial folder.

Run the Makefile

```
# make ARCH=<ARCH> CROSS_COMPILE=<CROSS_COMPILE> KDIR=<KERNEL_SOURCE>
KVER_MAJOR=<KERNEL_MAJOR> KVER_MINOR=<KERNEL_MINOR>
```

<ARCH>: The target ARM environment device's CPU architecture. For example, arm, arm64.

<CROSS_COMPILE>: The cross-compile toolchain path. If the toolchain is arm-linux-gnueabihf, and you have already added the path of the toolchain to the environment variable, you should enter "arm-linux-gnueabihf-" here.

<KERNEL_SOURCE>: The directory of the target kernel source.

<KERNEL_MAJOR>: The major version of the target ARM system's kernel source. You can use the command "make kernelversion" to get the kernel source's major version.

e.g.,

```
# make kernelversion
4.4.0
|
+--- kernel major version
```

<KERNEL_MINOR>: The minor version of the target ARM system's kernel source. You can use the command "make kernelversion" to get the kernel source's minor version.

e.g.,

```
# make kernelversion
4.4.0
  |
  +--- kernel minor version
```

The "make" command would be similar to the following example:

```
# make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- KDIR=/moxa/kernel
KVER_MAJOR=4 KVER_MINOR=4
```

After using the "make" command to cross-compile the drivers, the driver file "mxusbserial.ko" can be found in the /moxa/mxuport/driver/mxusbserial directory.

For the UPort 2210/2410 Series

To cross-compile UPort 2210/2410 Series driver, you can find "Makefile" in the driver folder, then run it.

```
# make ARCH=<ARCH> CROSS_COMPILE=<CROSS_COMPILE> KDIR=<KERNEL_SOURCE>
KVER_MAJOR=<KERNEL_MAJOR> KVER_MINOR=<KERNEL_MINOR>
```

<ARCH>: The target ARM environment device's CPU architecture. For example, arm, arm64.

<CROSS_COMPILE>: The cross-compile toolchain path. If the toolchain is arm-linux-gnueabihf, and you have already added the path of the toolchain to the environment variable, you should enter "arm-linux-gnueabihf-" here.

<KERNEL_SOURCE>: The directory of the target kernel source.

<KERNEL_MAJOR>: The major version of the target ARM system's kernel source. You can use the command "make kernelversion" to get the kernel source's major version.

e.g.,

```
# make kernelversion
4.4.0
|
+--- kernel major version
```

<KERNEL_MINOR>: The minor version of the target ARM system's kernel source. You can use the command "make kernelversion" to get the kernel source's minor version.

e.g.,

```
# make kernelversion
4.4.0
  |
  +--- kernel minor version
```

The "make" command would be similar to the following example:

```
# make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- /moxa/kernel KVER_MAJOR=4
KVER_MINOR=4
```

After using the "make" command to cross-compile the drivers, the driver file "mxuport2000.ko" can be found in the /moxa/mxuport2000/driver directory.

To cross-compile the UPort 2210/2410 Series driver's utility, you can follow the following steps:

You can find the source in the utility folder.

```
# cd /moxa/mxuport2000/utility/conftool/
```

Use the cross-compiler to compile the utility.

```
# <CROSS_COMPILE>gcc -o mxustty conftool.c
```

After using the cross compiler to compile the utility, you can find the utility execution 'mxustty' in /moxa/mxuport2000/utility/conftool/

1.2.5 Deploy cross-compiled binary to target

For the UPort 1100 Series

You should find the kernel module, mxu11x0.ko, under the output or driver source code directory.

Follow the steps below to deploy to the target Arm platform.

1. Copy the mxu11x0.ko to the path

   /lib/modules/`uname -r`/kernel/drivers/char on the ARM platform.

2. Boot into the ARM platform and load the driver.

   ```
   # modprobe mxu11x0
   ```

UPort 1200/1400/1600 Series

You should find following binaries under the output or driver source code directory.

mxuport.ko
mxusbserial.ko

Follow the steps below to deploy to the target Arm platform.

1. Copy the mxuport.ko to the path

   /lib/modules/`uname -r`/kernel/drivers/char on the ARM platform

2. Boot into the ARM platform and load the driver.

```
# modprobe mxuport
# modprobe mxusbserial
```

UPort 2210/2410 Series

You should find the kernel module, mxuport2000.ko, under the output or driver source code directory.

Follow the steps below to deploy to the target Arm platform:

1. Copy the mxuport2000.ko to the path

   /lib/modules/`uname -r`/kernel/drivers/char on the ARM platform.

2. Boot into the ARM platform and load the driver.

```
# modprobe mxuport2000
```

1.3 Porting to Raspberry Pi OS

Raspberry Pi OS images are prebuilt by www.raspberrypi.org. You can install the image and start up the system. The process to build the UPort Series driver is the same as with x86 Linux. Please refer to readme.txt to check the system requirements.

You may use the rpi-source to install the kernel source packages for a more convenient option. Please refer to the official website https://github.com/notro/rpi-source/wiki for more information.

rpi-source is a third-party package offering an integrated kernel resource for building a driver. The UPort is tested with this package to see if it works well. However, the requirements may vary for different Raspberry Pi OS versions. Please read the manual of rpi-source to understand the know-how and the limitations.

1.4 Porting to the Yocto Project on Raspberry Pi

1.4.1 Prerequisite

You are expected to be familiar with the Yocto Project. Please refer to https://docs.yoctoproject.org for the Yocto Project documentation for further understanding. Also, it is encouraged to follow the procedures in this guide unless you have sufficient knowledge about the UPort driver, the Yocto Project, and Raspberry Pi.

The branch zeus is referred to throughout in this section. Please base it on this version before reading the instructions in the Yocto Project documentation. You are required to build the Yocto image successfully with the "Yocto Project Quick Build" document.

In Yocto Project, you can select the platform you want to build. This guide installs Raspberry Pi BSP Layer as a demonstration in the following steps:

1. Suppose the Yocto is installed in /home/user/poky folder. Check out the source code of the Raspberry Pi BSP Layer.

   ```
   # cd /home/user/poky
   # git clone https://git.yoctoproject.org/cgit/cgit.cgi/meta-raspberrypi
   -b zeus
   ```

2. A meta-raspberrypi folder will be checked out now. Use the following instructions to set up Raspberry Pi BSP:

   ```
   # source oe-init-build-env
   ```

3. Use a text editor to add the following content to the configuration file './conf/local.conf'

   Add the type 'rpi-sdimg' optionally if an SD card is preferred
   IMAGE_FSTYPES="tar.bz2 ext3 rpi-sdimg"

   Change the machine name of your target
   Use raspberrypi2 for Pi 2 board
   Use raspberrypi3 for Pi 3 board
   Use raspberrypi3-64 for 64-bit Pi 3 board

   MACHINE ?= "raspberrypi2"

4. Use the text editor to add the following content to the configuration file './conf/bblayers.conf'

   Add this line '/home/user/poky/meta-raspberrypi' to BBLAYERS
   BBLAYERS ?= " \
   /home/user/poky/meta \
   /home/user/poky/meta-poky \
   /home/user/poky/meta-yocto-bsp \
   /home/user/poky/meta-raspberrypi \
   "

5. Build the target core-image-base by following this command and the Raspberry Pi image will be generated.

   ```
   # bitbake core-image-base
   ```

Once the above image runs on Raspberry Pi, go to the next section.

1.4.2 Create a Moxa layer for the Yocto Project

Introduction

Moxa UPort driver is packaged as a layer for Yocto. You can add or remove the driver by modifying BBLAYERS attribute in the bblayers.conf file.

The following sections describe how to create a meta-moxa layer for the zeus branch. Note that the process may vary if your target uses a different branch. Please refer to Yocto's manual for complete information.

An example is also available in the examples folder in the UPort Series driver.

You may follow the subsequent procedures to create the same meta-moxa layer.

Create an empty Moxa layer

Use the following commands to create an empty layer, named meta-moxa.

1. Initiate the environment first. Suppose the project is installed in /home/user/poky.

```
$ cd /home/user/poky
$ source oe-init-build-env
```

2. The above commands changed the directory to the built directory. Now, we change the directory back to the Yocto root directory.

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

3. Create meta-moxa:

   A message appears reminding you to add the layer later.

```
$ bitbake-layers create-layer meta-moxa
NOTE: Starting bitbake server...
Add your new layer with 'bitbake-layers add-layer meta-moxa'
```

   The meta-moxa directory will be created in /home/user/poky.

```
$ tree meta-moxa
```

```
meta-moxa
├── conf
│   └── layer.conf
├── COPYING.MIT
├── README
└── recipes-example
    └── example
        └── example_0.1.b
```

   The “recipes-example” folder is not necessary; it may be deleted at any time.

Create a recipe for the UPort driver

Use the following commands to create a recipe for installing the UPort driver kernel to the target platform.

1. Create a directory recipes-kernel in meta-moxa.

```
$ cd /home/user/poky
$ mkdir meta-moxa/recipes-kernel
```

2. The simplest way is to copy and modify from a hello example, which is available in the Yocto source code.

```
$ cp -r ./meta-skeleton/recipes-kernel/hello-mod ./meta-moxa/recipes-kernel
```

   The content of meta-moxa is now listed below:

```
$ tree meta-moxa
```

```
meta-moxa/
├── conf
│   └── layer.conf
├── COPYING.MIT
```

```
├── README
└── recipes-kernel
    └── hello-mod
        ├── files
        │   ├── COPYING
        │   ├── hello.c
        │   └── Makefile
        └── hello-mod_0.1.bb
```

3. Delete the unnecessary files in hello-mod and rename the hello-mod.

For the UPort 1100 Series

```
$ cd ./meta-moxa/recipes-kernel
$ rm ./hello-mod/files/COPYING
$ rm ./hello-mod/files/hello.c
$ mv ./hello-mod/hello-mod_0.1.bb ./hello-mod/uport1100_0.1.bb
$ mv ./hello-mod uport1100
```

For the UPort 1200/1400/1600 Series

- mxuport

```
$ cd ./meta-moxa/recipes-kernel
$ rm ./hello-mod/files/COPYING
$ rm ./hello-mod/files/hello.c
$ mv ./hello-mod/hello-mod_0.1.bb ./hello-mod/mxuport_0.1.bb
$ mv ./hello-mod mxuport
```

- mxusbserial

```
$ cd ./meta-moxa/recipes-kernel
$ rm ./hello-mod/files/COPYING
$ rm ./hello-mod/files/hello.c
$ mv ./hello-mod/hello-mod_0.1.bb ./hello-mod/mxusbserial_0.1.bb
$ mv ./hello-mod mxusbserial
```

For the UPort 2210/2410 Series

```
$ cd ./meta-moxa/recipes-kernel
$ rm ./hello-mod/files/COPYING
$ rm ./hello-mod/files/hello.c
$ mv ./hello-mod/hello-mod_0.1.bb ./hello-mod/uport2000_0.1.bb
$ mv ./hello-mod uport2000
```

4. Extract the UPort driver source code. Copy the following files into uport1100:

For the UPort 1100 Series

```
$ cp /moxa/ mxu11x0/COPYING-GPL.TXT ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mxu11x0.c ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mxu11x0.h ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mxu1110_fw.h ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mxu1130_fw.h ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mxu1131_fw.h ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mxu1150_fw.h ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mxu1151_fw.h ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mxu3001_fw.h ./uport1100/files/
```

```
$ cp /moxa/ mxu11x0/driver/mx_dist.h ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mx_ver.h ./uport1100/files/
```

For the UPort 1200/1400/1600 Series

- mxuport

```
$ cp /moxa/mxuport/COPYING-GPL.TXT ./mxuport/files/
$ cp /moxa/mxuport/driver/mxuport/mx-uport.c ./mxuport/files/
$ cp /moxa/mxuport/driver/mxuport/mx-uport.h ./mxuport/files/
$ cp /moxa/mxuport/driver/mxuport/UPort1250FW.h ./mxuport/files/
$ cp /moxa/mxuport/driver/mxuport/UPort1250IFW.h ./mxuport/files/
$ cp /moxa/mxuport/driver/mxuport/UPort1410FW.h ./mxuport/files/
$ cp /moxa/mxuport/driver/mxuport/UPort1450FW.h ./mxuport/files/
$ cp /moxa/mxuport/driver/mxuport/UPort1450IFW.h ./mxuport/files/
$ cp /moxa/mxuport/driver/mxuport/UPort1610_16FW.h ./mxuport/files/
$ cp /moxa/mxuport/driver/mxuport/UPort1610_8FW.h ./mxuport/files/
$ cp /moxa/mxuport/driver/mxuport/UPort1650_16FW.h ./mxuport/files/
$ cp /moxa/mxuport/driver/mxuport/UPort1650_8FW.h ./mxuport/files/
$ cp /moxa/mxuport/driver/mxuport/mx_ver.h ./mxuport/files/
$ cp /moxa/mxuport/driver/mxusbserial/mxusb-serial.h ./mxuport/files/
$ cp /moxa/mxuport/driver/mxusbserial/mxusb-serial-0.h
./mxuport/files/
$ cp /moxa/mxuport/driver/mxusbserial/mxusb-serial-12.h
./mxuport/files/
```

- mxusbserial

```
$ cp /moxa/mxuport/COPYING-GPL.TXT ./mxusbserial/files/
$ cp /moxa/mxuport/driver/mxusbserial/mxbus.c ./mxusbserial/files/
$ cp /moxa/mxuport/driver/mxusbserial/mxgeneric-0.c
./mxusbserial/files/
$ cp /moxa/mxuport/driver/mxusbserial/mxgeneric-12.c
./mxusbserial/files/
$ cp /moxa/mxuport/driver/mxusbserial/mxusb-serial-0.c
./mxusbserial/files/
$ cp /moxa/mxuport/driver/mxusbserial/mxusb-serial-12.c
./mxusbserial/files/
$ cp /moxa/mxuport/driver/mxusbserial/mxusb-serial-0.h
./mxusbserial/files/
$ cp /moxa/mxuport/driver/mxusbserial/mxusb-serial-12.h
./mxusbserial/files/
$ cp /moxa/mxuport/driver/mxusbserial/mxusb-serial.h
./mxusbserial/files/
```

For the UPort 2210/2410 Series

```
$ cp /moxa/mxuport2000/COPYING-GPL.TXT ./uport2000/files/
$ cp /moxa/mxuport2000/driver/mxuport2000.c ./uport2000/files/
$ cp /moxa/mxuport2000/driver/mxuport2000.h ./uport2000/files/
$ cp /moxa/mxuport2000/driver/mx_ver.h ./uport2000/files/
```

5. The content of the recipes-kernel is listed below:

For the UPort 1100 Series

```
$ tree ./
./
└── uport1100
    ├── files
    │   ├── COPYING-GPL.TXT
    │   ├── Makefile
    │   ├── mx_dist.h
    │   ├── mxu1110_fw.h
    │   ├── mxu1130_fw.h
    │   ├── mxu1131_fw.h
    │   ├── mxu1150_fw.h
    │   ├── mxu1151_fw.h
    │   ├── mxu11x0.c
    │   ├── mxu11x0.h
    │   ├── mxu3001_fw.h
    │   └── mx_ver.h
    └── uport1100_0.1.bb
```

For the UPort 1200/1400/1600 Series

- mxuport

```
$ tree ./
./
└── mxuport
    ├── files
    │   ├── Makefile
    │   ├── mx-uport.c
    │   ├── mx-uport.h
    │   ├── mxusb-serial-0.h
    │   ├── mxusb-serial-12.h
    │   ├── mxusb-serial.h
    │   ├── mx_ver.h
    │   ├── UPort1250FW.h
    │   ├── UPort1250IFW.h
    │   ├── UPort1410FW.h
    │   ├── UPort1450FW.h
    │   ├── UPort1450IFW.h
    │   ├── UPort1610_16FW.h
    │   ├── UPort1610_8FW.h
    │   ├── UPort1650_16FW.h
    │   └── UPort1650_8FW.h
    └── mxuport_0.1.bb
```

- mxusbserial

```
$ tree ./
./
└── mxusbserial
    ├── files
    │   ├── Makefile
    │   ├── mxbus.c
    │   ├── mxgeneric-0.c
    │   ├── mxgeneric-12.c
    │   ├── mxusb-serial-0.c
    │   ├── mxusb-serial-0.h
    │   ├── mxusb-serial-12.c
    │   ├── mxusb-serial-12.h
    │   └── mxusb-serial.h
    └── mxusbserial_0.1.bb
```

For the UPort 2210/2410 Series

```
$ tree ./
./
└── uport2000
    ├── files
    │   ├── COPYING-GPL.TXT
    │   ├── Makefile
    │   ├── mxuport2000.c
    │   ├── mxuport2000.h
    │   └── mx_ver.h
    └── uport2000_0.1.bb
```

6. Modify the content of 'Makefile' in the files folder as follows:

For the UPort 1100 Series

```
##### Makefile start #####
obj-m := mxu11x0.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
##### Makefile end #####
```

For the UPort 1200/1400/1600 Series

- mxuport

```
##### Makefile start #####
obj-m := mxuport.o
mxuport-objs := mx-uport.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
##### Makefile end #####
```

- mxusbserial

If the Yocto Project's kernel version is greater than or equal to 4.12.

```
##### Makefile start #####
obj-m := mxusbserial.o
mxusbserial-objs := mxbus.o mxgeneric-12.o mxusb-serial-12.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
##### Makefile end #####
```

If the Yocto Project's kernel version is lower than 4.12.

```
##### Makefile start #####
obj-m := mxusbserial.o
mxusbserial-objs := mxbus.o mxgeneric-0.o mxusb-serial-0.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
##### Makefile end #####
```

For the UPort 2210/2410 Series

```
##### Makefile start #####
obj-m := mxuport2000.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
##### Makefile end #####
```

7. Modify the content of the file '*_0.1.bb' as follows:

For the UPort 1100 Series

We have to modify the content of the file 'uport1100_0.1.bb':

```
##### uport1100_0.1.bb start #####
DESCRIPTION = "Linux kernel module for Moxa UPort 11x0 Series"
LICENSE = "GPLv3"
LIC_FILES_CHKSUM = "file://COPYING-
GPL.TXT;md5=3c34afdc3adf82d2448f12715a255122"
inherit module
SRC_URI = "file://Makefile \
          file://mx_dist.h \
          file://mxu1110_fw.h \
          file://mxu1130_fw.h \
          file://mxu1131_fw.h \
          file://mxu1150_fw.h \
          file://mxu1151_fw.h \
          file://mxu11x0.h \
          file://mxu3001_fw.h \
          file://mx_ver.h \
          file://mxu11x0.c \
          file://COPYING-GPL.TXT \
         "
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-mxu11x0"
##### uport1100_0.1.bb end #####
```


For the UPort 1200/1400/1600 Series

- mxuport

  We have to modify the content of the file 'mxuport_0.1.bb'.

```
##### mxuport_0.1.bb start #####
DESCRIPTION = "Linux kernel module for Moxa UPort 1200/1400/1600 Series"
LICENSE = "GPLv3"
LIC_FILES_CHKSUM = "file://COPYING-
GPL.TXT;md5=3c34afdc3adf82d2448f12715a255122"
inherit module
SRC_URI = "file://Makefile \
          file://mx-uport.c \
          file://mx-uport.h \
          file://mx_ver.h \
          file://UPort1250FW.h \
          file://UPort1250IFW.h \
          file://UPort1410FW.h \
          file://UPort1450FW.h \
          file://UPort1450IFW.h \
          file://UPort1610_16FW.h \
          file://UPort1610_8FW.h \
          file://UPort1650_16FW.h \
          file://UPort1650_8FW.h \
          file://mxusb-serial.h \
          file://mxusb-serial-0.h \
          file://mxusb-serial-12.h \
          file://COPYING-GPL.TXT \
         "
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-mxuport"
##### mxuport_0.1.bb end #####
```

- mxusbserial

  We have to modify the content of the file 'mxusbserial_0.1.bb':

```
##### mxusbserial_0.1.bb start #####
DESCRIPTION = "Linux kernel module for Moxa UPort 1200/1400/1600 Series usb
serial"
LICENSE = "GPLv3"
LIC_FILES_CHKSUM = "file://COPYING-
GPL.TXT;md5=3c34afdc3adf82d2448f12715a255122"
inherit module
SRC_URI = "file://Makefile \
          file://mxbus.c \
          file://mxgeneric-0.c \
          file://mxgeneric-12.c \
```



```
                file://mxusb-serial-0.c \
                file://mxusb-serial-12.c \
                file://mxusb-serial-0.h \
                file://mxusb-serial-12.h \
                file://mxusb-serial.h \
                file://COPYING-GPL.TXT \
               "
    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-mxusbserial"
    ##### mxusbserial_0.1.bb end #####
```

For the UPort 2210/2410 Series

We have to modify the content of the file 'uport2000_0.1.bb':

```
##### uport2000_0.1.bb start #####
DESCRIPTION = "Linux kernel module for Moxa UPort 2210/2410 Series"
LICENSE = "GPLv3"
LIC_FILES_CHKSUM = "file://COPYING-
GPL.TXT;md5=3c34afdc3adf82d2448f12715a255122"
nherit module
SRC_URI = "file://Makefile \
          file://mxuport2000.h \
          file://mx_ver.h \
          file://mxuport2000.c \
          file://COPYING-GPL.TXT \
         "
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-uport2000"
##### uport2000_0.1.bb end #####
```

8. If you are using the UPort 1200/1400/1600 Series driver, you have to modify the content of the file 'mx-uport.c'.

   Replace the following line

   ```
   #include "../mxusbserial/mxusb-serial.h"
   ```

   by

   ```
   #include "mxusb-serial.h"
   ```

Create a recipe for the UPort driver utilities

If you are using the UPort 2210/2410 Series driver, it contains a utility. Similar to creating a uport2000-kernel recipe, create a recipe for facilitating the UPort 2210/2410 driver's utility.

1. Create directory below in meta-moxa:

```
$ cd /home/user/poky
$ mkdir -p ./meta-moxa/recipes-utility/uport2000-tools/files
```

2. Copy the utility source code from UPort 2210/2410 driver utility folder.

```
$ cp /moxa/mxuport2000/COPYING-GPL.TXT ./meta-moxa/recipes-
utility/uport2000-tools/files/
$ cp /moxa/mxuport2000/utility/conftool/conftool.c ./meta-moxa/recipes-
utility/uport2000-tools/files/
```

3. Create a bb file ./meta-moxa/recipes-utility/uport2000-tools/uport2000-tools.bb, which has the following content:

```
##### uport2000-tools.bb start #####
DESCRIPTION = "Utilities for UPort 2210/2410 Series"
LICENSE = "GPLv3"
LIC_FILES_CHKSUM = "file://COPYING-
GPL.TXT;md5=3c34afdc3adf82d2448f12715a255122"
SRC_URI = " \
          file://conftool.c \
          file://COPYING-GPL.TXT \
         "
S = "${WORKDIR}"
DEST_DIR = "${D}${libdir}/mxuport2000/driver"
FILES_${PN} += "${libdir}/mxuport2000/driver/*"
do_compile () {
          ${CC} -o mxustty conftool.c
}
do_install () {
          install -m 0755 -d ${DEST_DIR}
          install -m 0755 ${S}/mxustty ${DEST_DIR}
          cp ${S}/mxustty ${DEST_DIR}/mxustty
}
INSANE_SKIP_${PN} = "ldflags"
##### uport2000-tools.bb end #####
```

4. The content of the recipes-utility is listed below:

```
$ tree recipes-utility
```

```
recipes-utility
└── uport2000-tools
    ├── files
    │   ├── conftool.c
    │   └── COPYING-GPL.TXT
    └── uport2000-tools.bb
```


1.4.3 Install a Moxa layer into the Yocto Project

1. Install the Moxa layer and UPort driver recipes into the Yocto Project.

   ```
   $ cd /home/user/poky
   $ source oe-init-build-env
   ```

   Use a text editor to add the following content to the configuration file: './conf/bblayers.conf'

   Add this line '/home/user/poky/meta-moxa' to BBLAYERS
   BBLAYERS ?= " \
   /home/user/poky/meta \
   /home/user/poky/meta-poky \
   /home/user/poky/meta-yocto-bsp \
   /home/user/poky/meta-raspberrypi \
   /home/user/poky/meta-moxa \

2. Use a text editor to add the following content to the configuration file: './conf/local.conf'.

   For the UPort 1100 Series
   IMAGE_INSTALL_append += " uport1100"

   For the UPort 1200/1400/1600 Series
   IMAGE_INSTALL_append += " mxuport mxusbserial"

   For the UPort 2210/2410 Series
   MAGE_INSTALL_append += " uport2000 uport2000-tools"

1.4.4 Deploy the Yocto image in Raspberry Pi

Build the image with the UPort driver.

```
$ cd /home/user/poky
$ source oe-init-build-env
$ bitbake core-image-base
```

A SD-card format image (.rpi-sdimg) is generated under /home/user/poky/build/tmp/deploy/images/raspberrypi2. It is suggested to use the Raspberry Pi official tool 'rpi-imager' to burn the image into the SD-card and then boot it into the Linux kernel in Raspberry Pi.

1.4.5 Start the UPort driver in Raspberry Pi

After logging into the system, start the UPort driver

For the UPort 1100 Series

```
root@raspberrypi2:~# modprobe mxu11x0
```

For the UPort 1200/1400/1600 Series

```
root@raspberrypi2:~# modprobe mxusbserial
root@raspberrypi2:~# modprobe mxuport
```

For the UPort 2210/2410 Series

Start the UPort 2210/2410 Series driver.

```
root@raspberrypi2:~# modprobe mxuport2000
```

Use mxustty to locate ttyUSB0.

```
root@raspberrypi2:~# cd /usr/lib/mxuport2000/driver
root@raspberrypi2:~# ./mxustty -l /dev/ttyUSB0
mxustty: Open device /dev/ttyUSB0 ok.
mxustty: Locating device...
mxustty: Press Enter to Stop the Buzzer or LED...
```

1.4.6 Troubleshooting

If the following error is encountered during the building of the image,

ERROR: Task (/home/user/poky/meta/recipes-devtools/binutils/binutils_2.34.bb:do_compile) failed with exit code '1'

It is suggested to compile binutils first, then compile the entire image:

```
$ bitbake binutils -c do_compile
$ bitbake core-image-base
```

2 For Linux Kernel 5.x

2.1 Introduction

This document details porting the Moxa UPort driver to a specified Arm-based platform. The following knowledge is recommended before reading the instructions in this guide.

- Linux Kernel Programming
- ARM Platform Compiler
- Yocto Project Document
- MOXA UC-Series Manual
- Raspberry Pi Manual

Instructions in this guide use examples of porting on the Moxa UC-Series Arm platform and Raspberry Pi. You can apply the experience of porting UPort driver to other platforms.

The UPort driver fully supports all modern-day Linux distributions running on x86 environments, and the driver core is also compatible with the Arm platform. This document will guide you on how to port the UPort driver core.

However, some platform-dependent services, such as installer, are not available. You may refer to the platform's documentation to fulfill the requirements.

2.2 Porting to Moxa UC-Series - Arm-based Computer

2.2.1 Build binaries on a general ARM platform

If your platform is powerful and consists of the necessary development tools, the driver can be built on the platform directly. You can refer to readme.txt of the UPort driver to understand the requirement.

The step of building this driver in an ARM environment is the same as in x86 and x64 environments.

```
# tar zxvf [DRIVER NAME].tgz
# cd mxu11x0
# ./mxinstall
```

2.2.2 Cross-compiler and the UPort driver

Note To cross-compile on a x86 or x64 Linux host, the target ARM environment's kernel source package and cross compiler toolchain must be installed first.

After installing and configuring the kernel source package and toolchain, you need to compile the source code with the kernel source package and toolchain.

V5.1 driver is for the Linux kernel source 5. In the following example, we demonstrate the cross-compiler on the Moxa UC-Series ARM-based computer, which runs kernel source 4. However, the steps are identical. You can refer to the product's manual for further detail.

1. Download the kernel source package webpage under the product page.

   ```
   $ git clone https://github.com/Moxa-Linux/am335x-linux-4.4
   ```

   You can use the following commands to show the git tag list and check out the tag of the specific UC device and firmware version.

   Show the tag list:

   ```
   $ git tag
   ```

   Check out to the specific tag:

   ```
   $ git checkout UC-2100_V1.7
   ```
   ← Replace the tag name, UC-2100_V1.7, with the UC Series that is being used.

2. Download the toolchain from the product's webpage. The toolchain, which is used by the UC Series, is arm-linux-gnueabihf. It is a script that will install the related packages. Execute the script and follow the steps to install the Linux cross-compiler tools. You will need the root privilege to install the toolchain and the kernel source.

   ```
   # sh arm-linux-gnueabihf_6.3_Build_amd64_<build_date>.sh
   ```

   If the script shows the notification message: "Please export these environment variables before using the toolchain", enter the following script command:

   ```
   # export PATH=$PATH:/usr/local/arm-linux-gnueabihf-6.3/usr/bin
   ```

3. The kernel source, which is used by the UC Series, is am335x-linux-4.4. You need to configure these files before starting to cross-compile.

   Move the kernel source to /moxa/kernel and configure the kernel source.

   ```
   # mv am335x-linux-4.4 /moxa/kernel
   # cd /moxa/kernel
   # make uc2100_defconfig ← Replace the uc2100 with the UC Series
                             that is being used.
   # make modules_prepare
   ```

   After the abovementioned steps, please follow the processes as set out in Section 2.3, "Moxa cross-compiling interactive script," and Section 2.4, "Manually build the UPort driver with a cross-compiler," to cross-compile Moxa's driver for the UC-Series platforms.

   The UPort driver, which includes the driver modules, needs to be compiled. The file of each UPort Series is listed as follow:

   UPort 1100 Series driver

   - mxu11x0.ko

   UPort 1200/1400/1600 Series driver

   - mxuport.ko

   UPort 2210/2410 Series driver

   - mxuport2000.ko

   If it is preferred to build these binaries with automatic script, please refer to Section 2.3, "Moxa cross-compiling interactive script." If you find the build script troublesome, or you prefer to build these binaries manually, please refer to Section 2.4 "Manually build the UPort driver with a cross-compiler."

   If you have generated the necessary binaries, please refer to section 2.5 to deploy to the target platform.

2.2.3 Moxa cross-compiling interactive script

To simplify the processes above, Moxa has provided an interactive script, "mxcc", to cross-compile these drivers. You may execute ./mxcc in the UPort driver source directory to cross-compile the MOXA driver.

For the UPort 1100 Series

The steps are as follows:

```
#./mxcc
Enter target device architecture (ARCH) [arm]:
Enter cross-compile (CROSS_COMPILE) [arm-linux-gnueabihf-]:
Enter target device kernel source directory [/moxa/kernel]:
****************************************
mxu11x0 cross-compile is success.
****************************************

**************************************************************************
MOXA UPort 1100 Series driver cross-compile finished.
```



```
If cross compile is success, driver is outputted to the output folder.
**************************************************************************
```

The binaries will now be generated and placed in the output directory under the /moxa/mxu11x0 folder.

For the UPort 1200/1400/1600 Series

The steps are as follows:

```
# ./mxcc
Enter target device architecture (ARCH) [arm]:
Enter cross-compile (CROSS_COMPILE) [arm-linux-gnueabihf-]:
Enter target device kernel source directory [/moxa/kernel/]:
****************************************
mxuport cross-compile is success.
****************************************
**************************************************************************
MOXA UPort 1200/1400/1600 Series driver cross-compile finished.
If cross compile is success, driver is outputted to the output folder.
**************************************************************************
```

The binaries will now be generated and placed in the output directory under the /moxa/mxuport folder.

For the UPort 2210/2410 Series

The steps are as follows:

```
Enter target device architecture (ARCH) [arm]:
Enter cross-compile (CROSS_COMPILE) [arm-linux-gnueabihf-]:
Enter target device kernel source directory [/moxa/kernel/]:
****************************************
mxuport2000 cross-compile is success.
****************************************
**************************************************************************
MOXA UPort 2210/2410 Series driver cross-compile finished.
If cross compile is success, driver is outputted to the output folder.
**************************************************************************
```

The binaries will now be generated and placed in the output directory under the /moxa/mxuport2000 folder.

2.2.4 Manually build the UPort driver with a cross-compiler

For the UPort 1100 Series

To cross-compile the UPort 1100 Series driver, you can find "Makefile" in the driver folder, and then run it.

```
# make ARCH=<ARCH> CROSS_COMPILE=<CROSS_COMPILE> KDIR=<KERNEL_SOURCE>
KVER_MAJOR=<KERNEL_MAJOR> KVER_MINOR=<KERNEL_MINOR>
```

<ARCH>: The target ARM environment device's CPU architecture. For example, arm, arm64.

<CROSS_COMPILE>: The cross-compile the toolchain path. If the toolchain is arm-linux-gnueabihf, and you have already added the path of the toolchain to the environment variable, you should enter "arm-linux-gnueabihf-" here.

<KERNEL_SOURCE>: The directory of the target kernel source.

<KERNEL_MAJOR>: The major version of the target ARM system's kernel source. You can use the command "make kernelversion" to get the kernel source's major version.

e.g.,

```
# make kernelversion
4.4.0
|
+--- kernel major version
```

<KERNEL_MINOR>: The minor version of the target ARM system's kernel source. You can use the command "make kernelversion" to get the kernel source's minor version.

e.g.,

```
# make kernelversion
4.4.0
  |
  +--- kernel minor version
```

The "make" command would be similar to the following example:

```
# make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- KDIR=/moxa/kernel
KVER_MAJOR=4 KVER_MINOR=4
```

After using the "make" command to cross-compile the drivers, the driver files "mxu11x0.ko" can be found in the /moxa/mxu11x0/driver directory.

For the UPort 1200/1400/1600 Series

To cross-compile the UPort 1200/1400/1600 Series driver, you can find "Makefile" in the driver/mxuport folder, and then run it.

Build mxuport.ko.

Run the Makefile in the /moxa/mxuport/driver/mxuport folder.

```
# make ARCH=<ARCH> CROSS_COMPILE=<CROSS_COMPILE> KDIR=<KERNEL_SOURCE>
KVER_MAJOR=<KERNEL_MAJOR> KVER_MINOR=<KERNEL_MINOR>
```

<ARCH>: The target ARM environment device's CPU architecture. For example, arm, arm64.

<CROSS_COMPILE>: The cross-compile toolchain path. If the toolchain is arm-linux-gnueabihf, and you have already added the path of the toolchain to the environment variable, you should enter "arm-linux-gnueabihf-" here.

<KERNEL_SOURCE>: The directory of the target kernel source.

<KERNEL_MAJOR>: The major version of the target ARM system's kernel source. You can use the command "make kernelversion" to get the kernel source's major version.

e.g.,

```
# make kernelversion
4.4.0
|
+--- kernel major version
```

<KERNEL_MINOR>: The minor version of the target ARM system's kernel source. You can use the command "make kernelversion" to get the kernel source's minor version.

e.g.,

```
# make kernelversion
4.4.0
  |
  +--- kernel minor version
```

The "make" command would be similar to the following example:

```
# make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- KDIR=/moxa/kernel
KVER_MAJOR=4 KVER_MINOR=4
```

After using the "make" command to cross-compile the drivers, the driver files "mxuport.ko" can be found in the /moxa/mxuport/driver/mxuport directory.

For the UPort 2210/2410 Series

To cross-compile UPort 2210/2410 Series driver, you can find "Makefile" in the driver folder, and then run it.

```
# make ARCH=<ARCH> CROSS_COMPILE=<CROSS_COMPILE> KDIR=<KERNEL_SOURCE>
KVER_MAJOR=<KERNEL_MAJOR> KVER_MINOR=<KERNEL_MINOR>
```

<ARCH>: The target ARM environment device's CPU architecture. For example, arm, arm64.

<CROSS_COMPILE>: The cross-compile the toolchain path. If the toolchain is arm-linux-gnueabihf, and you have already added the path of the toolchain to the environment variable, you should enter "arm-linux-gnueabihf-" here.

<KERNEL_SOURCE>: The directory of the target kernel source.

<KERNEL_MAJOR>: The major version of the target ARM system's kernel source. You can use the command "make kernelversion" to get the kernel source's major version.

e.g.,

```
# make kernelversion
4.4.0
|
+--- kernel major version
```

<KERNEL_MINOR>: The minor version of the target ARM system's kernel source. You can use the command "make kernelversion" to get the kernel source's minor version.

e.g.,

```
# make kernelversion
4.4.0
  |
  +--- kernel minor version
```

The "make" command would be similar to the following example:

```
# make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- KDIR=/moxa/kernel
KVER_MAJOR=4 KVER_MINOR=4
```

After using the "make" command to cross-compile the drivers, the driver files "mxuport2000.ko" can be found in the /moxa/mxuport2000/driver directory.

To cross-compile the UPort 2210/2410 Series driver's utility, you can follow the following steps:

You can find the source in the utility folder.

```
# cd /moxa/mxuport2000/utility/conftool/
```

Use the cross compiler to compile the utility.

```
# <CROSS_COMPILE>gcc -o mxustty conftool.c
```

After using the cross compiler to compile the utility, you can find the utility execution 'mxustty' in /moxa/mxuport2000/utility/conftool/

2.2.5 Deploy cross-compiled binary to target

For the UPort 1100 Series

You should find the kernel module, mxu11x0.ko, under the output or driver source code directory.

Follow the steps below to deploy to the target Arm platform.

1. Copy the mxu11x0.ko to the path

   /lib/modules/`uname -r`/kernel/drivers/char on the ARM platform.

2. Boot into the ARM platform and load the driver.

   ```
   # modprobe mxu11x0
   ```

For the UPort 1200/1400/1600 Series

You should find the kernel module, mxuport.ko, under the output or driver source code directory.

Follow the steps below to deploy to the target Arm platform.

1. Copy the mxuport.ko to the path

   /lib/modules/`uname -r`/kernel/drivers/char on the ARM platform

2. Boot into the ARM platform and load the driver.

   ```
   # modprobe mxuport
   ```

For the UPort 2210/2410 Series

You should find the kernel module, mxuport2000.ko, under the output or driver source code directory.

Follow the steps below to deploy to the target Arm platform.

1. Copy the mxuport2000.ko to the path

   /lib/modules/`uname -r`/kernel/drivers/char on the ARM platform.

2. Boot into the ARM platform and load the driver.

   ```
   # modprobe mxuport2000
   ```

2.3 Porting to Raspberry Pi OS

Raspberry Pi OS images are prebuilt by www.raspberrypi.org. You can install the image and start up the system. The process to build the UPort Series driver is the same as with x86 Linux. Please refer to readme.txt to check the system requirements.

You may use the rpi-source to install the kernel source packages for a more convenient option. Please refer to the official website https://github.com/notro/rpi-source/wiki for more information.

rpi-source is a third-party package offering an integrated kernel resource for building a driver. The UPort is tested with this package to see if it works well. However, the requirements may vary for different Raspberry Pi OS versions. Please read the manual of rpi-source to understand the know-how and the limitations.

2.4 Porting to the Yocto Project on Raspberry Pi

2.4.1 Prerequisite

You are expected to be familiar with the Yocto Project. Please refer to https://docs.yoctoproject.org for the Yocto Project documentation for further understanding. Also, it is encouraged to follow the procedures in this guide unless you have sufficient knowledge about the UPort driver, the Yocto Project, and Raspberry Pi.

The dunfell branch (3.1.9) is referred to throughout in this section. Please base it on this version before reading the instructions in the Yocto Project documentation. You are required to build the Yocto image successfully with the "Yocto Project Quick Build" document.

In Yocto Project, you can select the platform you want to build. This guide installs Raspberry Pi BSP Layer as a demonstration in the following steps:

1. Suppose the Yocto is installed in /home/user/poky folder. Check out the source code of the Raspberry Pi BSP Layer.

   ```
   # cd /home/user/poky
   # git clone https://git.yoctoproject.org/cgit/cgit.cgi/meta-raspberrypi
   -b dunfell
   ```


2. A meta-raspberrypi folder will be checked out now. Use the following instructions to set up Raspberry Pi BSP:

   ```
   # source oe-init-build-env
   ```

3. Use a text editor to add the following content to the configuration file './conf/local.conf'

   Add the type 'rpi-sdimg' optionally if an SD card is preferred
   IMAGE_FSTYPES="tar.bz2 ext3 rpi-sdimg"

   Change the machine name of your target
   Use raspberrypi2 for Pi 2 board
   Use raspberrypi3 for Pi 3 board
   Use raspberrypi3-64 for 64-bit Pi 3 board

   MACHINE ?= "raspberrypi2"

4. Use the text editor to add the following content to the configuration file './conf/bblayers.conf'

   Add this line '/home/user/poky/meta-raspberrypi' to BBLAYERS
   BBLAYERS ?= " \
   /home/user/poky/meta \
   /home/user/poky/meta-poky \
   /home/user/poky/meta-yocto-bsp \
   /home/user/poky/meta-raspberrypi \
   "

5. Build the target core-image-base by following this command and the Raspberry Pi image will be generated:

   ```
   # bitbake core-image-base
   ```

Once the above image runs on Raspberry Pi, go to the next section.

2.4.2 Create a Moxa layer for the Yocto Project

Introduction

The Moxa UPort driver is packaged as a layer for Yocto. You can add or remove the driver by modifying BBLAYERS attribute in the bblayers.conf file.

The following sections describe how to create the meta-moxa layer for the dunfell branch (3.1.9). Note that the process may vary if your target uses a different branch. Please refer to Yocto's manual for complete information.

An example is also available in the examples folder in the UPort Series driver.

You may follow the subsequent procedures to create the same meta-moxa layer.

Create an empty Moxa layer

Use the following commands to create an empty layer, named meta-moxa.

1. Initiate the environment first. Suppose the project is installed in /home/user/poky:

   ```
   $ cd /home/user/poky
   $ source oe-init-build-env
   ```


2. The above commands changed the directory to the built directory. Now, we change the directory back to the Yocto root directory.

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

3. Create meta-moxa:

   A message appears reminding you to add the layer later.

```
$ bitbake-layers create-layer meta-moxa
NOTE: Starting bitbake server...
Add your new layer with 'bitbake-layers add-layer meta-moxa'
```

   The meta-moxa directory will be created in /home/user/poky.

```
$ tree meta-moxa
```

```
meta-moxa
├── conf
│   └── layer.conf
├── COPYING.MIT
├── README
└── recipes-example
    └── example
        └── example_0.1.bb
```

   The "recipes-example" folder is not necessary; it may be deleted at any time.

Create a recipe for the UPort driver

Use the following commands to create a recipe for installing the UPort driver kernel to the target platform.

1. Create a directory recipes-kernel in meta-moxa:

```
$ cd /home/user/poky
$ mkdir meta-moxa/recipes-kernel
```

2. The simplest way is to copy and modify from a hello example, which is available in the Yocto source code.

```
$ cp -r ./meta-skeleton/recipes-kernel/hello-mod ./meta-moxa/recipes-kernel
```

   The content of meta-moxa now is listed below:

```
$ tree meta-moxa
```

```
meta-moxa/
├── conf
│   └── layer.conf
├── COPYING.MIT
├── README
└── recipes-kernel
    └── hello-mod
        ├── files
        │   ├── COPYING
        │   ├── hello.c
```

```
|   └── Makefile
└── hello-mod_0.1.bb
```

3. Delete the unnecessary files in hello-mod and rename the hello-mod.

   For the UPort 1100 Series

   ```
   $ cd ./meta-moxa/recipes-kernel
   $ rm ./hello-mod/files/COPYING
   $ rm ./hello-mod/files/hello.c
   $ mv ./hello-mod/hello-mod_0.1.bb ./hello-mod/uport1100_0.1.bb
   $ mv ./hello-mod uport1100
   ```

   For the UPort 1200/1400/1600 Series

   ```
   $ cd ./meta-moxa/recipes-kernel
   $ rm ./hello-mod/files/COPYING
   $ rm ./hello-mod/files/hello.c
   $ mv ./hello-mod/hello-mod_0.1.bb ./hello-mod/mxuport_0.1.bb
   $ mv ./hello-mod mxuport
   ```

   For the UPort 2210/2410 Series

   ```
   $ cd ./meta-moxa/recipes-kernel
   $ rm ./hello-mod/files/COPYING
   $ rm ./hello-mod/files/hello.c
   $ mv ./hello-mod/hello-mod_0.1.bb ./hello-mod/uport2000_0.1.bb
   $ mv ./hello-mod uport2000
   ```

4. Extract the UPort driver source code. Copy the following files into uport1100:

   For the UPort 1100 Series

   ```
   $ cp /moxa/ mxu11x0/COPYING-GPL.TXT ./uport1100/files/
   $ cp /moxa/ mxu11x0/driver/mxu11x0.c ./uport1100/files/
   $ cp /moxa/ mxu11x0/driver/mxu11x0.h ./uport1100/files/
   $ cp /moxa/ mxu11x0/driver/mxu1110_fw.h ./uport1100/files/
   $ cp /moxa/ mxu11x0/driver/mxu1130_fw.h ./uport1100/files/
   $ cp /moxa/ mxu11x0/driver/mxu1131_fw.h ./uport1100/files/
   $ cp /moxa/ mxu11x0/driver/mxu1150_fw.h ./uport1100/files/
   $ cp /moxa/ mxu11x0/driver/mxu1151_fw.h ./uport1100/files/
   $ cp /moxa/ mxu11x0/driver/mxu3001_fw.h ./uport1100/files/
   $ cp /moxa/ mxu11x0/driver/usb-serial.h ./uport1100/files/
   $ cp /moxa/ mxu11x0/driver/mx_ver.h ./uport1100/files/
   ```

   For the UPort 1200/1400/1600 Series

   ```
   $ cp /moxa/mxuport/COPYING-GPL.TXT ./mxuport/files/
   $ cp /moxa/mxuport/driver/mxuport/mx-uport.c ./mxuport/files/
   $ cp /moxa/mxuport/driver/mxuport/mx-uport.h ./mxuport/files/
   $ cp /moxa/mxuport/driver/mxuport/UPort1250FW.h ./mxuport/files/
   $ cp /moxa/mxuport/driver/mxuport/UPort1250IFW.h ./mxuport/files/
   $ cp /moxa/mxuport/driver/mxuport/UPort1410FW.h ./mxuport/files/
   $ cp /moxa/mxuport/driver/mxuport/UPort1450FW.h ./mxuport/files/
   $ cp /moxa/mxuport/driver/mxuport/UPort1450IFW.h ./mxuport/files/
   $ cp /moxa/mxuport/driver/mxuport/UPort1610_16FW.h ./mxuport/files/
   $ cp /moxa/mxuport/driver/mxuport/UPort1610_8FW.h ./mxuport/files/
   $ cp /moxa/mxuport/driver/mxuport/UPort1650_16FW.h ./mxuport/files/
   ```

```
$ cp /moxa/mxuport/driver/mxuport/UPort1650_8FW.h ./mxuport/files/
$ cp /moxa/mxuport/driver/mxuport/mx_ver.h ./mxuport/files/
```

For the UPort 2210/2410 Series

```
$ cp /moxa/mxuport2000/COPYING-GPL.TXT ./uport2000/files/
$ cp /moxa/mxuport2000/driver/mxuport2000.c ./uport2000/files/
$ cp /moxa/mxuport2000/driver/mxuport2000.h ./uport2000/files/
$ cp /moxa/mxuport2000/driver/mx_ver.h ./uport2000/files/
```

5. The content of the recipes-kernel is listed below:

For the UPort 1100 Series:

```
$ tree ./
./
└── uport1100
    ├── files
    │   ├── COPYING-GPL.TXT
    │   ├── Makefile
    │   ├── mxu1110_fw.h
    │   ├── mxu1130_fw.h
    │   ├── mxu1131_fw.h
    │   ├── mxu1150_fw.h
    │   ├── mxu1151_fw.h
    │   ├── mxu11x0.c
    │   ├── mxu11x0.h
    │   ├── mxu3001_fw.h
    │   ├── mx_ver.h
    │   └── usb-serial.h
    └── uport1100_0.1.bb
```

For the UPort 1200/1400/1600 Series:

```
$ tree ./
./
└── mxuport
    ├── files
    │   ├── COPYING-GPL.TXT
    │   ├── Makefile
    │   ├── mx-uport.c
    │   ├── mx-uport.h
    │   ├── mx_ver.h
    │   ├── UPort1250FW.h
    │   ├── UPort1250IFW.h
    │   ├── UPort1410FW.h
    │   ├── UPort1450FW.h
    │   ├── UPort1450IFW.h
    │   ├── UPort1610_16FW.h
```



```
│       ├── UPort1610_8FW.h
│       ├── UPort1650_16FW.h
│       └── UPort1650_8FW.h
└── mxuport_0.1.bb
```

For the UPort 2210/2410 Series:

```
$ tree ./
./
└── uport2000
    ├── files
    │   ├── COPYING-GPL.TXT
    │   ├── Makefile
    │   ├── mxuport2000.c
    │   ├── mxuport2000.h
    │   └── mx_ver.h
    └── uport2000_0.1.bb
```

6. Modify the content of 'Makefile' in the files folder as follows:

For the UPort 1100 Series

```
##### Makefile start #####
obj-m := mxu11x0.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
##### Makefile end ####
```

For the UPort 1200/1400/1600 Series

```
##### Makefile start #####
obj-m := mxuport.o
mxuport-objs := mx-uport.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
##### Makefile end #####
```

For the UPort 2210/2410 Series

```
##### Makefile start #####
obj-m := mxuport2000.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
##### Makefile end #####
```

7. Modify the content of the file '*_0.1.bb' as follows:

   For the UPort 1100 Series

   We have to modify the content of the file 'uport1100_0.1.bb':

```
##### uport1100_0.1.bb start #####
DESCRIPTION = "Linux kernel module for Moxa UPort 11x0 Series"
LICENSE = "GPLv3"
LIC_FILES_CHKSUM = "file://COPYING-
GPL.TXT;md5=
3c34afdc3adf82d2448f12715a255122"
inherit module
SRC_URI = "file://Makefile \
          file://mxu1110_fw.h \
          file://mxu1130_fw.h \
          file://mxu1131_fw.h \
          file://mxu1150_fw.h \
          file://mxu1151_fw.h \
          file://mxu11x0.h \
          file://mxu3001_fw.h \
          file://mx_ver.h \
          file://usb-serial.h \
          file://mxu11x0.c \
          file://COPYING-GPL.TXT \
         "
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-mxu11x0"
##### uport1100_0.1.bb end #####
```

   For the UPort 1200/1400/1600 Series

   We have to modify the content of the file 'mxuport_0.1.bb':

```
##### mxuport_0.1.bb start #####
DESCRIPTION = "Linux kernel module for Moxa UPort 1200/1400/1600 Series"
LICENSE = "GPLv3"
```

```
LIC_FILES_CHKSUM = "file://COPYING-
GPL.TXT;md5=3c34afdc3adf82d2448f12715a255122"
inherit module
SRC_URI = "file://Makefile \
          file://mx-uport.c \
          file://mx-uport.h \
          file://mx_ver.h \
          file://UPort1250FW.h \
          file://UPort1250IFW.h \
          file://UPort1410FW.h \
          file://UPort1450FW.h \
          file://UPort1450IFW.h \
          file://UPort1610_16FW.h \
          file://UPort1610_8FW.h \
          file://UPort1650_16FW.h \
          file://UPort1650_8FW.h \
          file://COPYING-GPL.TXT \
         "
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-mxuport"
##### mxuport_0.1.bb end #####
```

For the UPort 2210/2410 Series

We have to modify the content of the file 'uport2000_0.1.bb':

```
##### uport2000_0.1.bb start #####
DESCRIPTION = "Linux kernel module for Moxa UPort 2210/2410 Series"
LICENSE = "GPLv3"
LIC_FILES_CHKSUM = "file://COPYING-
GPL.TXT;md5=3c34afdc3adf82d2448f12715a255122"
inherit module
SRC_URI = "file://Makefile \
          file://mxuport2000.h \
          file://mx_ver.h \
          file://mxuport2000.c \
          file://COPYING-GPL.TXT \
         "
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-uport2000"
##### uport2000_0.1.bb end #####
```

Create a recipe for the UPort driver utilities

If you are using the UPort 2210/2410 Series driver, it contains a utility. Similar to creating a uport2000-kernel recipe, create a recipe for facilitating the UPort 2210/2410 driver's utility.

1. Create directory below in meta-moxa:

```
$ cd /home/user/poky
$ mkdir -p ./meta-moxa/recipes-utility/uport2000-tools/files
```

2. Copy the utility source code from the UPort 2210/2410 driver utility folder.

```
$ cp /moxa/mxuport2000/COPYING-GPL.TXT ./meta-moxa/recipes-utility/uport2000-tools/files/
$ cp /moxa/mxuport2000/utility/conftool/conftool.c ./meta-moxa/recipes-utility/uport2000-tools/files/
```

3. Create a bb file ./meta-moxa/recipes-utility/uport2000-tools/uport2000-tools.bb, which has the following content:

```
##### uport2000-tools.bb start #####
DESCRIPTION = "Utilities for UPort 2210/2410 Series"
LICENSE = "GPLv3"
LIC_FILES_CHKSUM = "file://COPYING-GPL.TXT;md5=3c34afdc3adf82d2448f12715a255122"
SRC_URI = " \
          file://conftool.c \
          file://COPYING-GPL.TXT \
         "
S = "${WORKDIR}"
DEST_DIR = "${D}${libdir}/mxuport2000/driver"
FILES_${PN} += "${libdir}/mxuport2000/driver/*"
do_compile () {
          ${CC} -o mxustty conftool.c
}
do_install () {
          install -m 0755 -d ${DEST_DIR}
          install -m 0755 ${S}/mxustty ${DEST_DIR}
          cp ${S}/mxustty ${DEST_DIR}/mxustty
}
INSANE_SKIP_${PN} = "ldflags"
##### uport2000-tools.bb end #####
```

4. The content of the recipes-utility is listed below:

```
$ tree recipes-utility
recipes-utility
└── uport2000-tools
    ├── files
    │   ├── conftool.c
    │   └── COPYING-GPL.TXT
    └── uport2000-tools.bb
```

2.4.3 Install a Moxa layer into the Yocto Project

1. Install the Moxa layer and UPort driver recipes into the Yocto Project.

   ```
   $ cd /home/user/poky
   $ source oe-init-build-env
   ```

   Use a text editor to add the following content to the configuration file: './conf/bblayers.conf'

   Add this line '/home/user/poky/meta-moxa' to BBLAYERS
   BBLAYERS ?= " \
   /home/user/poky/meta \
   /home/user/poky/meta-poky \
   /home/user/poky/meta-yocto-bsp \
   /home/user/poky/meta-raspberrypi \
   /home/user/poky/meta-moxa \
   "

2. Use a text editor to add the following content to the configuration file: './conf/local.conf'.

   For the UPort 1100 Series
   IMAGE_INSTALL_append += " uport1100"

   For the UPort 1200/1400/1600 Series
   IMAGE_INSTALL_append += " mxuport"

   For the UPort 2210/2410 Series
   MAGE_INSTALL_append += " uport2000 uport2000-tools"

2.4.4 Deploy the Yocto image in Raspberry Pi

Build the image with UPort driver.

```
$ cd /home/user/poky
$ source oe-init-build-env
$ bitbake core-image-base
```

A SD-card format image (.rpi-sdimg) is generated under /home/user/poky/build/tmp/deploy/images/raspberrypi2. It is suggested to use the Raspberry Pi official tool 'rpi-imager' to burn the image into the SD-card and then boot it into the Linux kernel in Raspberry Pi.

2.4.5 Start the UPort driver in Raspberry Pi

After logging into the system, start the UPort driver

For the UPort 1100 Series

```
root@raspberrypi2:~# modprobe mxu11x0
```

For the UPort 1200/1400/1600 Series

```
root@raspberrypi2:~# modprobe mxuport
```

For the UPort 2210/2410 Series

```
root@raspberrypi2:~# modprobe mxuport2000
```

Use mxustty to locate ttyUSB0.

```
root@raspberrypi2:~# cd /usr/lib/mxuport2000/driver
root@raspberrypi2:~# ./mxustty -l /dev/ttyUSB0
mxustty: Open device /dev/ttyUSB0 ok.
mxustty: Locating device...
mxustty: Press Enter to Stop the Buzzer or LED...
```

2.4.6 Troubleshooting

If the following error is encountered during the building of the image,

ERROR: Task (/home/user/poky/meta/recipes-devtools/binutils/binutils_2.34.bb:do_compile) failed with exit code '1'

It is suggested to compile binutils first, then compile the entire image:

```
$ bitbake binutils -c do_compile
$ bitbake core-image-base
```